ΤΕΙ Κρήτης

Τεχνολογικό Εκπαιδευτικό Ίδρυμα Κρήτης

Πτυχιακή εργασία

Προγραμματισμός εργαλειομηχανών αριθμητικού ελέγχου με μακροεντολές

Σπουδαστής: Μελαμπιανάκης Χαράλαμπος

Επιβλέποντες: Δρ. Πετούσης Μάρκος

Δρ. Βιδάκης Νεκτάριος

Ηράκλειο 2018

1. Ευχαριστίες

Η παρούσα εργασία εκπονήθηκε το διάστημα μεταξύ Ιανουαρίου και Ιουνίου 2018 στα πλαίσια του προπτυχιακού προγράμματος του τμήματος μηχανολογίας του Τεχνολογικού εκπαιδευτικού ιδρύματος Κρήτης.

Σε αυτό το σημείο αισθάνομαι την ανάγκη να εκφράσω τις ευχαριστίες μου σε ορισμένους ανθρώπους που η συμβολή και η συμπαράσταση τους ήταν πολύτιμη και καθοριστική στην εκπόνηση της παρούσας πτυχιακής εργασίας.

Ιδιαίτερες ευχαριστίες οφείλω στον καθηγητή και επιβλέποντα Κύριο Μάρκο Πετούση για τις πολύτιμες συμβουλές, την καθοδήγηση, τις παραγωγικές υποδείξεις και το πολύ καλό κλίμα συνεργασίας που διαμόρφωσε συμβάλλοντας τα μέγιστα για την κατάρτιση της εργασίας. Κυρίως τον ευχαριστώ διότι ήταν ο άνθρωπος που μου γνώρισε την δημιουργικότητα του κατασκευαστικού κλάδου της μηχανολογίας και ειδικότερα τον κόσμο των εργαλειομηχανών, κάτι που για μένα έπαιξε καθοριστικό ρόλο διότι συνειδητοποίησα ότι είναι το αντικείμενο που θέλω να ασχοληθώ στην μετέπειτα πορεία μου σαν μηχανικός.

Επίσης θα ήθελα να ευχαριστήσω όλους τους καθηγητές του τμήματος για τις πολύτιμες γνώσεις που μου προσέφεραν όλα αυτά τα χρόνια, το ειλικρινές ενδιαφέρον τους και την σημαντική βοήθειά τους.

Κλείνοντας ευχαριστώ ειλικρινά την οικογένεια μου για την ηθική τους στήριξη και την εμπιστοσύνη που μου έδειξε όλα αυτά τα χρόνια των σπουδών μου.

Μελαμπιανάκης Χαράλαμπος

Ηράκλειο, Ιούνιος 2018

2. Περίληψη

Ο σκοπός της εργασίας είναι να παρατεθούν τα πλεονεκτήματα του παραμετρικού προγραμματισμού όσο δυνατόν πληρέστερα καθώς επίσης να γίνει μια άμεση σύγκριση με άλλους τρόπους προγραμματισμού ώστε να εξαχθούν συμπεράσματα για την αποδοτικότητα και την παραγωγικότητα αυτής της μεθόδου. Μέσα στη μελέτη συμπεριλαμβάνονται επίσης βασικές έννοιες για την καλύτερη κατανόηση του αντικειμένου και κάποια αντιπροσωπευτικά παραδείγματα που εμφανίζουν τα βασικότερα πλεονεκτήματα και τις δυνατότητες που παρέχει στον προγραμματιστή.

Μπορεί να αποτελέσει επίσης έναν οδηγό εκμάθησης για οποιονδήποτε έχει βασικές γνώσεις και επιθυμεί να βελτιωθεί όσον αφορά των προγραμματισμό των εργαλειομηχανών γενικότερα.

Κατά τη διαδικασία της συγγραφής, σκοπός ήταν η ορθή και όσο το δυνατόν πληρέστερη ανάλυση του θέματος έτσι ώστε το περιεχόμενο της εργασίας να είναι κατανοητό και σαφές, γι' αυτό η ανάλυση του θέματος γίνεται με χρήση πληθώρας διαγραμματικών αναπαραστάσεων, παραδειγμάτων, γραφημάτων και συγκεντρωτικών πινάκων που θα μπορούν να αξιοποιηθούν από οποιονδήποτε θέλει να χρησιμοποιήσει τα τελικά συμπεράσματα που θα παραχθούν.

3. Abstract

The purpose of this thesis is the quotation of the advantages of parametric programming as well to compare directly other ways of programming in order to extract conclusions about the efficiency and the productivity of this method. In this thesis are included some basic meanings for the best comprehension of the subject and some representative examples that demonstrate the basic advantages and possibilities which are provided to the programmer.

It can be also used as a learning guide for anyone who has the basic knowledge and wishes to be improved over the computer numerical control machines programming generally.

During the writing process, the purpose was the right and the best analysis of the topic in order the content of the thesis to be understandable and clear, that's why the analysis of the topic is done with plenty of diagrammatic representations, examples. Graphs and aggregate tables which can be exploited by anyone who wants to use the final conclusions that are conducted.

4. Περιεχόμενα

5. Εισαγωγή στον προγραμματισμό εργαλειομηχανών

1. Γενικά εισαγωγικά στοιχεία για τον προγραμματισμό

Στην κατασκευαστική βιομηχανία η εξέλιξη της ψηφιακής τεχνολογίας και της επιστήμης των ηλεκτρονικών υπολογιστών παίζει πολύ σημαντικό ρόλο. Ειδικότερα, η αξιοποίηση της εξέλιξης αυτής στο χώρο των μηχανουργικών κατεργασιών, οδήγησε στην εμφάνιση των εργαλειομηχανών C.N.C.

Θεμελιακό στοιχείο της λειτουργίας των εργαλειομηχανών C.N.C. είναι το πρόγραμμα κατεργασίας. Όλες οι πληροφορίες που είναι απαραίτητες για την κατεργασία ενός συγκεκριμένου τεμαχίου συντάσσονται με καθορισμένους κανόνες που υπαγορεύονται από το αντίστοιχο σύστημα ελέγχου της εργαλειομηχανής και αποτελούν στο σύνολο τους το περιεχόμενο του προγράμματος κατεργασίας.

Απαιτείται καλά καταρτισμένο και έμπειρο προσωπικό για τη σωστή σύνταξη του προγράμματος κατεργασίας, τη χρήση όλων των δυνατοτήτων προγραμματισμού έτσι ώστε να επιτυγχάνεται η διαμόρφωση έξυπνων προγραμμάτων, αλλά και την ανάπτυξη προγραμμάτων που να στοχεύουν στην πλήρη αξιοποίηση των δυνατοτήτων των εργαλειομηχανών C.N.C

Δυστυχώς, υπάρχει έλλειψη κατάλληλα εκπαιδευμένου εργατικού δυναμικού σε αυτόν τον τομέα. Ο κύριος λόγος είναι ότι λίγοι άνθρωποι στη βιομηχανία έχουν πολύ καλές γνώσεις προγραμματισμού και τις περισσότερες φορές δεν είναι πάρα πολύ πρόθυμοι να μοιραστούν τις γνώσεις και την εμπειρία τους με άλλους, για προφανείς λόγους! Τα τελευταία χρόνια εμφανίστηκαν ορισμένα εκπαιδευτικά κέντρα για να καλυφθεί η ανάγκη αυτή όμως παρέχουν μόνο στοιχειώδεις και πολύ βασικές γνώσεις. Τα ελάχιστα ανά τον κόσμο κέντρα εκπαίδευσης στα οποία παρέχεται προχωρημένη εκπαίδευση κοστίζουν αρκετά και για έναν νέο επαγγελματία στον κλάδο συνήθως τα ποσά αυτά είναι απαγορευτικά. Υπάρχουν βεβαίως και τα εγχειρίδια των εργαλειομηχανών τα οποία μπορούν να αποτελέσουν συμβουλευτικά εργαλεία, όμως είναι πολύ δύσκολή η εκπαίδευση από ένα manual διότι ο σκοπός τους είναι να χρησιμοποιούνται σαν βιβλία αναφοράς και δεν θεωρούνται εκπαιδευτικά συγγράμματα, με αυτή τη λογική μάλιστα έχουν συγγραφεί.

5.1 Μεθόδοι προγραμματισμού εργαλειομηχανών CNC

Οι βασικές μεθόδοι με τις οποίες μπορεί να προγραμματιστεί μια εργαλειομηχανή CNC στις μέρες μας είναι 4. Η καθεμιά από αυτές έχει τα πλεονεκτήματα και τα μειονεκτήματα της και δεν θα είχε κανένα νόημα να ειπωθεί ότι κάποιος τρόπος είναι καλύτερος από κάποιον άλλο. Ο καθένας απ' αυτούς έχει το δικό του πεδίο εφαρμογής και ένας καλός προγραμματιστής οφείλει να γνωρίζει άπταιστα όλους τους τρόπους προγραμματισμού ώστε να θεωρείτε αποδοτικός και παραγωγικός. Αυτό όμως προϋποθέτει την γνώση και την δυνατότητα να κρίνει με ποιον τρόπο θα επιλέξει να προγραμματίσει την εργαλειομηχανή του βάσει των αναγκών της συγκεκριμένης εφαρμογής που αντιμετωπίζει.

Εικόνα 1, 5αξονική κατεργασία

5.1.1 Συμβατικός προγραμματισμός με κώδικα ISO (G-Code)

Είναι απ' τις ευκολότερες μεθόδους γι' αυτό μάλιστα έχει και τόσο ευρεία χρήση στην βιομηχανία καθώς θεωρείται βασική γνώση για οποιονδήποτε ασχολείται με τον προγραμματισμό εργαλειομηχανών. Είναι επίσης πολύ εύκολο να φτάσει κανείς σε ένα πολύ ικανοποιητικό επίπεδο γνώσεων πολύ γρήγορα.

Η γλώσσα που συντάσσεται ένα πρόγραμμα είναι τυποποιημένη και η μονάδα ελέγχου διαβάζει και εκτελεί μια μια τις γραμμές του κώδικα με την σειρά όπως έχουν συνταχθεί. Κάθε γραμμή στον κώδικα μπορεί να περιέχει μια η και παραπάνω εντολές (με κάποιους περιορισμούς) αλλά πάντα τηρείται αυστηρά η σειρά εκτέλεσης τους από τον ελεγκτή της εργαλειομηχανής.

Η μέθοδος αυτή έχει αρκετές δυνατότητες και για εύκολες και απλές εφαρμογές είναι τις περισσότερες φορές η πιο αποδοτική. Είναι πολύ σημαντική η γνώση ανάγνωσης και γραφής προγραμμάτων σε κώδικα ISO καθώς αυτό βοηθάει στην πρόληψη, την αποφυγή και την διόρθωση σφαλμάτων ακόμα και κατά την χρήση άλλων μεθόδων προγραμματισμού.

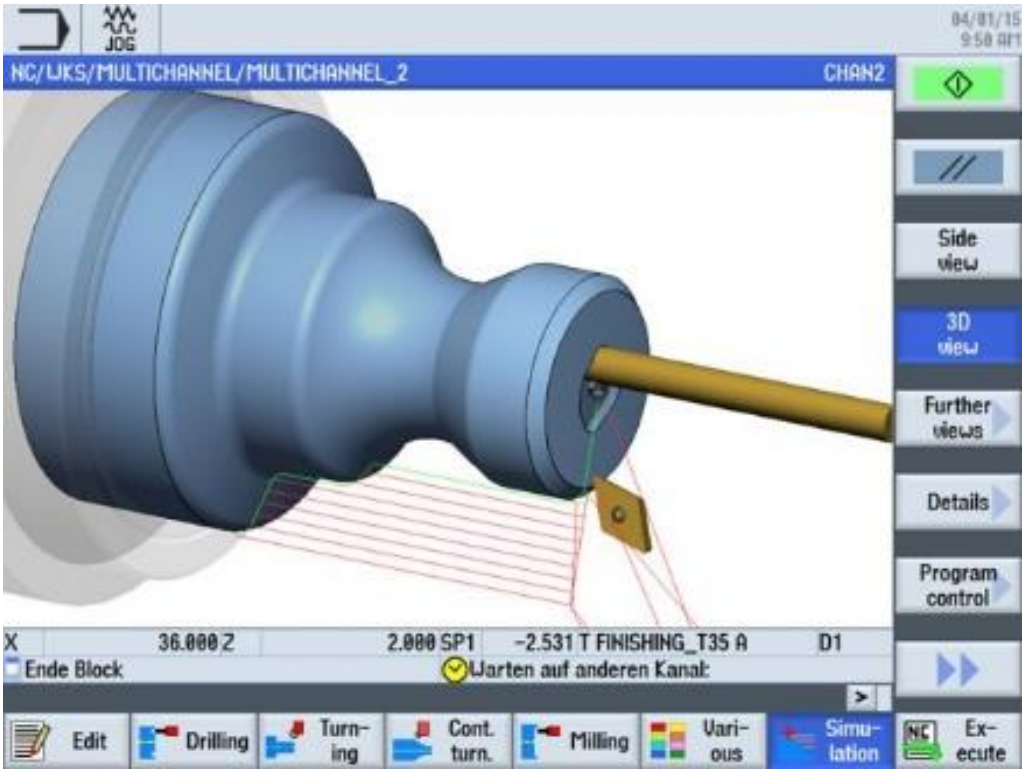
Το βασικό μειονέκτημα της μεθόδου είναι ότι δεν έχει βασικά χαρακτηριστικά που διαθέτουν άλλες γλώσσες προγραμματισμού όπως πχ. την χρήση μεταβλητών, μαθηματικών & λογικών εκφράσεων. loops κλπ. Αυτό την καθιστά πολλές φορές αντιπαραγωγική και όχι και τόσο αποδοτική μέθοδο για συνθέτες και πολύπλοκες εφαρμογές παρόλα αυτά οι δυνατότητες της είναι πολύ μεγάλες.

Εικόνα 2 προγραμματισμός απευθείας στον ελεγκτή της μηχανής

5.1.2 Διαλογικός προγραμματισμός (Conversational Programming)

Η μέθοδος αυτή είναι η πιο εύχρηστη και φιλική ως προς τον χειριστή από όλες καθώς δεν χρειάζεται κανείς να γνωρίζει καμιά γλώσσα προγραμματισμού. Αρκεί μόνο να γνωρίζει τι χρειάζεται να γίνει και με ποια σειρά. Στην ουσία το λογισμικό του ελεγκτή καθοδηγεί με διαδραστικό τρόπο τον προγραμματιστή να ορίσει μια σειρά από δεδομένα (γεωμετρία, συνθήκες κοπής, το είδος και την σειρά των κατεργασιών κλπ.) που απαιτούνται ώστε να ολοκληρωθεί ο προγραμματισμός του τεμαχίου.

Οι περιορισμοί σε αυτή την περίπτωση είναι συνήθως γεωμετρικοί καθώς σε πολύπλοκες γεωμετρίες μπορεί ο προγραμματισμός με αυτή την μέθοδο να μην είναι εφικτός. Όμως σε αρκετές περιπτώσεις είναι ο πιο γρήγορος τρόπος προγραμματισμού. Μάλιστα οι μεγαλύτεροι κατασκευαστές controller εργαλειομηχανών κατασκευάζουν δικά τους λογισμικά ώστε να δώσουν την δυνατότητα του διαλογικού προγραμματισμού στους πελάτες τους (πχ. Siemens Sinumerik (εικόνα 1, 2017), Heidenhein, Hurco Winmax, Mitsubishi, Fanuc κ.α.)

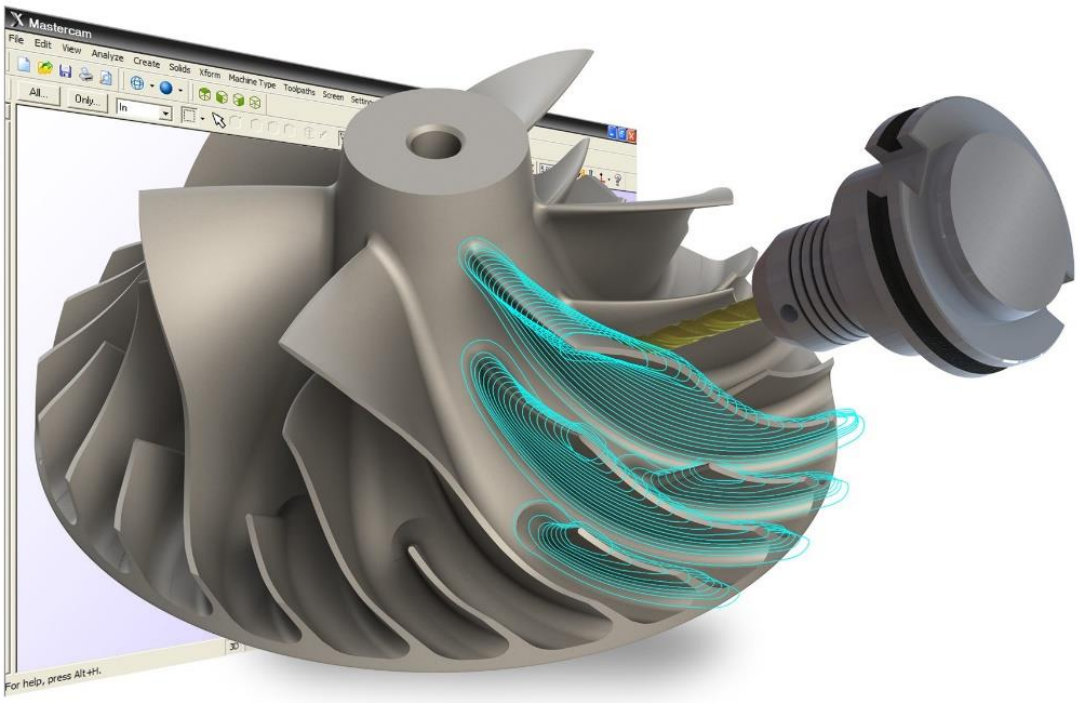


Εικόνα 3, περιβάλλον προσομοίωσης σε ελεγκτή με διαλογικό προγραμματισμό

5.1.3 Προγραμματισμός με την χρήση λογισμικών CAM

Τα λογισμικά CAM την τελευταία δεκαετία έχουν φτάσει σε ένα επίπεδο που υπερκαλύπτουν τις ανάγκες για τους περισσοτέρους χρήστες και οι δυνατότητες τους είναι σχεδόν απεριόριστες σε αυτό που καλούνται να κάνουν όμως ο βασικός ρόλος τέτοιον λογισμικών είναι να προγραμματίζουν ότι δεν γίνεται η δεν συμφέρει να γίνει με συμβατικό προγραμματισμό (G-code). Η διαφορά του προγραμματισμού με αυτή την μέθοδο και του κώδικα G είναι όση και η διαφορά επίλυση αριθμητικών πράξεων με το χέρι και τον υπολογιστή. Σε μια πολύπλοκη περίπτωση σίγουρα ο υπολογιστής θα έκανε πιο γρήγορα την δουλειά, όμως σε πολύ απλές περιπτώσεις μπορεί να γίνει ακόμα και αντιπαραγωγικός.

Το Βασικό πλεονέκτημα όμως της μεθόδου αυτής είναι σε πολύπλοκες γεωμετρίες (5axis) και πολλές φορές την καθιστά μονόδρομο (εικόνα 1, n.d.). Ένα ακόμα πλεονέκτημα που έχουν τα λογισμικά CAM είναι η δυνατότητα προγραμματισμού με πιο σύγχρονες μεθόδους κατεργασίας (helical, trochoidal milling κλπ.) ώστε να επιτευχθεί μεγαλύτερη διάρκεια ζωής στα κοπτικά εργαλεία ,καλύτερες επιφάνειες και διαστατικές ακρίβειες.



Εικόνα 4 Προγραμματισμός σε περιβάλλον CAM

5.1.4 Παραμετρικός Προγραμματισμός

Είναι ίσως η λιγότερο διαδεδομένη μέθοδος λόγο τις δυσκολίας εκμάθησης και χρήσης που έχει. Στην πραγματικότητα όμως η χρήση των προγραμμάτων macro είναι πολύ εύκολη .Η δυσκολία είναι στην ανάπτυξη νέων προγραμμάτων macro ανάλογα με τις ανάγκες της εφαρμογής. Έτσι όμως αξιοποιείται στο έπακρο η δυνατότητα της κάθε μηχανής προγραμματιστικά και τα πλεονεκτήματα αντισταθμίζουν την όποια δυσκολία κληθεί να αντιμετωπίσει ο προγραμματιστής.

Τις περισσότερες φορές οι προγραμματιστές που δεν γνωρίζουν τις macros επιλέγουν να δουλέψουν με λογισμικά CAM θεωρώντας ότι έτσι είναι πιο γρήγοροι και αποτελεσματικοί όμως αυτό δεν ισχύει πάντα. Ο προγραμματισμός με macros έχει σαν αποτέλεσμα να δημιουργούνται προγράμματα με λιγότερες γραμμές κώδικα, πιο εύκολα στην ανάγνωση ακόμα και από κάποιον τρίτο, πιο εύκολα στην διόρθωση και με πολλές επιπλέον δυνατότητες που ένα λογισμικό CAM δεν μπορεί να παρέχει.

Για παράδειγμα με ένα λογισμικό CAM δεν μπορεί να παραχθεί μια εξατομικευμένη υπορουτίνα. Ένα πρόγραμμα macro έχει εντελώς διαφορετική φιλοσοφία από οποιαδήποτε άλλη μέθοδο προγραμματισμού με δυνατότητες που οι περισσότεροι δεν γνωρίζουν και αυτό βοηθάει έναν προγραμματιστή να ξεχωρίσει στην αγορά.

Αναλυτικότερα θα παρατεθούν σε παρακάτω κεφάλαια όλες οι δυνατότητες του παραμετρικού προγραμματισμού μεθοδικά ώστε να γίνει κατανοητό και ευνόητο πως γίνεται η χρήση και η ανάπτυξη νέων προγραμμάτων για κάθε περίπτωση με παραδείγματα.

6. Βασικές έννοιες

Ο προγραμματισμός με macro είναι μια τεχνική προγραμματισμού που συνδυάζει την συμβατική μέθοδο προγραμματισμού με κάποιες επιπλέον λειτουργίες της μονάδας ελέγχου της εργαλειομηχανής ώστε να προκύψουν περισσότερες δυνατότητες και περισσότερη ευελιξία. Η συγκεκριμένη τεχνική είναι παρόμοια με τις βασικές γλώσσες προγραμματισμού όπως η C++ και η Visual Basic όπου χρησιμοποιούνται στην ανάπτυξη λογισμικών

Η γνώση του τι ακριβώς είναι οι macros και τι δυνατότητες παρέχουν στον χρήστη οδηγεί στην καλύτερη χρήση. Υπάρχουν πολλές περιπτώσεις που είναι απαραίτητο να συνυπάρξουν σε ένα πρόγραμμα με άλλες μεθόδους προγραμματισμού (πχ CAM). Αυτό δεν σημαίνει ότι σε μια τέτοια περίπτωση η μια μέθοδος αντικαθιστά την άλλη, άλλα ότι και οι δυο συνεργάζονται σαν προγραμματιστικά εργαλεία ώστε να επιτευχθεί ένας συγκεκριμένος στόχος όπως για παράδειγμα ο έλεγχος εξωτερικού εξοπλισμού που συνεργάζεται με την εργαλειομηχανή, αυτοματισμοί ,κατασκευές custom ρουτινών κλπ.

Σε οποιαδήποτε γλώσσα επικοινωνίας εμπεριέχονται μέσα χαρακτήρες, λέξεις και εκφράσεις που αν συνταχθούν σωστά καταλήγουν σε ένα κείμενο με συγκεκριμένο σκοπό και νόημα. Ακριβώς το ίδιο συμβαίνει και με τα προγράμματα Macro. Μια άμεση σύγκριση με την καθημερινή μας γλώσσα θα βοηθήσει στην καλύτερη κατανόηση βασικών εννοιών που χρησιμοποιούνται σε τέτοια προγράμματα.

- Χαρακτήρες: G, M, N, X, Y, Z, 0, 1, 2, 3, -, (), #, *
- Με τον κατάλληλο συνδυασμό χαρακτήρων προκύπτουν οι ¨λέξεις¨: G01, M08, #512, X-350.
- Με τον κατάλληλο συνδυασμό λέξεων προκύπτουν λογικές εκφράσεις (blocks): G00 X40. Y50. , #1=#2/#3
- Τέλος, με τον κατάλληλο συνδυασμό λογικών εκφράσεων δομείται ένα ολοκληρωμένο πρόγραμμα.

7. Πεδίο εφαρμογής

Σίγουρα οι δυνατότητες είναι σχεδόν απεριόριστες και αυτό από μόνο του αρκεί για να διαπιστωθεί ότι η χρήση των macro μπορεί να διευκολύνει τον προγραμματιστή σε πολλούς τομείς όπως για παράδειγμα σε:

- Οικογένειες κομματιών
- Κατασκευή custom εντολών G & M
- Κατασκευή custom υπορουτινών
- Καθοδήγηση εξωτερικών συσκευών
- Probing

- Δυναμικός έλεγχος και διαχείριση των Offset
- Δυναμικός έλεγχος και διαχείριση παραμέτρων της μονάδας ελέγχου
- Alarms & warnings
- Πολύπλοκα και σύνθετα toolpaths

8. Δομή παραμετρικών προγραμμάτων

Η δομή των παραμετρικών προγραμμάτων δεν διαφέρει και πολύ από αυτή των μη παραμετρικών αφού οι βασικές τεχνικές προγραμματισμού εφαρμόζονται και στις δυο περιπτώσεις. Η βασική διαφορά βρίσκεται κυρίως στην ευελιξία και στην δυναμικότητα της εκτέλεσης των παραμετρικών προγραμμάτων. Αυτά τα χαρακτηριστικά οφείλονται στις βασικές προγραμματιστικές δυνατότητες που είναι διαθέσιμες στον παραμετρικό προγραμματισμό οι οποίες είναι η χρήση μεταβλητών και σταθερών του συστήματος η ακόμα και η δημιουργία νέων. Επίσης υπάρχει η δυνατότητα να πραγματοποιηθούν μαθηματικές πράξεις μεταξύ των σταθερών και των μεταβλητών καθώς και η εισαγωγή λογικών πράξεων και η ανακατεύθυνση της ροής του προγράμματος.

Στον συμβατικό προγραμματισμό δίδεται πρόσβαση σε κάποιες σταθερές και μεταβλητές του συστήματος αλλά με έμμεσο τρόπο. Για παράδειγμα υπάρχει η δυνατότητα να κληθεί μια τιμή για ένα συγκεκριμένο offset (tool ή work) μέσω της σειράς εντολών G43 ή να αλλάξει η τιμή αυτή μέσω της εντολής G10 αλλά δεν είναι άμεσα διαχειρίσιμες οι τιμές του πίνακα. Εν αντιθέσει στα παραμετρικά προγράμματα ο προγραμματιστής μπορεί να διαχειριστεί ένα μεγάλο πλήθος σταθερών και μεταβλητών του συστήματος η δικές του και να τις αξιοποιήσει τόσο σε υπολογισμούς όσο και στην λήψη αποφάσεων και την ανακατεύθυνση της ροής του προγράμματος.

Αυτές οι προγραμματιστικές δυνατότητες είναι η διαφορά μεταξύ παραμετρικών και συμβατικών προγραμμάτων. Κατά τα άλλα η δομή ενός προγράμματος ακολουθεί την ιδιά λογική τόσο για τα προγράμματα όσο και για τα υποπρογράμματα. Δηλαδή, ένα παραμετρικό πρόγραμμα εξακολουθεί να είναι η επαλληλία εντολών αρχής, αλλαγής εργαλείων, κατεργασίας και τέλους. Ενώ ένα υποπρόγραμμα macro είναι ένα κομμάτι προγράμματος που εκτελείται εμβόλιμα στην εκτέλεση ενός άλλου κύριου προγράμματος

9. Μεταβλητές

Η εισαγωγή μεταβλητών τιμών σε ένα πρόγραμμα ξεκλειδώνει αυτόματα την ευελιξία του προγράμματος και δίνει τη δυνατότητα στον προγραμματιστή να δημιουργήσει προγράμματα στα οποία οι περισσότερες αριθμητικές τιμές των λέξεων μπορούν να παραμετροποιηθούν. Αυτό σημαίνει ότι εάν χρησιμοποιηθούν μεταβλητές για να οριστεί η θέση μιας σειράς σημείων σε ένα πρόγραμμα, η αλλαγή των τιμών των μεταβλητών θα σημαίνει αυτόματα την αλλαγή στην τελική γεωμετρία του παραγόμενου τεμαχίου.

Οι μεταβλητές λοιπόν είναι μια δομή δεδομένων που μπορούν να χρησιμοποιηθούν για να αποθηκεύσουν ή να τροποποιήσουν αριθμητικές η λογικές τιμές. Η σύνταξη τους γίνεται με το σύμβολο της δίεσης (#) και έναν αριθμό που παίζει το ρόλο του ονόματος της μεταβλητής.

Για παράδειγμα: Η #567 είναι η μεταβλητή με το όνομα 567.

9.1 Τύποι μεταβλητών

Οι ελεγκτές τον εργαλειομηχανών έχουν κάποιες προκαθορισμένες ομάδες μεταβλητών που έχουν συγκεκριμένες ιδιότητες. Αυτό σημαίνει ότι ο προγραμματιστής δεν είναι ελεύθερος να χρησιμοποιήσει όποιο όνομα μεταβλητής θέλει. Οι ιδιότητες που έχουν οι εκάστοτε μεταβλητές θα πρέπει να είναι πλήρως κατανοητές πριν την ανάπτυξη ενός παραμετρικού προγράμματος ώστε να αποφεύγονται λάθη που έχουν να κάνουν με την άστοχη επιλογή μεταβλητών.

9.1.1 Κενή μεταβλητή (Null)

Η μεταβλητή #0 αποτελεί μια ομάδα από μόνη της αφού έχει την ιδιαιτερότητα να μην μπορεί να της καταχωρηθεί καμία τιμή. Αυτό πρακτικά σημαίνει ότι ο ελεγκτής της μηχανής δεν κατοχυρώνει θέση μνήμης για αυτή την μεταβλητή. Αυτές οι μεταβλητές ονομάζονται κενές η Null.

Αναλυτικότερα, αν υπάρχει μια μεταβλητή με μια συγκεκριμένη τιμή και το ζητούμενο είναι να μείνει κενή (να αφαιρέσουμε δηλαδή την υπάρχουσα τιμή από την μεταβλητή) τότε εξισώνεται με την μεταβλητή Null. Για παράδειγμα:

Έστω ότι #75=12.45

Αυτό σημαίνει ότι η μεταβλητή 75 έχει στην μνήμη της την τιμή 12,45. Αρά οπού κληθεί αυτή η μεταβλητή είναι σαν να αντικαθίσταται ο αριθμός 12.45.

Εάν το ζητούμενο είναι να σταματήσει να υπάρχει τιμή στην μνήμη της μεταβλητής πρέπει να μπει κενό.

#75= #0

Αυτό μπορεί να χρησιμοποιηθεί στην περίπτωση που ο χειριστής του προγράμματος επιθυμεί να ορίζει μόνος του κάθε φορά όποια τιμή θέλει η και σε άλλες πιο σύνθετες περιπτώσεις ακολουθούν αναλυτικότερα παρακάτω.

9.1.2 Τοπικές μεταβλητές (Local variables)

Οι τοπικές μεταβλητές έχουν συνήθως το εύρος #1 - #33 και μπορούν να αποθηκεύσουν οποιαδήποτε τιμή τους οριστεί η οποία θα αντιστοιχηθεί σε μια θέση της μνήμης του ελεγκτή. Η ιδιαιτερότητα που έχουν οι τοπικές μεταβλητές είναι ότι μπορούν να χρησιμοποιηθούν σε προγράμματα και να πάρουν τις τιμές που θα τους οριστούν, ενώ μετά το πέρας της εκτέλεσης του προγράμματος θα χαθούν και δεν θα μεταφερθούν σε άλλο πρόγραμμα. Έχουν δηλαδή ισχύ μόνο στο επίπεδο του προγράμματος που εκτελείται κάθε φορά. Έτσι αν έχει οριστεί μια σειρά μεταβλητών σε ένα κυρίως πρόγραμμα (πχ #1, #2, #4), οι μεταβλητές αυτές επηρεάζουν μόνο το κύριο πρόγραμμα. Αν το κύριο πρόγραμμα παρακάτω καλέσει ένα παραμετρικό υποπρόγραμμα που εμπεριέχει τοπικές μεταβλητές #1 #2 #4 τότε αυτές θα αντιμετωπιστούν ως διαφορετικές χωρίς να υπάρχει σύγχυση μεταξύ αυτών και του κυρίου προγράμματος. Αντίστοιχα το ίδιο θα ισχύσει αν κληθεί ένα δεύτερο παραμετρικό υποπρόγραμμα εντός του τρέχοντος και ούτω καθεξής μέχρι να συμπληρωθεί ο μέγιστος αριθμός εμφώλευσης (nesting) παραμετρικών προγραμμάτων ο οποίος είναι ίδιος με τον αριθμό εμφώλευσης συμβατικών προγραμμάτων (συνήθως 4).

Αυτό πρακτικά σημαίνει ότι κάθε φορά που καταχωρούνται τιμές σε τοπικές μεταβλητές αυτές αφορούν μόνο το τρέχον πρόγραμμα η υποπρόγραμμα και παραμένουν ενεργές όσο αυτό εκτελείται. Δηλαδή η #1 του κυρίου προγράμματος έχει διαφορετική θέση στη μνήμη του ελεγκτή από την #1 του κάθε υποπρογράμματος. Γίνεται λοιπόν αντιληπτό ότι κάθε τοπική μεταβλητή μπορεί να χρησιμοποιηθεί μέχρι και 5 φορές αν θεωρηθεί ότι ο βαθμός εμφώλευσης είναι 4.

Κάθε φορά που εκτελείται μια εντολή M99 η M30 οι τοπικές μεταβλητές του τρέχοντος προγράμματος η υποπρογράμματος γίνονται αυτομάτως Null.

9.1.3 Κοινές μεταβλητές (Common Variables)

Οι κοινές μεταβλητές χρησιμοποιούνται για να καταχωρηθούν τιμές που πρέπει να αξιοποιηθούν τόσο στο κύριο πρόγραμμα όσο και στα υποπρογράμματα που θα κληθούν κατά τη διάρκεια της εκτέλεσης. Οι κοινές μεταβλητές χωρίζονται σε δυο ομάδες όπου η πρώτη ομάδα συνήθως έχει το εύρος #100-#149 και αντιστοιχεί σε θέσεις προσωρινής μνήμης (RAM) ενώ η δεύτερη ομάδα έχει συνήθως το εύρος #500-#531 και αντιστοιχεί σε θέσεις μόνιμης μνήμης. Αυτό σημαίνει ότι οι τιμές της πρώτης ομάδας γίνονται κενές όταν απενεργοποιηθεί ο ελεγκτής ενώ οι τιμές της δεύτερη ομάδας παραμένουν αποθηκευμένες ακόμα και μετά από μια επανεκκίνηση της εργαλειομηχανής.

Δηλαδή, η πρώτη ομάδα χρησιμοποιείται για την αποθήκευση τιμών για το τρέχον πρόγραμμα ώστε να είναι δυνατή η χρήση των μεταβλητών αυτών τόσο στο κυρίως όσο και στα περιεχόμενα macros. Ενώ η δεύτερη ομάδα μεταβλητών μπορεί να αξιοποιηθεί για την αποθήκευση τιμών που μπορεί να χρησιμοποιηθεί επανειλημμένα από διαφορετικά προγράμματα και υποπρογράμματα και συνεπώς έχει νόημα να κρατηθούν αποθηκευμένες στον ελεγκτή της εργαλειομηχανής.

Για να επιλεχθεί ο κατάλληλος τύπος κοινής μεταβλητής για μια δεδομένη περίπτωση πρέπει να ξεκαθαριστεί εξ αρχής αν οι τιμές των μεταβλητών που θα οριστούν εντός ενός προγράμματος ή υποπρογράμματος πρόκειται να αξιοποιηθούν για τους σκοπούς μόνο αυτού προγράμματος, ή εάν οι τιμές που θα τους καταχωρηθούν θα είναι διαθέσιμες κάθε στιγμή στον ελεγκτή της εργαλειομηχανής. Άρα για μεταβλητές που οι τιμές τους απαιτείται να ορίζονται κάθε φορά από το τρέχον πρόγραμμα ή υποπρόγραμμα και είναι επιθυμητό να αξιοποιούνται όσο εργαζόμαστε

στην εργαλειομηχανή επιλέγουμε την χρήση κοινών μεταβλητών που συνδέονται με την προσωρινή μνήμη του ελεγκτή ενώ απ' την άλλη όταν απαιτείται να οριστεί μια μεταβλητή που θα θεωρείται στο εξής παράμετρος του συστήματος που με δυνατότητα τροποποίησης κατά την εκτέλεση των τρεχόντων προγραμμάτων και υποπρογραμμάτων τότε επιλέγεται ο ορισμός μεταβλητής από το εύρος των κοινών μεταβλητών που αποθηκεύονται στην μόνιμη μνήμη του ελεγκτή.

Ιδιαίτερη προσοχή πρέπει να δίνεται κατά την επιλογή και τον ορισμό κοινών μεταβλητών που αποθηκεύονται στην μόνιμη μνήμη του ελεγκτή επειδή η τιμή τους παραμένει μόνιμα και ο μόνος τρόπος να γίνουν ξανά κενές είναι να τους καταχωρηθεί η τιμή #0 .Για παράδειγμα: #507 = #0

9.1.4 Μεταβλητές συστήματος (System Variables)

Οι μεταβλητές συστήματος χρησιμοποιούνται από τον ελεγκτή για να καταχωρούνται διάφορα στοιχεία της κατάστασης της εργαλειομηχανής όπως για παράδειγμα η τρέχουσα θέση των αξόνων ή τιμές των αντισταθμίσεων κ.α.

Η ιδιαιτερότητα που έχουν οι μεταβλητές συστήματος είναι ότι ο προγραμματιστής δεν έχει πάντα πλήρη δικαιώματα στην διαχείριση τους. Αυτό σημαίνει ότι κατά περίπτωση μπορεί είτε μόνο να προσπελάσει την τιμή τους (Read only) είτε και να την τροποποιήσει. Για παράδειγμα οι μεταβλητές συστήματος που διαχειρίζονται τις αντισταθμίσεις των εργαλείων (tool offset) δίνουν το δικαίωμα στον προγραμματιστή και για ανάγνωση και για εγγραφή που σημαίνει ότι μπορεί να ανακτήσει μια τέτοια μεταβλητή και να την αλλάξει κατά την διάρκεια εκτέλεσης του προγράμματος. Από την άλλη πλευρά οι μεταβλητές που δείχνουν την τρέχουσα θέση των αξόνων είναι μόνο για ανάγνωση. Το δικαίωμα ανάγνωσης και εγγραφής της κάθε μεταβλητής συστήματος που χρησιμοποιείται πρέπει να ελέγχεται διότι προσπάθεια εγγραφής σε μεταβλητή που δεν παρέχει αυτό το δικαίωμα θα ενεργοποιήσει συντακτικό σφάλμα κατά την εκτέλεση. Ένας τρόπος να ελεγχθεί η δυνατότητα εγγραφής σε μια μεταβλητή συστήματος πριν την αξιοποιηθεί σε κάποιο πρόγραμμα είναι να δοκιμαστεί σε περιβάλλον MDI για να παρατηρηθεί άμεσα αν θα ενεργοποιηθεί κάποιο σφάλμα ή αν θα καταχωρηθεί κανονικά η τιμή στην μεταβλητή. Παρόλα αυτά πρέπει να αναφερθεί ότι στην περίπτωση μεταβλητών του συστήματος που δεν υπάρχει δικαίωμα εγγραφής, είναι δυνατό να παρθεί η τιμή και να καταχωρηθεί σε κάποια τοπική η κοινή

μεταβλητή και στην συνέχεια να γίνει επεξεργασία όπως απαιτείται στην εκάστοτε περίπτωση. Για παράδειγμα:

#1 = #5021 (εκχώρηση της τιμής της τρέχουσας θέσης του άξονα X στην τοπική μεταβλητή #1)

Το εύρος αυτής της ομάδας μεταβλητών είναι συνήθως από την μεταβλητή #1000 και πάνω και για να λάβει γνώση ο προγραμματιστής για τον ρόλο και τα δικαιώματα της κάθε μεταβλητής πρέπει να ανατρέξει στο αντίστοιχο εγχειρίδιο του ελεγκτή της συγκεκριμένης μηχανής (Parameters manual). Οι μεταβλητές αυτές είναι ορισμένες από το ίδιο το σύστημα και οι τιμές τους καταχωρούνται αυτομάτως λόγω της κατάστασης του συστήματος η ως αποτέλεσμα του χειρισμού της μηχανής. Ο ορισμός των μεταβλητών του συστήματος εξαρτάται άμεσα από τον εκάστοτε ελεγκτή και μπορεί να υπάρχουν διαφορές και μεταξύ διαφορετικών μοντέλων ελεγκτών ακόμα και από τον ίδιο κατασκευαστή.

Στον παρακάτω πίνακα (Smid, 2005) παρουσιάζονται οι συνηθέστερες και πιο χρήσιμες μεταβλητές συστήματος από ελεγκτή Fanuc.

#2000	Αντισταθμίσεις εργαλείων
#2001	T1
#2002	T2
.	.
.	.
.	.
#2200	T200
#5021-#5026	Τρέχον θέση αξόνων
#5041-#5046	Τρέχον θέση αξόνων βάση work offset

#5221-#5226	G54 (X,Y,Z,A,B,C)
#5241-#5246	G55 (X,Y,Z,A,B,C)
#5261-#5266	G56 (X,Y,Z,A,B,C)
#5281-#5286	G57 (X,Y,Z,A,B,C)
#5301-#5306	G58 (X,Y,Z,A,B,C)
#5321-#5326	G59 (X,Y,Z,A,B,C)
#4120	T code
#4119	S code
#3901	Counter 1
#3902	Counter 2
#3003	Single block
#3004	Feed Hold

9.2 Οι μεταβλητές στον προγραμματισμό

Οι μεταβλητές είναι πάντα ισοδύναμες με την τιμή που τους έχει καταχωρηθεί. Έτσι αν υποθέσουμε ότι η μεταβλητή #123 έχει την τιμή 1, τότε αυτή μπορεί να χρησιμοποιηθεί τόσο για την πραγματοποίηση αριθμητικών πράξεων όσο και για να αντικατασταθεί το αριθμητικό τμήμα εντολών που χρησιμοποιούνται στον προγραμματισμό. Δηλαδή η λέξη G#123 είναι ισοδύναμη με την εντολή G01.

Οι μόνες περιπτώσεις που το αριθμητικό τμήμα δεν μπορεί να αντικατασταθεί από μεταβλητές είναι τα ονόματα προγράμματος (O_____) και η βοηθητικές εντολές N για την αρίθμηση των γραμμών (πχ N105).

Η καταχώρηση των τιμών σε μια μεταβλητή γίνεται με την σύνταξη #__ = __

οπότε για το παραπάνω παράδειγμα μεταβλητής θα ισχύει : #123=1

Αξίζει να σημειωθεί ότι όσον αφορά την καταχώρηση των αριθμητικών τιμών στις μεταβλητές, πρέπει να λαμβάνεται υπόψιν αν η μεταβλητή κατά τη χρήση της θα χρειαστεί να έχει ακέραια η πραγματική τιμή. Για παράδειγμα αν η τιμή της μεταβλητής #123 ήταν 1.0 τότε το G#123 θα έβγαζε alarm. Αντίστοιχα αν η τιμή της ήταν 1 στην εντολή G01 X#123 F1000. μπορεί να είχε σαν αποτέλεσμα την εκτέλεση λογικού σφάλματος επειδή πολλοί ελεγκτές απαιτούν οι τιμές των εντολών να δίνονται με πραγματικές τιμές. Συνεπώς οι ελεγκτές θα βγάλουν alarm σε ένα συντακτικό λάθος άλλα θα εκτελέσουν κανονικά το λογικό σφάλμα γι’ αυτό, όπως στον συμβατικό έτσι και στον παραμετρικό προγραμματισμό, τα λογικά σφάλματα είναι τα πιο επικίνδυνα και τα πιο δύσκολα να εντοπιστούν.

10. Πράξεις

Όπως προαναφέρθηκε οι μεταβλητές επιτρέπουν ουσιαστικά να αποθηκευτεί μια αριθμητική η λογική τιμή σε μια θέση μνήμης του ελεγκτή όπου στην συνέχεια μπορεί να ανακτηθεί καλώντας την εν λόγω μεταβλητή. Η λειτουργία αυτή δεν θα είχε κανένα νόημα αν δεν ήταν δυνατό να αξιοποιηθούν οι τιμές αυτές για να εκτελεστούν αριθμητικές και λογικές πράξεις που άλλωστε αυτή είναι και η πραγματική αξία του παραμετρικού προγραμματισμού. Έτσι οι μεταβλητές μπορούν να συμμετέχουν σε

λογικές η αριθμητικές πράξεις που τα αποτελέσματα τους θα αποτελούν τιμές άλλες μεταβλητές.

Τα δεδομένα που μπορούν να αποθηκευτούν σε μια μεταβλητή μπορεί να είναι είτε αριθμητικά (δεκαδικές η δυαδικές τιμές) είτε λογικά (True / False, δηλ. 1 ή 0 αντίστοιχα). Η σύνταξη της καταχώρησης μιας τιμής σε μια μεταβλητή αναφέρθηκε προηγουμένως, αλλά αυτό που δεν επισημάνθηκε είναι ότι με τον ίδιο τρόπο μπορεί να εκχωρηθεί η τιμή μιας υπάρχουσας μεταβλητής σε μια άλλη. Δηλαδή για παράδειγμα: #1 = #3 .

Αυτό δεν υπονοεί ισότητα αλλά ότι η τιμή από την μεταβλητή #3 ανακτήθηκε και εκχωρήθηκε στην μεταβλητή #1. Όμως αυτές οι δυο μεταβλητές συνεχίζουν να είναι ανεξάρτητες μεταξύ τους και η κάθε μια καταλαμβάνει τη δική της θέση στην μνήμη του ελεγκτή.

Στην παρακάτω αλληλουχία εντολών παρουσιάζεται μια εκχώρηση τιμής από μια μεταβλητή σε μια άλλη και το τί συμβαίνει στην συνέχεια της εκτέλεσης με τις τιμές των μεταβλητών.

κώδικας	#1	#3
#1=1	1	#0
#3=2	1	2
#1 = #3	2	2
#3=1	2	1

Στον παρακάτω πίνακα παρουσιάζονται οι διαθέσιμες πράξεις μεταξύ μεταβλητών καθώς και οι συναρτήσεις που υποστηρίζονται από τον ελεγκτή. Οι συναρτήσεις αξιοποιούνται για την πραγματοποίηση υπολογισμών που σε άλλη περίπτωση αντίστοιχες πράξεις θα ήταν αρκετά πολύπλοκες και χρονοβόρες.

Η προτεραιότητα των πράξεων τηρείτε ακριβώς όπως και στον μαθηματικό λογισμό με την μόνη διαφορά ότι στον παραμετρικό προγραμματισμό αντί των παρενθέσεων χρησιμοποιούνται τετράγωνες αγκύλες επειδή οι παρενθέσεις χρησιμοποιούνται για την εισαγωγή σχολίων στο πρόγραμμα. Έτσι βάση

προτεραιότητας πρώτα εκτελούνται οι συναρτήσεις, μετά οι πολλαπλασιασμοί και οι διαιρέσεις και τέλος οι προσθέσεις και οι αφαιρέσεις εκτός αν υπάρχουν πράξεις εντός τετράγωνων αγκυλών οπότε η προτεραιότητα επαναπροσδιορίζεται με την εκτέλεση των πράξεων εντός των αγκυλών. Για παράδειγμα , αν το ζητούμενο είναι να υπολογιστεί η ταχύτητα περιστροφής της ατράκτου και να καταχωρηθεί σε μια μεταβλητή θα είχαμε τα εξής:

$N= \frac{Vc * 1000}{\pi * D}$,

όπου : N= ταχύτητα περιστροφής της ατράκτου (rpm)

Vc= ταχύτητα κοπής (m/min)

π= 3,14

D= διάμετρος κοπτικού εργαλείου (mm)

Άρα,

#1= (Vc)

#2= (D)

#3= (N)

#1=150

#2=6

#3=#1*1000/[3.14 * #2]

Στις λογικές πράξεις η προτεραιότητα της εκτέλεσης των AND είναι μεγαλύτερη από αυτή των OR / XOR ενώ και πάλι οι τετράγωνες αγκύλες μπορούν να αλλάξουν τον υπολογισμό με την εκτέλεση πρώτα των πράξεων που βρίσκονται μέσα στις αγκύλες.

11. Προγράμματα & υποπρογράμματα Macro

11.1 Συμβατικά & Macro υποπρογράμματα

Τα υποπρογράμματα είναι κομμάτια προγραμμάτων που βρίσκονται αποθηκευμένα στην μνήμη του ελεγκτή της μηχανής και όπως κάθε άλλο πρόγραμμα έχει ένα δικό του αριθμό (όνομα). Δεν είναι ολοκληρωμένα προγράμματα και είναι σχεδιασμένα ώστε να εκτελέσουν συγκεκριμένες μόνο εντολές εφόσον κληθούν από κάποιο άλλο πρόγραμμα.

Στον συμβατικό προγραμματισμό συνηθίζεται η χρήση υποπρογραμμάτων σε περιπτώσεις που υπάρχει μια επαναλαμβανόμενη κατεργασία (πχ ένα περιφερικό φρεζάρισμα σε πολλά πάσα, μια σειρά οπών κλπ.) . Για να κατανοηθούν οι διαφορές ενός συμβατικού υποπρογράμματος από ένα Macro οφείλουμε να γνωρίζουμε αρκετά καλά πρώτα πως αυτό γίνεται με τον συμβατικό προγραμματισμό.

Υπάρχουν δυο τρόποι να κληθεί ένα υποπρόγραμμα στον συμβατικό προγραμματισμό. Είτε δημιουργείται ένα κύριο πρόγραμμα και ένα υποπρόγραμμα ξεχωριστά που θα καλείται να εκτελεστεί από το κύριο σε συγκεκριμένες θέσεις, είτε να μπει το υποπρόγραμμα μέσα στο κύριο (σε συγκεκριμένη θέση) και να καλείται κάθε φορά που το χρειάζεται. Και οι δυο μέθοδοι είναι αποδέκτες και κάθε φορά επιλέγεται οποία βολεύει περισσότερο ανάλογα την εφαρμογή.

Μέθοδος 1

%

O12345 (Main program)	Όνομα κύριου προγράμματος
G54 G90 G17 G80 G40	Γραμμή ασφαλείας
T1 M06	Επιλογή εργαλείου #1
G00 X0. Y0.	Θέση εκτέλεσης υποπρογράμματος
G00 G43 H1 Z20. M08	Αντιστάθμιση ύψους για το εργαλείο #1
M98 P54321 (subprogram)	κλήση εξωτερ. υποπρογράμματος
G55 G90 X0. Y0.	

M98 P54321

G56 G90 X0. Y0.

M98 P54321

GZ100. M09	Θέση ασφαλείας
G53 Y0.	Θέση παρκαρίσματος
M30	Τέλος προγράμματος

%

Μέθοδος 2

%

O12345 (Main program)	Όνομα κυρίου προγράμματος
G54 G90 G17 G80 G40	Γραμμή ασφαλείας
T1 M06	Επιλογή εργαλείου #1
G00 X0. Y0.	Θέση εκτέλεσης υποπρογράμματος
G00 G43 H1 Z20. M08	Αντιστάθμιση ύψους
M97 P54321 (subprogram)	κλήση εσωτερικού υποπρογράμματος

G55 G90 X0. Y0.

M97 P54321

G56 G90 X0. Y0.

M97 P54321

GZ100. M09	Θέση ασφαλείας
G53 Y0.	Θέση παρκαρίσματος
M30	Τέλος προγράμματος

;

N54321 Εσωτερικό υποπρόγραμμα

G83 G99 Z-10. R2. Q1. F100.

G91 Y7. L8

.

.

.

G00 Z20. Θέση ασφαλείας

M99 Επιστροφή στο κυρίως πρόγραμμα

%

Στις παραπάνω περιπτώσεις το υποπρόγραμμα έκανε μια σειρά από δέκα οπές αυτό σημαίνει ότι όσες φορές κληθεί μέσα στο κύριο πρόγραμμα (3) η εργαλειομηχανή θα πάει να κάνει τις δέκα αυτές οπές στην θέση που έχει προγραμματιστεί. Όταν όμως το ζητούμενο είναι κάθε φορά που καλείται να του ορίζεται για παράδειγμα και πόσες οπές θα κάνει, αυτό δεν είναι εφικτό με συμβατικό προγραμματισμό. Πρέπει λοιπόν να δημιουργηθεί ένα υποπρόγραμμα macro που κάθε φορά που θα καλείται να του ορίζεται και πόσες οπές πρέπει να κάνει.

12. Κλήση Macros

Τα παραμετρικά υποπρογράμματα ενσωματώνουν μεταβλητές που τις κάνει να έχουν δυναμικότητα κατά την εκτέλεση τους. Οι μεταβλητές αυτές συνήθως είναι τοπικές και πρέπει να αρχικοποιούνται κατά την κλίση του macro. Για αυτό τον λόγο τα παραμετρικά υποπρογράμματα δεν καλούνται με τις εντολές M98 και M97 διότι έτσι δεν θα είχαμε την δυνατότητα να αρχικοποιηθούν οι τιμές των τοπικών μεταβλητών που χρησιμοποιούνται.

Η κλήση ενός παραμετρικού υποπρογράμματος γίνεται πάλι με δυο τρόπους.

Είναι δυνατό να κληθεί ένα macro υποπρόγραμμα είτε με G65 είτε με G66. Οι εντολές G65 και G66 συντάσσονται παρόμοια με τις M98 και M97 με την διαφορά ότι

εδώ υπάρχει η δυνατότητα να αρχικοποιηθούν οι τιμές των τοπικών μεταβλητών (arguments) που χρησιμοποιούνται

Η βασική σύνταξη της εντολής κλήσης για τις macros είναι της μορφής:

G65 P_____ L__ <Arguments>

Οπού, P =όνομα του macro, L = αριθμός επαναλήψεων της κλήσης, Arguments = οι τοπικές μεταβλητές που πρέπει να αρχικοποιηθούν.

Αναλυτικότερα:

G65 P54321 H5 (P= Όνομα υποπρογράμματος, H= είσοδος τιμής 5 για τοπική μεταβλητή #11)

ή

G66 P54321 H5 (P= Όνομα υποπρογράμματος, H= είσοδος τιμής 5 για τοπική μεταβλητή #11)

Και στις δυο περιπτώσεις θα κληθεί το υποπρόγραμμα macro με αριθμό 54321 και θα κάνει 5 οπές. Η διαφορά τους είναι ότι με την εντολή G66 το υποπρόγραμμα γίνεται modal και αν θέλουμε να κληθεί ξανά και σε άλλες θέσεις απλά ορίζονται οι συντεταγμένες που θέλουμε να επανεκτελεστεί το υποπρόγραμμα χωρίς να απαιτείται η κλήση του. Στο τέλος ακυρώνουμε την κατάσταση modal με την εντολή G67 .

Παράδειγμα:

O12345 (Main program)	Όνομα κυρίου προγράμματος
G54 G90 G17 G80 G40	Γραμμή ασφαλείας
T1 M06	Επιλογή εργαλείου #1
G00 X0. Y0.	Θέση εκτέλεσης υποπρογράμματος
G00 G43 H1 Z20. M08	αντιστάθμιση ύψους
G65 P54321 H5	κλήση macro O54321 για 5 οπές
G00 X100.	Νέα θέση
G65 P54321 H7	κλήση macro O54321 για 7 οπές
G00 X200.	Νέα θέση

G65 P54321 H9	κλήση macro O54321 για 9 οπές
G00 X300.	Νέα θέση
G65 P54321 H11	Κλήση macro O54321 για 11 οπές
G00 Z100. M09	Θέση ασφαλείας
G53 Y0.	Παρκάρισμα
M30	Τέλος προγράμματος

O12345 (Main program)	Όνομα κυρίου προγράμματος
G54 G90 G17 G80 G40	Γραμμή ασφαλείας
T1 M06	Επιλογή εργαλείου #1
G00 X0. Y0.	Θέση εκτέλεσης υποπρογράμματος
G00 G43 H1 Z20. M08	αντιστάθμιση ύψους
G66 P54321 H5	κλήση macro O54321 για 5 οπές
G00 X100. H7.	Νέα θέση για 7 οπές
G00 X200. H9.	Νέα θέση για 9 οπές
G00 X300. H11.	Νέα θέση για 11 οπές
G67	ακύρωση modal κατάστασης
G00 Z100. M09	Θέση ασφαλείας
G53 Y0.	Παρκάρισμα
M30	Τέλος προγράμματος

Αυτό είναι πολύ χρήσιμο σε περιπτώσεις που πρέπει να κληθεί το υποπρόγραμμα πάρα πολλές φορές. Όσες περισσότερες φορές απαιτείται η κλήση του υποπρογράμματος τόσες περισσότερες γραμμές κώδικα γλιτώνουμε και αυτό είναι

κάτι που κάνει όλο το πρόγραμμα πιο ευανάγνωστο πολύ πιο εύκολο στη διόρθωση και με μικρότερες πιθανότητες αναγραμματισμού και άλλων πιθανών σφαλμάτων.

Υπάρχει η δυνατότητα επίσης μέσα σε ένα υποπρόγραμμα να γίνει κλήση ενός δεύτερου υποπρογράμματος (nesting) και αυτό να φτάσει μέχρι 4 επίπεδα, δίνοντας έτσι στον προγραμματιστή την ευελιξία να δημιουργήσει πολύ σύνθετα και πολύπλοκα προγράμματα.

13. Arguments

Οι παράμετροι Arguments είναι «λέξεις» που η κάθε μια αντιστοιχεί σε μια συγκεκριμένη τοπική μεταβλητή. Οι παράμετροι μπορεί να ληφθούν από δυο διαφορετικούς πίνακες (Sinha, 2005) παραμέτρων που αναλόγως ποιος πίνακας χρησιμοποιείται κάθε φορά αλλάζει και η σύνταξη της κλήσης των macro.

Πίνακας 1

Arguments	**Τοπικές μεταβλητές**
A	#1
B	#2
C	#3
D	#7
E	#8
F	#9
H	#11
I	#4
J	#5
K	#6
M	#13

Q	#17
R	#18
S	#19
T	#20
U	#21
V	#22
W	#23
X	#24
Y	#25
Z	#26

Πίνακας 2

Arguments	**Τοπικές μεταβλητές**
A	#1
B	#2
C	#3
I	#4
J	#5
K	#6
I2	#7
J2	#8
K2	#9

I3	#10
J3	#11
K3	#12
I4	#13
J4	#14
K4	#15
I5	#16
J5	#17
K5	#18
I6	#19
J6	#20
K6	#21
I7	#22
J7	#23
K7	#24
I8	#25
J8	#26
K8	#27
I9	#28
J9	#29
K9	#30
I10	#31
J10	#32
K10	#33

Ο πρώτος πίνακας παραμέτρων μπορεί να χρησιμοποιηθεί για την αρχικοποίηση μέχρι και 21 τοπικών μεταβλητών που θα αξιοποιηθούν κατά την εκτέλεση του macro, ενώ ο δεύτερος πίνακας δίνει την δυνατότητα να αρχικοποιηθούν μέχρι και 33 τοπικές μεταβλητές. Συνήθως προτιμάται ο πρώτος πίνακας λόγο της πι κοιλίας των γραμμάτων που συντάσσονται καθώς αυτό κάνει την κλήση αλλά και τον προγραμματισμό του macro πιο φιλική.

Στον πρώτο πίνακα στερείται η δυνατότητα αρχικοποίησης 12 μεταβλητών αλλά αυτό δεν σημαίνει ότι ο προγραμματιστής δεν μπορεί να τις χρησιμοποιήσει μέσα στο παραμετρικό πρόγραμμα. Επίσης πρέπει να τονισθεί ότι 5 λέξεις δεν μπορούν να χρησιμοποιηθούν για την αρχικοποίηση τιμών σε τοπικές μεταβλητές επειδή είναι ήδη κατειλημμένες για την σύνταξη της κλήσης του macro η είναι κλεισμένες λέξεις όπως το όνομα του προγράμματος και ο αριθμός των γραμμών. Αυτές είναι οι G__ ,L__,N__,O__,P__.

Παράδειγμα: G65 P98765 W50. H35. R-2. F800.

<VARIABLES>

#23=50

#11=35

#18=-2

#9=800

G00 Z#18	Κατεβαίνει στο Z-2
G1 X0. YO F#9	Παίρνει θέση X0 Y0 με πρόωση F800
Y#11	Μετατόπιση μόνο κατά Y μέχρι το 35
X#23	Μετατόπιση μόνο κατά X μέχρι το 50
Y0.	Μετατόπιση μόνο κατά Y μέχρι το 0
X0.	Μετατόπιση μόνο κατά X μέχρι το 50
GZ20. M9	Ταχεία μετατόπιση μόνο κατά Z μέχρι το 20
M99	επιστροφή στο κυρίως Πρόγραμμα

14. Λήψη αποφάσεων & έλεγχος ροής προγράμματος

14.1 Δομημένος προγραμματισμός

Η εισαγωγή των μεταβλητών και η διαχείριση των παραμέτρων της εργαλειομηχανής παρέχουν κάποιες επιπλέον δυνατότητες οι οποίες όμως είναι στην ουσία περιορισμένες και συχνά ασήμαντες στην προγραμματιστική πρακτική. Η πραγματική δύναμη του παραμετρικού προγραμματισμού προκύπτει από τις δυνατότητες δομημένου προγραμματισμού που προσφέρει στον προγραμματιστή.

Ο δομημένος προγραμματισμός βασίζεται στην λογική όπου ένα πρόβλημα αναλύεται σε μικρότερες εργασίες (tasks) μέχρι να φτάσει σε βαθμό που μπορεί να βρεθεί λύση με απλές διαδικασίες. Ένα άλλο επίσης πολύ σημαντικό χαρακτηριστικό του δομημένου προγραμματισμού είναι ότι οι τελικές διαδικασίες μπορεί αναλόγως την περίσταση να εκτελούνται κατ' επιλογή ή/και να επαναλαμβάνονται για όσο απαιτηθεί. Έτσι, ο δομημένος προγραμματισμός έρχεται να συμπληρώσει και να ολοκληρώσει τις δυνατότητες του παραμετρικού προγραμματισμού.

Αν η εισαγωγή των τιμών στις μεταβλητές ενός προγράμματος καθορίζεται μόνο μια φορά από τον προγραμματιστή ουσιαστικά ο τρόπος που τις αξιοποιούμε είναι είτε για να παραμετροποιήσουμε κάποιες σταθερές ώστε να αλλάζουν μαζικά κατά την αρχή του προγράμματος είτε για να δημιουργήσουμε απλά παραμετρικά υποπρογράμματα. Πέρα από αυτή την τετριμμένη πρακτική, οι μεταβλητές είναι χρήσιμες επειδή μπορούν να αλλάζουν τις τιμές τους κατά την διάρκεια της εκτέλεσης του προγράμματος παρέχοντας μας την δυνατότητα να τις χρησιμοποιούμε σε δομές λήψεις αποφάσεων η όπου απαιτείται κάποια επαναληπτική διαδικασία και κάποιες τιμές θέλουμε να αλλάζουν ανά επανάληψη.

Οι εντολές προγραμματισμού που επεκτείνουν τον συμβατικό προγραμματισμό των εργαλειομηχανών και μας δίνουν τη δυνατότητα λήψης αποφάσεων, ανακατεύθυνσης της ροής και επαναλήψεων είναι σχεδόν ίδιες στην γλώσσα προγραμματισμού ηλεκτρονικών υπολογιστών Basic. Μπορεί η Basic να μην χρησιμοποιείται πια στον προγραμματισμό ηλεκτρονικών υπολογιστών, παρόλα αυτά στις εφαρμογές που αφορούν τις εργαλειομηχανές ξεκλειδώνει έναν νέο κόσμο δυνατοτήτων που μπορούν να κάνουν τα προγράμματα να είναι δυναμικά προσαρμόσιμα στις εκάστοτε αρχικές συνθήκες που ορίζονται από τον προγραμματιστή.

14.2 Ανακατεύθυνση ροής εκτέλεσης προγράμματος- GOTO

Μια από τις δομές που επιτρέπει την απλή ανακατεύθυνση της ροής της εκτέλεσης ενός προγράμματος είναι η απλή ανακατεύθυνση με την εντολή GOTO. Η εντολή αυτή μπορεί να ανακατευθύνει την ροή εκτέλεσης προς έναν δεδομένο δείκτη γραμμής στο τρέχον πρόγραμμα ή υποπρόγραμμα και συντάσσεται ως εξής:

GOTO n , όπου «n» είναι ο αριθμός της γραμμής που απαιτείται να ανακατευθυνθεί το εκτελούμενο πρόγραμμα.

Για παράδειγμα η σύνταξη GOTO 18 σημαίνει ότι με την εκτέλεση της το πρόγραμμα θα συνεχίσει την εκτέλεση του από την εντολή με δείκτη γραμμής N18. Επίσης η σύνταξη μπορεί να γίνει αξιοποιώντας και μεταβλητές στην θέση του δείκτη, συνεπώς η εντολή GOTO#18 σημαίνει ότι με την εκτέλεση της το πρόγραμμα θα συνεχίσει την εκτέλεση του από την εντολή με αριθμό δείκτη γραμμής που θα καθοριστεί από την τρέχουσα τιμή της μεταβλητής #18. Η παραπάνω εντολή μπορεί είτε να προσπεράσει μια αλληλουχία εντολών είτε να δημιουργήσει έναν ατέρμονα βρόχο.

14.3 Λήψη αποφάσεων – IF

Η δυνατότητα λήψης αποφάσεων είναι από τις πιο σημαντικές δυνατότητες που προσφέρει ο παραμετρικός προγραμματισμός. Οι αποφάσεις που λαμβάνονται κατά την εκτέλεση ενός προγράμματος βασίζονται στην έκβαση της εκτέλεσης λογικών πράξεων. Το αποτέλεσμα της εκτελέσιμης λογικής πράξης που περιλαμβάνεται σε μια σύνταξη μιας γραμμής λήψης απόφασης καθορίζει εάν θα εκτελεστεί η όχι το εκτελέσιμο μέρος της απόφασης.

14.4 Λογικές πράξεις

Οι λογικές πράξεις βασίζονται σε συγκρίσεις μεταξύ αριθμητικών τιμών και το αποτέλεσμα τους είναι μια δυαδική τιμή (0|1 – False|True). Οι προγραμματιστικές δυνατότητες που δίνονται είναι η απευθείας σύγκριση δυο αριθμητικών τιμών και

φυσικά η πραγματοποίηση λογικών πράξεων μεταξύ των αποτελεσμάτων των συγκρίσεων.

Οι τελεστές σύγκρισης αριθμητικών τιμών περιλαμβάνονται στον παρακάτω πίνακα:

τελεστές	**Σημασία**
EQ	Ίσο με (=)
NE	Διάφορο με (≠)
GT	Μεγαλύτερο από (>)
GE	Μεγαλύτερο από ή ίσο με (≥)
LT	Μικρότερο από (<)
LE	Μικρότερο από ή ίσο με (≤)

Η σύνταξη των αριθμητικών συγκρίσεων γίνεται παρεμβάλλοντας τον αντίστοιχο τελεστή μεταξύ των τιμών που θέλουμε να συγκρίνουμε. Για παράδειγμα εάν χρειάζεται να γίνει έλεγχος για την ισότητα των τιμών στις μεταβλητές #3 και #1 : #3 EQ #1 και το αποτέλεσμα που θα παίρναμε θα ήταν μια δυαδική τιμή (0|1 – False|True) αναλόγως αν η υπόθεση της σύγκρισης είναι σωστή η λάθος.

Τα αποτελέσματα των αριθμητικών συγκρίσεων μπορούν να συνδυαστούν με τη χρήση λογικών πράξεων που μας επιτρέπουν την σύνταξη πιο πολύπλοκων συνθηκών κατά τη λήψη αποφάσεων. Ο παραμετρικός προγραμματισμός επιτρέπει τρία βασικά είδη λογικών πράξεων (AND, OR, XOR) η συμπεριφορά των οποίων παρουσιάζεται στον παρακάτω πίνακα ανάλογα με τις εισόδους που καθορίζουμε κάθε φορά.

A	**B**	**AND**	**OR**	**XOR**
0	1	0	1	1
0	0	0	0	0
1	1	1	1	0

1	0	0	1	1

Αναλυτικότερα, η πράξη AND δίνει αληθή τιμή μόνο όταν και οι δυο τιμές που εισάγονται είναι ταυτόχρονα αληθείς ενώ η OR επιστρέφει αληθή τιμή όταν τουλάχιστον μια από τις δυο τιμές είναι αληθείς. Η XOR θα δώσει αληθή τιμή όταν τουλάχιστον η μια απ' τις δυο αλλά όχι ταυτόχρονα και οι δυο είναι αληθείς.

Οι είσοδοι στις αριθμητικές συγκρίσεις μπορεί να είναι είτε μεμονωμένες μεταβλητές η αριθμητικές τιμές είτε πολυπλοκότερες αριθμητικές πράξεις μεταξύ μεταβλητών και σταθερών τιμών. Στην δεύτερη περίπτωση θα προηγηθούν οι αριθμητικές πράξεις της σύγκρισης που θα δώσουνε να αποτέλεσμα (0|1 – False|True).

Η προτεραιότητα των λογικών πράξεων ορίζει ότι πρώτα εκτελούνται οι πράξεις AND και μετά οι πράξεις OR και XOR αντίστοιχα. Σε περίπτωση που απαιτείται διαφορετική σειρά εκτέλεσης των λογικών πράξεων μπορεί να γίνει χρήση τετράγωνων αγκυλών οπότε η εκτέλεση θα δώσει προτεραιότητα στις πράξεις εντός των αγκυλών.

Στην πράξη οι αποφάσεις που λαμβάνονται κατά τον προγραμματισμό είναι αποτελέσματα συνδυασμού συγκρίσεων μεταξύ αριθμητικών τιμών. Η εντολή IF μας δίνει τη δυνατότητα εκτέλεσης κατ' επιλογή μια η παραπάνω εντολές. Η σύνταξη της εμφανίζεται με δύο παραλλαγές :

IF [λογική έκφραση] GOTO n | IF [λογική έκφραση] then…

Η πρώτη επιλογή παρέχει την δυνατότητα να ανακατεύθυνσης της ροής του προγράμματος στον δείκτη μιας γραμμής εφόσον η λογική έκφραση είναι αληθής ενώ η δεύτερη επιλογή παρέχει την δυνατότητα να εκτέλεσης κατ' επιλογή μια αριθμητική πράξη. Παράδειγμα για την τροποποίηση της τιμής μιας μεταβλητής. Και στις δυο περιπτώσεις όμως το δεύτερο σκέλος της σύνταξης (GOTO | THEN) δεν θα εκτελεστεί εάν δεν είναι αληθής η λογική έκφραση που ελέγχεται.

Δυο ισοδύναμα παραδείγματα προγραμματισμού της εντολής IF αξιοποιώντας και τους δυο τρόπους σύνταξης θα μπορούσε να είναι:

…	…

#1=…	#1=…
#2=…	#2=…
IF[#1GT#2]GOTO5	IF[#1LT#2] THEN #2=[2*#1]
#2=[2*#1]	M05
N5 M05	…
…	…

Στο παραπάνω παράδειγμα φαίνεται ότι σε κάποιες περιπτώσεις η σύνταξη IF-THEN μπορεί να αντικαταστήσει και να απλοποιήσει την σύνταξη IF-GOTO όμως αν πρέπει να ληφθεί η απόφαση αν θα εκτελεστεί η όχι μια πιο μακροσκελή σειρά από εντολές τότε ίσως η σύνταξη IF-GOTO να είναι μονόδρομος όπως φαίνεται στο παρακάτω παράδειγμα…

…

#1=…

#2=…

IF[#1GT#2] GOTO5

#2=2*#1

G00 X#1

G01 Y#2 F500

N5 M05

…

14.5 Προγραμματισμός επαναληπτικών διαδικασιών

Οι δυνατότητες που ξεκλειδώνει η λήψη αποφάσεων δεν περιορίζονται μόνο στην κατ’ επιλογή εκτέλεση μιας σειράς εντολών αλλά και στην εκτέλεση επαναληπτικών διαδικασιών που μπορούν να μειώσουν το μήκος των προγραμμάτων. Οι διαδικασίες αυτές επιτρέπουν τον προγραμματισμό και την επαναλαμβανόμενη εκτέλεση μιας σειρά εντολών που εάν έχουν οριστεί βάσει μεταβλητών θα δίνουν κάθε φορά

διαφορετικό αποτέλεσμα. Για παράδειγμα στον προγραμματισμό ενός pocket που το βασικό toolpath θα επαναλαμβάνεται κάθε φορά σε διαφορετικό βάθος κοπής.

14.6 Προγραμματισμός επανάληψης με την εντολή IF-GOTO

Ο πιο απλός τρόπος να προγραμματισμού μια επαναληπτικής διαδικασίας είναι με την αξιοποίηση της εντολής λήψης αποφάσεων και πιο συγκεκριμένα με την σύνταξη IF-GOTO. Με αυτό τον τρόπο επιτρέπεται να ανακατευθύνουμε την ροή του προγράμματος βάσει της απόφασης που θα ληφθεί (true/false) κατά την εκτέλεση της λογικής πράξης που περιλαμβάνεται στην σύνταξη. Πρέπει να σημειωθεί ότι ο τρόπος αυτός δεν ενδείκνυται γιατί η δομή της επαναληπτικής διαδικασίας δεν είναι ξεκάθαρη και μπορεί να συγχύσει τον αναγνώστη του προγράμματος παρόλα αυτά έχει κάποια σημεία όπου υπερέχει σε σχέση με την κανονική δομή που προσφέρει ο παραμετρικός προγραμματισμός για σύνταξη επαναληπτικών διαδικασιών (WHILE).

Παρακάτω ακολουθεί ένα παράδειγμα επαναληπτικής διαδικασίας με την εντολή λήψης απόφασης. Με αυτή την σύνταξη οι εντολές που πρόκειται να επαναληφθούν θα εκτελεστούν τουλάχιστον μια φορά πριν κληθεί η εντολή IF-GOTO που θα εξετασθεί η λογική συνθήκη η οποία θα επιτρέψει η όχι την επαναληπτική διαδικασία

…

N10 #1=…

#2=…

… (κώδικας προς επανάληψη)

IF[#1GT#2]GOTO10

…

14.7 Προγραμματισμός επαναληπτικών διαδικασιών με την εντολή WHILE

Η εντολή WHILE χρησιμοποιείται στον προγραμματισμό επαναληπτικών διαδικασιών επιτρέποντας την επανάληψη μιας σειράς εντολών για όσο μια λογική έκφραση έχει αληθή τιμή και η σύνταξη της έχει ως εξής:

WHILE [Λογική έκφραση] DO n

…

…

…

END n

,όπου n είναι ο δείκτης που ορίζει την αρχή και το τέλος του βρόγχου επανάληψης (πχ. 1,2,3 κλπ.)

Όπως φαίνεται από την σύνταξη, ο έλεγχος της λογικής έκφρασης γίνεται πρίν την εκτέλεση των εντολών που περιλαμβάνονται στον βρόγχο επανάληψης. Αυτό πρακτικά σημαίνει ότι σε αντίθεση με την εντολή IF-GOTO, η σύνταξη με την εντολή WHILE δεν θα εκτελέσει ποτέ τις εντολές εντός του βρόγχου επανάληψης εάν η λογική έκφραση δεν είναι αληθής. Επιπλέον οι εντολές που περιλαμβάνονται εντός του βρόγχου επανάληψης είναι ευδιάκριτα διαχωρισμένες από τον υπόλοιπο κώδικα, πράγμα που κάνει τον έλεγχο του προγράμματος ευκολότερο.

Κατά τον προγραμματισμό επαναλήψεων με αυτό τον τρόπο είναι επίσης δυνατό οι εντολές while να εμφωλευθούν (nesting) ώστε να δημιουργηθούν πιο σύνθετοι βρόγχοι επανάληψης. Συνήθως ο βαθμός εμφώλευσης είναι ίσος με 3 που σημαίνει ότι μπορούμε να προγραμματίσουμε διαδικασίες που να έχουν έως τρία επίπεδα επαναλήψεων χωρίς αυτό να απαγορεύει να έχουμε παραπάνω από μια ανεξάρτητη διαδικασία σε κάθε επίπεδο όπως φαίνεται στο παρακάτω παράδειγμα:

WHILE [Λογική έκφραση] DO1	WHILE [Λογική έκφραση] DO1	WHILE [Λογική έκφραση] DO1
…	…	…
END1	WHILE [Λογική έκφραση] DO2	WHILE [Λογική έκφραση] DO2
…	…	…

WHILE [Λογική έκφραση] DO1 … END1	WHILE [Λογική έκφραση] DO3 … END3 … END2 … END1	WHILE [Λογική έκφραση] DO3 … END3 … END2 … WHILE [Λογική έκφραση] DO2 … END2 … END1

Η διακοπή της επαναληπτικής διαδικασίας είναι εφικτή ανακατευθύνοντας τη ροή εκτός του βρόγχου επανάληψης. Αυτό μπορεί να γίνει με την εκτέλεση μιας εντολής GOTO που θα είναι αποτέλεσμα μια εντολής λήψης απόφασης (IF-GOTO). Όπως φαίνεται στο παρακάτω παράδειγμα ο βρόγχος θα επαναλαμβάνεται έως ότου γίνει ψευδής η λογική έκφραση της WHILE ή όταν γίνει αληθής η λογική έκφραση της IF-GOTO.

…

WHILE [Λογική έκφραση 1] DO1

…

IF[Λογική έκφραση 2] GOTO12

…

END1

…

N12…

…

Θεωρείται εσφαλμένη η σύνταξη σε περιπτώσεις που οι εντολές WHILE εμπλέκονται μεταξύ τους (περίπτωση 1) ή περιπτώσεις που η χρήση της εντολής IF-GOTO ανακατευθύνει την εκτέλεση του προγράμματος εντός του βρόγχου επανάληψης μια εντολής WHILE (περίπτωση 2).

Αναλυτικότερα:

Περίπτωση 1	**Περίπτωση 2**
…	…
WHILE [Λογική έκφραση] DO1	IF [Λογική έκφραση] GOTO10
…	…
WHILE [Λογική έκφραση] DO2	WHILE [Λογική έκφραση] DO1
…	…
END1	N10
…	…
END2	END1

15. Σφάλματα & μηνύματα ειδοποίησης

Η δυναμικότητα της σύνταξης των παραμετρικών προγραμμάτων και υποπρογραμμάτων κρύβει πολλές φορές κινδύνους κατά την εκτέλεση του προγράμματος που θα πρέπει να προβλέπονται, να ελέγχονται και μέσω κατάλληλων ενεργειών να ειδοποιούν τον χειριστή για πιθανά λάθη κατά τον χειρισμό. Τέτοιοι κίνδυνοι μπορεί να περιλαμβάνουν σφάλματα που πρόκειται να προκύψουν κατά την εκτέλεση αριθμητικών η λογικών πράξεων. Εξάλλου ο χειριστής-εκτελεστής του προγράμματος δεν είναι σίγουρο ότι θα είναι πάντα και ο ίδιος ο προγραμματιστής που ανέπτυξε το πρόγραμμα. Αυτό σημαίνει ότι ίσως ο χειριστής να μην λαμβάνει υπόψιν του πιθανές παραδοχές που έχει θέσει ο προγραμματιστής.

Ο προγραμματιστής λοιπόν πρέπει να προβλέπει τα όποια προβλήματα μπορεί να προκύψουν κατά την εκτέλεση του προγράμματος που αναπτύσσει και στην

συνέχεια έχοντας ξεκάθαρη εικόνα να μπορεί να τα διαχωρίσει σε αυτά που θα οδηγήσουν σε σφάλμα εκτέλεσης ή υπολογισμού και σε αυτά που είναι μικρότερης σημασίας και ο τελικός χρήστης του προγράμματος θα πρέπει απλά να ενημερωθεί.

Έχοντας διακρίνει και αξιολογήσει την κρισιμότητα των πιθανών προβλημάτων, ο προγραμματιστής έχει δυο εναλλακτικές για να ειδοποιήσει τον τελικό χρήστη του παραμετρικού προγράμματος. Αν το πρόβλημα πρόκειται να οδηγήσει σε σφάλμα εκτέλεσης ή υπολογισμού τότε θα πρέπει να ενεργοποιηθεί ένα σφάλμα το οποίο θα διακόψει την εκτέλεση του προγράμματος ώστε να μην χρειαστεί να προκύψει το πρόβλημα εν μέσω της εκτέλεσης.

Οι ελεγκτές των εργαλειομηχανών δίνουν την δυνατότητα προγραμματισμού σφάλματος ή μηνυμάτων ειδοποίησης στον παραμετρικό προγραμματισμό και η σύνταξη έχει ως εξής:

N___#3000=___(μήνυμα για τον τελικό χρήστη)

Όπου N είναι ο δείκτης της γραμμής του σφάλματος ενώ το #3000=___ είναι η τιμή της παραμέτρου του σφάλματος και παίρνει τιμές από 0-200.

Στην περίπτωση που ο προγραμματιστής δεν θέλει να διακόψει εντελώς την εκτέλεση του προγράμματος μπορεί αντί για σφάλμα να αφήσει ένα μήνυμα ειδοποίησης το οποίο κάνει προσωρινή παύση έως ότου ο χειριστής πιέσει το κουμπί cycle start όπου συνεχίζεται η εκτέλεση του προγράμματος από εκεί που είχε σταματήσει. Η σύνταξη μηνύματος ειδοποίησης γίνεται ακριβώς όπως και με την σύνταξη των σφαλμάτων με την μόνη διαφορά ότι εδώ γίνεται η χρήση της παραμέτρου 3006 αντί της 3000.

Ιδιαίτερη προσοχή πρέπει να δίνεται στον τρόπο κλήσης των μηνυμάτων ώστε να εξασφαλίζεται ότι δεν θα κληθούν καταλάθος. Μια καλή πρακτική είναι να συγκεντρώνονται όλα τα μηνύματα σφαλμάτων και ειδοποιήσεων στην αρχή ή στο τέλος του προγράμματος και η κανονική εκτέλεση του προγράμματος να ανακατευθύνεται με την εντολή GOTO ώστε να μην εκτελεστούν παρά μόνο όταν χρειαστούν. Επίσης θα πρέπει να λαμβάνεται υπόψιν ότι η έκταση τέτοιων μηνυμάτων είναι συνήθως περιορισμένη (τυπικά 26 χαρακτήρες) και για αυτό είναι χρήσιμο ο προγραμματιστής να φροντίζει εντός του κώδικα να περιλαμβάνει με τη μορφή σχολίων περεταίρω επεξήγηση των σφαλμάτων ώστε ο χειριστής να μπορεί να ανατρέχει και να ενημερώνεται αναλυτικότερα. Παρακάτω δίνεται ένα τυπικό

παράδειγμα που οι κλήσεις των ειδοποιήσεων και των σφαλμάτων δίνονται στο τέλος του προγράμματος.

```
%
#1=___(ακτίνα κοπτικού)
#2=___(ράδιο εσωτ. γωνίας στο pocket)
#3=…
…
IF[#1GT#2]GOTO1001
…
GOTO9999
(ERROR MESSAGES)
N1001 #3000=101 ( TOO BIG TOOL DIAMETER)
N1002…
N1003…
N9999
M99
(ERRORS & WARNINGS TROUBLESHOOTING)
(ERROR 101: DECREASE TOOL DIAMETER)
(ERROR 102: …)
%
```

Η εκτέλεση του σφάλματος στο παραπάνω παράδειγμα θα διακόψει την εκτέλεση του προγράμματος και θα εμφανίσει στην οθόνη του ελεγκτή το μήνυμα: 101 TOO BIG TOOL DIAMETER.

16. Δημιουργία εξατομικευμένων G & M εντολών

Γενικά οι ελεγκτές που υποστηρίζουν τον παραμετρικό προγραμματισμό επιτρέπουν την δημιουργία συνήθως έως 10 νέων G και 10 νέων M εντολών. Οι εντολές αυτές ουσιαστικά χρησιμοποιούνται για να καλέσουν ένα παραμετρικό υποπρόγραμμα όπως θα το καλούσαν και οι εντολές G65 και G66 με την διαφορά ότι ο χειριστής δεν χρειάζεται να θυμάται το όνομα που έχει αποθηκευτεί στην μνήμη του ελεγκτή. Πρακτικά αυτό δίνει την δυνατότητα στον προγραμματιστεί να επεκτείνει τις δυνατότητες του ελεγκτή της εργαλειομηχανής εισάγοντας νέες εντολές.

Για τον καθορισμό του αριθμού της εντολής που θα δημιουργήσει ο προγραμματιστής θα πρέπει να λάβει υπόψιν του τις υπάρχουσες εντολές ώστε να επιλέξει έναν αριθμό εντολής που να μην είναι ήδη καταχωρημένος. Σε περίπτωση που χρησιμοποιηθεί αριθμός εντολής από άλλη εντολή ενδέχεται να δημιουργηθεί πρόβλημα κάνοντας την εργαλειομηχανή να παραβλέψει ή/και να σβήσει την υπάρχουσα εντολή. Για τον καθορισμό της εντολής G ή M που καλεί ένα παραμετρικό υποπρόγραμμα ακολουθείται η εξής διαδικασία:

1) Δίδεται στο παραμετρικό υποπρόγραμμα όνομα σε συγκεκριμένο εύρος αριθμών (συνήθως O90010-O90019 για εντολές G και O90020-O90029 για εντολές M).

2) Αποθηκεύεται ο επιθυμητός αριθμός (##) της εντολής G ή M στην παράμετρο του ελεγκτή που αντιστοιχεί στο αντίστοιχο όνομα προγράμματος (συνήθως #6050-#6059 για εντολές G και #6080-#6089 για εντολές M).

3) Το macro πλέον μπορεί να κληθεί με την εντολή G## ή M##

Στον παρακάτω πίνακα δίνεται τυπικά μια αντιστοίχιση των ονομάτων προγράμματος με τις παραμέτρους που τους αντιστοιχούν.

Εντολές G		Εντολές M	
Παράμετρος	Πρόγραμμα	Παράμετρος	Πρόγραμμα
#6050	O9010	#6080	O9020
#6051	O9011	#6081	O9021
#6052	O9012	#6082	O9022

#6053	O9013	#6083	O9023
#6054	O9014	#6084	O9024
#6055	O9015	#6085	O9025
#6056	O9016	#6086	O9026
#6057	O9017	#6087	O9027
#6058	O9018	#6088	O9028
#6059	O9019	#6089	O9029

Εκτός της κλήσης παραμετρικών υποπρογραμμάτων με την χρήση εντολών M μπορούν αντίστοιχα να κληθούν και μη παραμετρικά υποπρογράμματα με εντολές M όπως ακριβώς γίνεται και με τις εντολές M98 / M97. Υπενθυμίζεται ότι η διαφορά στην κλήση των δύο είναι ότι τα παραμετρικά δέχονται σαν ορίσματα κατά την κλήση τους μεταβλητές ενώ τα συμβατικά υποπρογράμματα είναι απλώς αυτόνομα τμήματα προγράμματος. Η σύνταξη κλήσης με κάποια εντολή M για L επαναλήψεις έχει ως εξής: M__ L__

Η διαδικασία καθορισμού εντολής M που θα καλεί ένα συγκεκριμένο υποπρόγραμμα είναι όμοια με όσα αναφέρθηκαν παραπάνω και στον παρακάτω πίνακα δίνεται τυπικά μια αντιστοίχιση των ονομάτων προγράμματος με τις παραμέτρους που τους αντιστοιχούν.

Εντολές M	
Παράμετρος	Πρόγραμμα
#6070	O9000
#6071	O9001
#6072	O9002
#6073	O9003
#6074	O9004

#6075	O9005
#6076	O9006
#6077	O9007
#6078	O9008
#6079	O9009

Η κλήση υποπρογραμμάτων με εντολές M υποστηρίζεται μόνο σε ελεγκτές που υποστηρίζουν παραμετρικό προγραμματισμό διότι θα πρέπει να καταχωρηθεί τιμή στην παράμετρο που δείχνει στο όνομα του εκάστοτε υποπρογράμματος.

17. Probing

Μια από τις σημαντικότερες χρήσεις του παραμετρικού προγραμματισμού στις εργαλειομηχανές CNC είναι η πραγματοποίηση κύκλων probing για τον αυτόματο καθορισμό των offset του τεμαχίου και των εργαλείων καθώς και τον ποιοτικό έλεγχο πάνω στην εργαλειομηχανή.

Αξίζει να σημειωθεί ότι δεν είναι δυνατόν να πραγματοποιηθεί ολοκληρωμένος μετρητικός έλεγχος των τεμαχίων με την εργαλειομηχανή που κατασκευάστηκαν αφού δεν πρόκειται να δείξει κάποια πραγματική απόκλιση στις διαστάσεις η στην γεωμετρία. Το μόνο που μπορεί να φανεί σε αυτό το στάδιο και μόνο εφόσον το probe που χρησιμοποιείται είναι διαπιστευμένο ακολουθώντας τις συνήθεις διαδικασίες του καλιμπραρίσματος όπως αυτές προβλέπονται από τους κατασκευαστές, είναι αν υπάρχει κάποια φθορά η παραμόρφωση στο κοπτικό εργαλείο κατά την κατεργασία

Η Τεχνολογία των probe χρησιμοποιεί αισθητήρες οι οποίοι είναι συνδεδεμένοι ασύρματα με έναν κεντρικό δέκτη (εικόνα 4, n.d.) που βρίσκεται μέσα στην εργαλειομηχανή ο οποίος είναι συνδεδεμένος ενσύρματα με τον ελεγκτή της εργαλειομηχανής. Ο δέκτης στέλνει ένα σήμα ενεργοποίησης στον ελεγκτή κάθε φορά που ενεργοποιείται κάποιος αισθητήρας. Οι αισθητήρες αυτοί (Hall sensor) αξιοποιούνται τόσο σε probes που τοποθετούνται στην άτρακτο σαν εργαλείο για μετρήσεις πάνω στο τεμάχιο όσο και ανεξάρτητα για την μέτρηση των εργαλείων

(διάμετρος, μήκος). Υπάρχουν δηλαδή δύο τύποι probes. Τα probes που μετράνε μεγέθη που αφορούν εργαλεία (εικόνα 3, n.d.) και τα probes που μετράνε μεγέθη που αφορούν το τεμάχιο (εικόνα 2, n.d.).

Εικόνα 5, probe για μηδενισμό τεμαχίου

Εικόνα 6, probe για μηδενισμό κοπτικών εργαλείων

Εικόνα 7, Δέκτης για αναμετάδοση σήματος από τους αισθητήρες στον ελεγκτή της εργαλειομηχανής

Συνδυάζοντας τις δυνατότητες του παραμετρικού προγραμματισμού με την λειτουργικότητα των probes ελαχιστοποιείται η πιθανότητα παραγωγής τεμαχίων εκτός προδιαγραφών και οι αξιοποίηση τους είναι κομβικής σημασίας για την παραγωγή τεμαχίων με πολύ μικρές γεωμετρικές και διαστατικές ανοχές και ειδικότερα όταν απαιτείται μεγάλος αριθμός τεμαχίων.

Παρόλα αυτά για να γίνει λόγος για κατεργασίες ακριβείας θα πρέπει να χρησιμοποιούντα μόνο probes ακριβείας τα οποία διακριβώνονται σε τακτά χρονικά διαστήματα (calibration) και εκτός αυτού έχει εξασφαλισθεί ότι περιορίζονται οι όποιοι άλλοι παράγοντες εισαγωγής σφάλματος κατά την μέτρηση με το probe – πχ. θερμοκρασία δωματίου, προθέρμανση εργαλειομηχανής, ποιότητα κοπτικών εργαλείων, γεωμετρική κατάσταση της εργαλειομηχανής (αλφάδιασμα).

Τα probes που είναι υπεύθυνα για την μέτρηση των κοπτικών τοποθετούνται μόνιμα σε ένα σημείο της εργαλειομηχανής το οποίο συνήθως ορίζεται σε κάποια ελεύθερη περιοχή του τραπεζιού. Σε τόρνους η σε πολυαξονικές εργαλειομηχανές γίνεται χρήση βραχίονα για την τοποθέτηση-απομάκρυνση του probe. Τα probes που είναι υπεύθυνα για τον μηδενισμό και τον μετρητικό έλεγχο των τεμαχίων φορτώνονται στην εργαλειομηχανή όπως και τα κοινά εργαλεία με την διαφορά ότι κατά την κίνηση τους δεν περιστρέφονται και ανάλογα τον τύπο του probe μπορεί να απαιτείται και ο προσανατολισμός της ατράκτου (spindle orientation). Επιπλέον λόγο της ευαίσθητης φύσης των probe συνήθως ορίζονται ως βαριά εργαλεία ώστε να γίνει πιο αργά η κλήση τους.

18. Προγραμματισμός Probing

Όλα τα probes στηρίζονται στο γεγονός ότι κάθε φορά που ενεργοποιείτε ένας αισθητήρας στέλνει ένα σήμα στον δέκτη και αυτός με την σειρά του αναμεταδίδει ένα σήμα ελέγχου στον ελεγκτή το οποίο ονομάζεται “skip signal”.

Αυτό το σήμα ελέγχου από μόνο του δεν ενεργοποιεί καμία διαδικασία στην εργαλειομηχανή. Οπότε αν έχει προγραμματιστεί μια κίνηση για να ακουμπήσει η ακίδα με την εντολή G01 το probe δεν θα σταματήσει όταν δοθεί το σήμα από τον αισθητήρα αλλά θα συνεχίσει μέχρι να ολοκληρωθεί κανονικά ολόκληρη η κίνηση που έχει προγραμματιστεί. Για τις διαδικασίες του probing εισάγεται η εντολή G31. Η εντολή αυτή ονομάζεται Skip function ή Feed until skip και συντάσσεται όμοια με την G01.

G31 X_ Y_ Z_ F_

Όπου X,Y,Z είναι οι άξονες που μετατοπίζεται το probe

Και F είναι η ταχύτητα πρόωσης (federate)

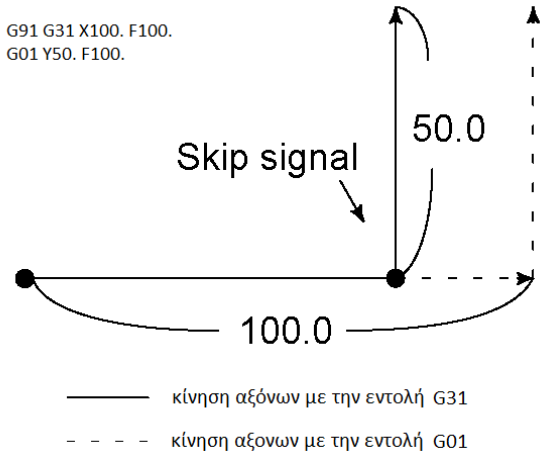
Η διάφορες της G01 και της G31 (εικόνα 5, n.d.) είναι ότι η πρώτη είναι modal ενώ η δεύτερη όχι και ότι η G31 θα διακόψει την προγραμματισμένη κίνηση μόλις ο αισθητήρας του probe ενεργοποιηθεί. Πρακτικά όταν προγραμματίσουμε μια ευθύγραμμη κίνηση με την εντολή G31 η εργαλειομηχανή θα εκκινήσει τους άξονες και θα αρχίσει να εκτελεί κανονικά την εντολή σαν να ήταν G01 αλλά όταν ο αισθητήρας ενεργοποιηθεί ο δέκτης θα στείλει στον ελεγκτή το skip signal και θα διακοπεί αμέσως η κίνηση όλων των αξόνων ακόμα και αν δεν έχει ολοκληρωθεί η προγραμματισμένη κίνηση (εικόνα 5). Παράλληλα η θέσεις των αξόνων αποθηκεύονται σε μια σειρά μεταβλητών ώστε να μπορούν να αξιοποιηθούν στο πρόγραμμα που συντάσσεται.

Οι παράμετροι που αξιοποιούνται ώστε να αποθηκευτούν οι θέσεις των αξόνων μετά από την ενεργοποίηση του skip signal είναι:

#5061 για τον άξονα X

#5062 για τον άξονα Y

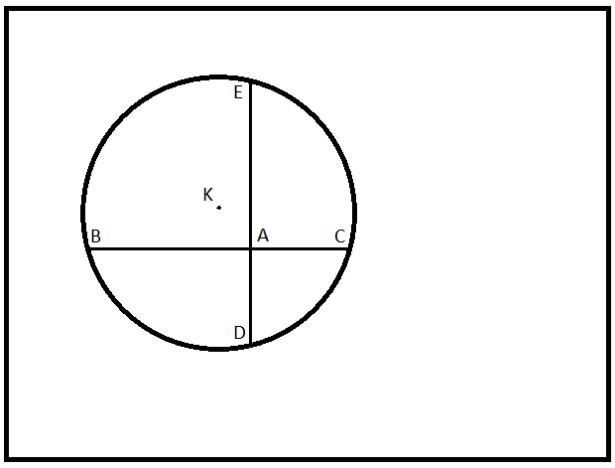
#5063 για τον άξονα Z



Εικόνα 8, σύγκριση toolpath G31-G01

Κατά την ανάπτυξη παραμετρικών προγραμμάτων που περιέχουν διαδικασίες probing πρέπει να δίνεται ιδιαίτερη προσοχή διότι οι κινήσεις προς κάποια επιφάνεια προγραμματίζονται ως κινήσεις μέσα στο υλικό και το skip signal αναλαμβάνει να σταματήσει τους άξονες ώστε να αποφευχθεί η σύγκρουση. Επίσης προσοχή απαιτείται στις κινήσεις απομάκρυνσης του probe από την επιφάνεια μετά το skip signal διότι η G31 **δεν** είναι modal. Κατά τον προγραμματισμό διαδικασιών probing συνίσταται η χρήση σχετικών συντεταγμένων (G91) ώστε να ελαχιστοποιήσουμε τις πιθανότητες σύγκρουσης του probe με το τεμάχιο.

Παρακάτω παρατίθεται ενδεικτικό παραμετρικό υποπρόγραμμα που αναλαμβάνει να βρει το κέντρο ενός κύκλου Φ200.



O88888	Όνομα προγράμματος
#10=#5061	Αποθήκευση τρέχουσας συντεταγμένης X στην μεταβλητή #10
G91 G31 X-200. F200.	Κίνηση προς το σημείο B με σχετικά γρήγορη πρόωση

G01 X2. F200.	Απομάκρυνση μετά το skip signal
G31 X-5. F50.	Ξανακουμπάει με αργή πρόωση
#11=#5061	Αποθηκεύει την συντεταγμένη X στο σημείο B
G31 X200. F200. G01 x-2 F200. G31 X5. F50.	Probing στο σημείο C με την ιδιά στρατηγική όπως στο σημείο B
#12=#5061	Αποθηκεύει την συντεταγμένη X στο σημείο C
#13=[#12-#11]/2	Υπολογισμός κέντρου χορδής BC
G90 X#13	Μετατόπιση στο κέντρο της χορδής BC
G91 G31 Y-200. F200. G01 Y2. F200. G31 Y-5. F50.	Probing στο σημείο D με την ιδιά στρατηγική
#14=#5062	Αποθηκεύει την συντεταγμένη Y στο σημείο D

G31 Y200. F200. G01 Y-2. F200. G31 Y5.F50	Probing στο σημείο Ε με την ιδιά στρατηγική
#15=#5062	Αποθηκεύει την συντεταγμένη Υ στο σημείο Ε
#16=[#15-#14]/2	Υπολογισμός κέντρου χορδής ED
G90 G01 X#13 Y#16 F200.	ευθύγραμμη μετατόπιση στα κέντρα των χορδών (κέντρο του κύκλου)
G53 Z0.	Ευθύγραμμη μετατόπιση στο Z0 του G52
M99	Τέλος υποπρογράμματος, επιστροφή στο κύριο πρόγραμμα.

Όπως φαίνεται στο παραπάνω παράδειγμα κάθε φορά που το probe παίρνει μια μέτρηση η στρατηγική που ακολουθείται περιλαμβάνει αρχικά μια γρήγορη σχετικά επαφή, απομάκρυνση από το τεμάχιο και ξανά μια δεύτερη επαφή με πιο αργή πρόωση. Η πρόωση που το probe κατευθύνεται στο τεμάχιο είναι ένας σημαντικός παράγοντας που επηρεάζει την ακρίβεια της μέτρησης. Για αυτό προγραμματίζεται η δεύτερη προσέγγιση με μικρότερη πρόωση.

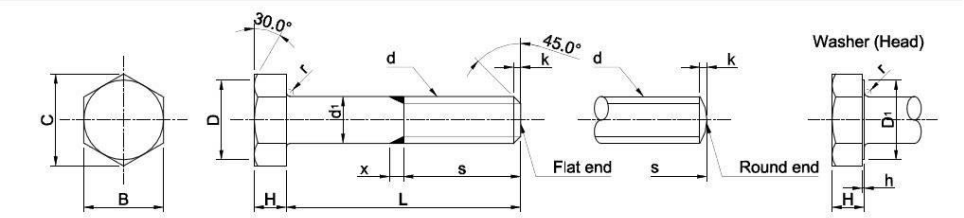
Η πρόωση της δεύτερης προσέγγισης πρέπει να είναι σταθερή για κάθε μέτρηση που πραγματοποιείτε για να διασφαλιστεί ότι δεν προκύπτουν σφάλματα από την παραμόρφωση της ακίδας κατά την επαφή της με το τεμάχιο αλλά και από τυχόν σφάλματα κατά το σταμάτημα των αξόνων της εργαλειομηχανής. Είναι σημαντικό επίσης να χρησιμοποιηθεί η ίδια πρόωση και κατά την διαπίστευση και ρύθμιση του probe (probe calibration). Τέλος, σε υποπρογράμματα που περιλαμβάνουν probing πρέπει πάντα να απενεργοποιείται η δυνατότητα του χειριστή να μεταβάλει την ενεργή ταχύτητα πρόωσης (feed override) γιατί αυτό θα επηρέαζε άμεσα την ακρίβεια των μετρήσεων.

19. Οικογένειες κομματιών

Η ανάγκη να κατασκευαστούν όμοιες γεωμετρίες είναι πολύ συχνό φαινόμενο και αυτή είναι μια ακόμα από τις περιπτώσεις που ο παραμετρικός προγραμματισμός λύνει τα χεριά στον προγραμματιστή. Επιπλέον είναι η μόνη μέθοδος προγραμματισμού η οποία μπορεί να παράγει προγράμματα που θα «δουλέψουν» για περισσότερες από μια γεωμετρίες.

Αναφερόμενοι στον όρο ‘οικογένειες κομματιών’ όμως δεν σημαίνει πάντα ότι πρόκειται για παρόμοιες γεωμετρίες αποκλειστικά. Μπορεί όντως οι γεωμετρίες να αποκλίνουν μεταξύ των κομματιών αλλά αυτό είναι μόνο μια περίπτωση οικογένειας κομματιών. Υπάρχουν και άλλες περιπτώσεις που μπορεί να χρησιμοποιηθεί το ίδιο παραμετρικό πρόγραμμα για παραπάνω από ένα κομμάτι. Τέτοιες περιπτώσεις είναι:

- όμοια κομμάτια
- ιδιά κομμάτια για διαφορετικά υλικά (τροποποιήσεις σε συνθήκες κοπής)
- κομμάτια υπό κλίμακα
- διαφορετικό τρόπο συγκράτησης
- διαφορετικό φασεολόγιο
- κομμάτια σχεδιασμένα παραμετρικά (εικόνα 6, n.d.)



HEXAGONAL HEAD BOLT STANDARD SIZE (FROM JIS B 1180, 1938) (TABLE 83)							
(d)	d_1	H	B	C	D	r	k
Nominal Size of Thread	Basic Dimension	Basic Dimension	Basic Dimension	Approx.	Approx.	Max.	Approx.
M 3 x 0.5	3	2	5.5	6.4	5.3	0.2	0.6
(M 3.5)	3.5	2.4	6	6.9	5.8	0.2	0.6
M 4 x 0.7	4	2.8	7	8.1	6.8	0.3	0.8
(M 4.5)	4.5	3.2	8	9.2	7.8	0.3	0.8
M 5 x 0.8	5	3.5	8	9.2	7.8	0.3	0.9
M 6	6	4	10	11.5	9.8	0.5	1
(M 7)	7	5	11	12.7	10.7	0.5	1
M 8	8	5.5	13	15	12.6	0.5	1.2
M 10	10	7	17	19.6	16.5	0.8	1.5
M 12	12	8	19	21.9	18	0.8	2
(M 14)	14	9	22	25.4	21	0.8	2
M 16	16	10	24	27.7	23	1.2	2
(M 18)	18	12	27	31.2	26	1.2	2.5
M 20	20	13	30	34.6	29	1.2	2.5
(M 22)	22	14	32	37	31	1.2	2.5
M 24	24	15	36	41.6	34	1.6	3

HEXAGONAL HEAD BOLT STANDARD SIZE (FROM JIS B 1180, 1938) (TABLE 83)							
(d)	d_1	H	B	C	D	r	k
Nominal Size of Thread	Basic Dimension	Basic Dimension	Basic Dimension	Approx.	Approx.	Max.	Approx.
(M 27)	27	17	41	47.3	39	1.6	3
M 30	30	19	46	53.1	44	1.6	3.5
(M 33)	33	21	50	57.7	48	2	3.5
M 36	36	23	55	63.5	53	2	4
(M 39)	39	25	60	69.3	57	2	4
M 42	42	26	65	75	62	2	4.5
(M 45)	45	28	70	80.8	67	2	4.5
M 48	48	30	75	86.5	72	2	5
(M 52)	52	33	80	92.4	77	2.5	5
M 56	56	35	85	98.1	82	2.5	5.5
(M 60)	60	38	90	104	87	2.5	5.5
M 64	64	40	95	110	92	2.5	6
(M 68)	68	43	100	115	97	2.5	6
M 72	72	45	105	121	102	2.5	6
(M 76)	76	48	110	127	107	3	6
M 80	80	50	115	133	112	3	6

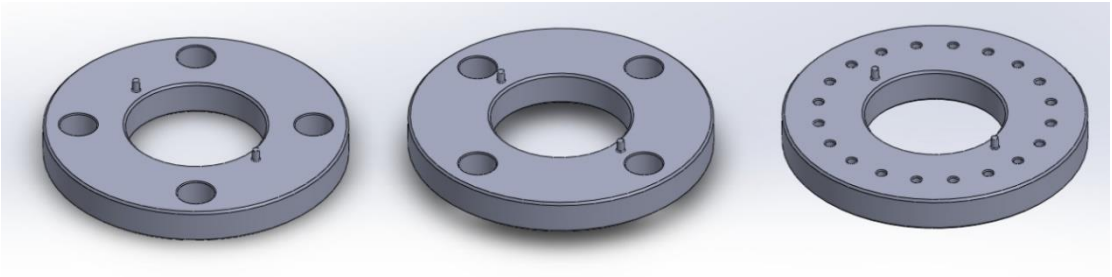
ISO METRIC	
NOMINAL LENGTH OF BOLT	LENGTH OF THREAD
Up to and including 125mm	2D + 6mm
Over 125mm up to and including 200mm	2D + 12mm
Over 200mm	2D + 25mm

Εικόνα 9 Παράδειγμα Παραμετρικού σχεδιασμού

Σε περιπτώσεις τέτοιων προγραμμάτων η λογική που ακολουθείται είναι η εξής:

Είτε ο χειριστής ορίζει κατά την κλήση του macro τα μεταβαλλόμενα μεγέθη δίνοντας τις επιθυμητές τιμές στα arguments, είτε ο προγραμματιστής έχει γράψει συγκεκριμένα κομμάτια κώδικα μέσα στο macro για κάθε περίπτωση ξεχωριστά και ανάλογα με την τιμή που θα πάρει κάποιο argument κατά την κλήση θα εκτελεστεί και το αντίστοιχο κομμάτι κώδικα. Αυτό προτιμάται σε περιπτώσεις οπού οι επιλογές είναι λίγες και συγκεκριμένες. Όταν οι μεταβαλλόμενες τιμές έχουν μεγαλύτερο εύρος και οι επιλογές είναι περισσότερες (πολλές φορές άπειρες) τότε συνηθίζεται η πρώτη περίπτωση οπού ο χειριστής καλείτε να ορίσει αυτές τις τιμές μέσω των arguments. Βέβαια αυτό προϋποθέτει ότι ο χειριστής έχει γνώση του τι κάνει το κάθε argument η μπορεί να διαβάσει και να κατανοήσει το παραμετρικό πρόγραμμα. Αυτό απαιτείται διότι ο χειριστής και ο προγραμματιστής δεν είναι πάντα το ίδιο άτομο.

Στο παρακάτω παράδειγμα (εικόνα 10, n.d.) είναι εμφανές πόσο ευέλικτος μπορεί να είναι ο παραμετρικός προγραμματισμός σε τέτοιου είδους εφαρμογές.



Εικόνα 10 όμοια τεμάχια

Τα 3 αυτά διαφορετικά κομμάτια έχουν κάποιες ομοιότητες και κάποιες διαφορές μεταξύ τους. Αναλυτικότερα:

Ομοιότητες:

-Ίδια εξωτερική-εσωτερική διάμετρο

-ίδιο πάχος

-2 αντιδιαμετρικοί κύλινδροι ευθυγραμμίσεως ιδίας διατομής και ύψους

Διαφορές:

-Ο αριθμός των οπών

-Η διάμετρος των οπών

-Η διάμετρος των κέντρων των οπών

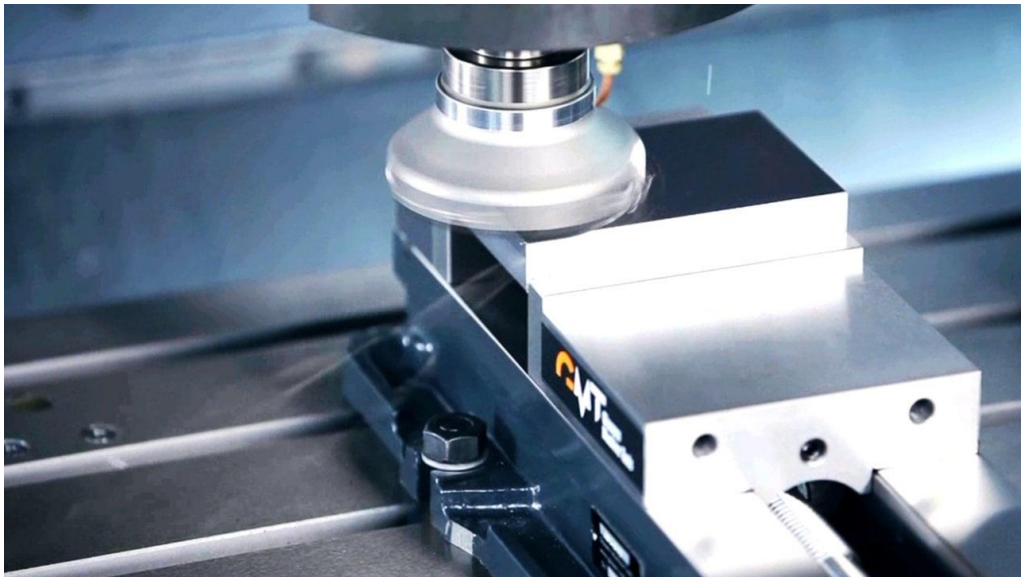
-Ο σχετικός προσανατολισμός οπών-κυλίνδρων ευθυγραμμίσεως.

Για την ανάπτυξη ενός προγράμματος που να μπορεί να κατασκευάζει και τα τρία αυτά διαφορετικά κομμάτια ακολουθείται η εξής λογική. Προγραμματίζεται με συμβατικό προγραμματισμό κάθε στοιχείο των κομματιών που είναι κοινό και στα τρία κομμάτια. Ενώ για τα σημεία που υπάρχουν διαφορές γίνεται κλήση παραμετρικού υποπρογράμματος και δίνονται μέσω των arguments οι κατάλληλες τιμές για την εκτέλεση του.

Αναλυτικό φασεολόγιο και στρατηγική κοπής για τα εν λόγο κομμάτια θα μπορούσε να είναι :

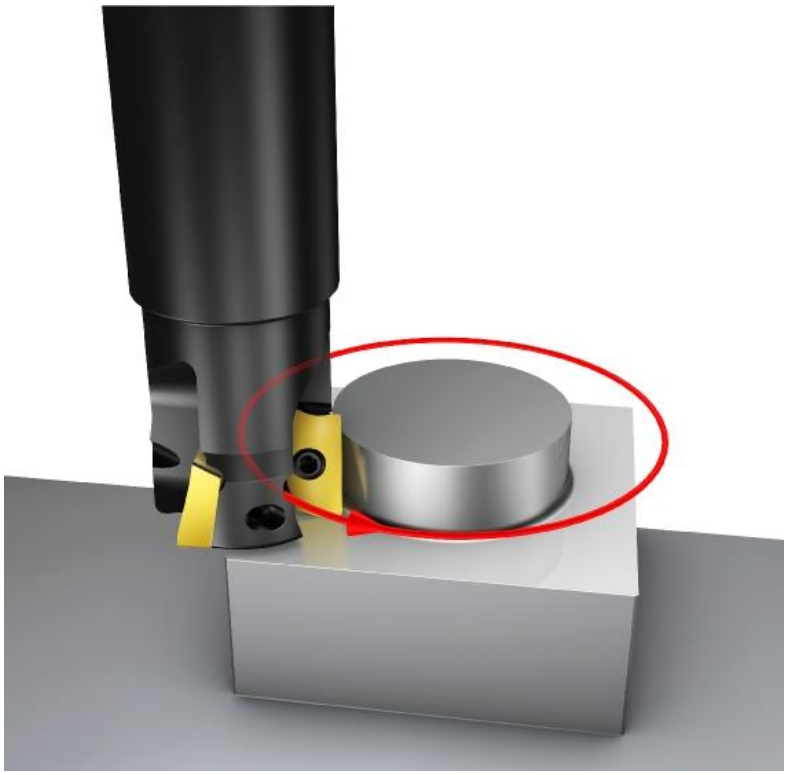
Φάση 1: ακατέργαστο υλικό

- μετωπικό φρεζάρισμα (face milling) στο πιο πάνω σημείο του τεμαχίου (εικόνα 11, n.d.)



Εικόνα 11, μετωπικό φρεζάρισμα

- περιφερειακό φρεζάρισμα για εσωτερική-εξωτερική διάμετρο (εικόνα 12, n.d.)

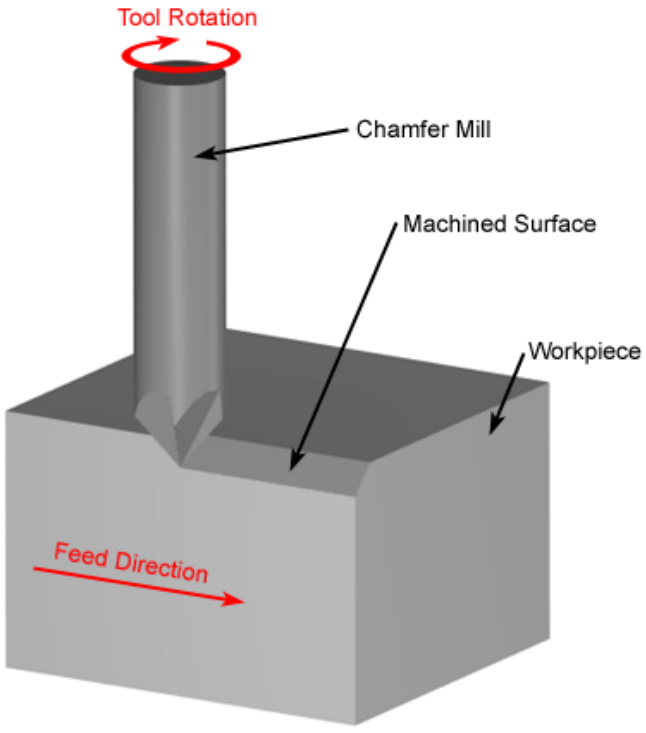


Εικόνα 12, περιφερειακό φρεζάρισμα

- κατασκευή πείρων ευθυγραμμίσεως
- διάτρηση οπών (εικόνα 13, n.d.)

Εικόνα 13,διάτρηση οπών με ταχυτρύπανο

- Λοξότμηση ακμών (εικόνα 14, n.d.)



Εικόνα 14, κατεργασία λοξότμησης ακμών

Φάση 2: ανάποδη συγκράτηση

- Μετωπικό φρεζάρισμα για το πάχος του τελικού τεμαχίου
- Λοξότμηση ακμών

Είναι προφανές μέχρι τώρα ότι στην εν λόγο εφαρμογή απαιτείται να γινεί χρήση παραμετρικού προγραμματισμού μόνο για τις οπές (θέση-πλήθος-διάμετρος) για όλα τα υπόλοιπα μπορεί και είναι ευκολότερο και αποδοτικότερο να γίνει χρήση συμβατικού προγραμματισμού. Για αυτό τον λόγο δεν θα γίνει αναλυτική αναφορά σε ολόκληρο το πρόγραμμα αλλά μόνο στο παραμετρικό κομμάτι.

Ένα επιπλέον στοιχείο που οφείλει να γνωρίζει ο προγραμματιστής κατά την ανάπτυξη του συγκεκριμένου προγράμματος είναι ότι το σημείο μηδενισμού βρίσκεται στο κέντρο και στην πάνω μεριά του τεμαχίου.

Τα μεγέθη που πρέπει να είναι μεταβαλλόμενα είναι:

- Οι συνθήκες κοπής (ώστε να χρησιμοποιείτε και για διαφορετικά υλικά)
- Η επιλογή του εργαλείου (για οπές διαφόρων διαμέτρων)
- Η ακτίνα των κέντρων των οπών με βάση το κέντρο του τεμαχίου
- Η γωνία της πρώτης οπής με βάση τον οριζόντιο άξονα
- Το τελικό βάθος διάτρησης
- Ο αριθμός των οπών
- Ο αριθμός των απαιτούμενων κομματιών

Όλα τα παραπάνω αντιστοιχούν το κάθε ένα σε μια μεταβλητή. Κάποιες από αυτές τις μεταβλητές θα παίρνουν την τιμή τους από τα arguments που εισάγει ο χειριστής κατά την πρώτη εκτέλεση του προγράμματος ενώ κάποιες άλλες θα προκύπτουν από πράξεις που θα εκτελούνται μεταξύ μεταβλητών.

Αναλυτικότερα:

(Variables / Arguments)

#19 = S_ (rpm)

#9 = F_ (ταχύτητα πρόωσης)

#26 = Z_ (βάθος διάτρησης)

#20 = T_ (εργαλείο)

#4 = I_ (ακτίνα του κέντρου των οπών από το κέντρο του κομματιού)

#5 = J_ (γωνία πρώτης οπής με βάση τον οριζόντιο άξονα)

#11 = H_ (πλήθος οπών)

%

O90999 (FLANGES FAMILY)

#106=#4001 *(G00,01,02,03)*

#107=#4002 (G17,18,19)

#108=#4003 *(G90,91)*

#109=#4014 *(G54-59)*

T#20 M06 (επιλογή εργαλείου)

S#19 M03 (στροφές περιστροφής του εργαλείου)

G00 X0 Y0 (κέντρο τεμαχίου)

G00 G43 H20 Z20. M08 (κατεβαίνει στο Z20 με αντιστάθμιση και ψυκτικό υγρό)

G81 Z#26 R2. F#9 L0 (ρουτίνα διάτρησης με L0 ώστε να μην εκτελεστεί σε αυτή την θέση)

G70 I#4 J#5 L#11 (ρουτίνα διάτρησης σε κύκλο)

G80 (ακύρωση ρουτινών)

G00 Z100. M09 (ανεβαίνει στο Z100 και σταματάει το ψυκτικό υγρό)

G#106 G#107 G#108 G#109

M99 (επιστροφή στο βασικό πρόγραμμα)

%

Κατά την κλήση του παραπάνω υποπρογράμματος και δίνοντας τις κατάλληλες τιμές στα arguments το πρόγραμμα αυτό είναι σε θέση να παράγει οποιοδήποτε από

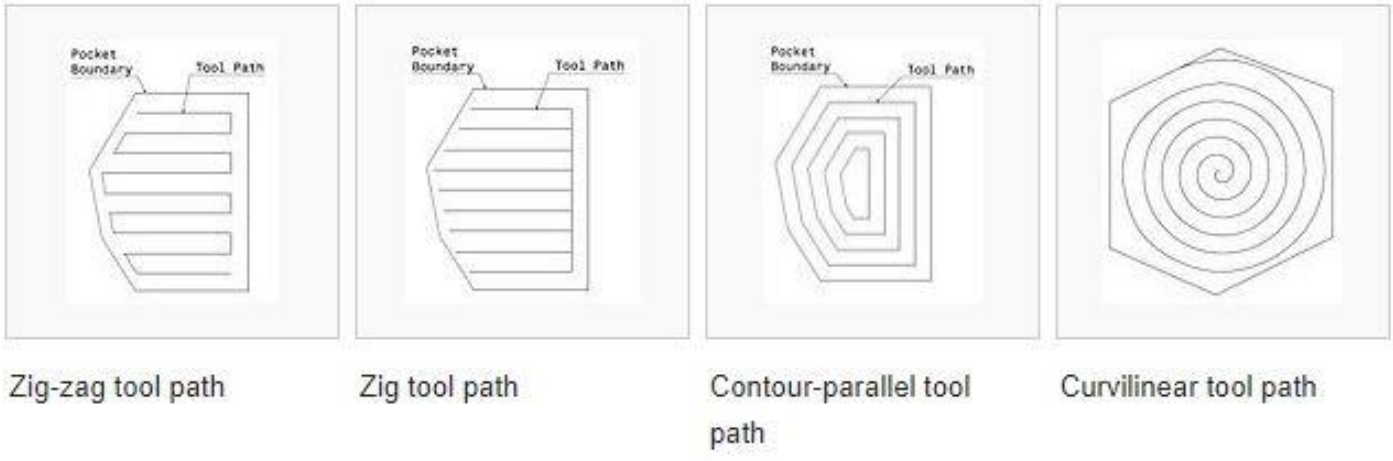
τα τρία αυτά κομμάτια αλλά και οποιοδήποτε άλλο κομμάτι έχει διαφορές μόνο ως προς το πλήθος, την διάμετρο και την θέση των κέντρων των οπών σε ότι υλικό απαιτείται. Αυτό δεν θα ήταν δυνατό με καμία άλλη μέθοδο προγραμματισμού. Αξιοσημείωτη επίσης είναι η ευκολία ανάγνωσης και διόρθωσης των μεταβλητών αφού όλα αυτά ο χειριστής τα ορίζει σε μια μόλις γραμμή του κώδικα κατά την κλήση του υποπρογράμματος.

ΚΛΗΣΗ:

*G65 P*90999 T_ S_ F_ Z_ I_ J_ H_

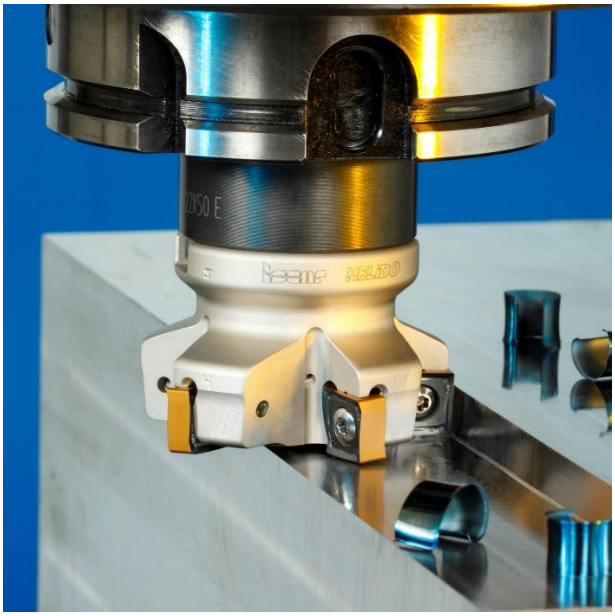
20. Ρουτίνα μετωπικού φρεζαρίσματος

Το μετωπικό φρεζάρισμα είναι μια από τις συνηθέστερες κατεργασίες και μπορεί να γίνει με πολλούς τρόπους (εικόνα 15, n.d.)

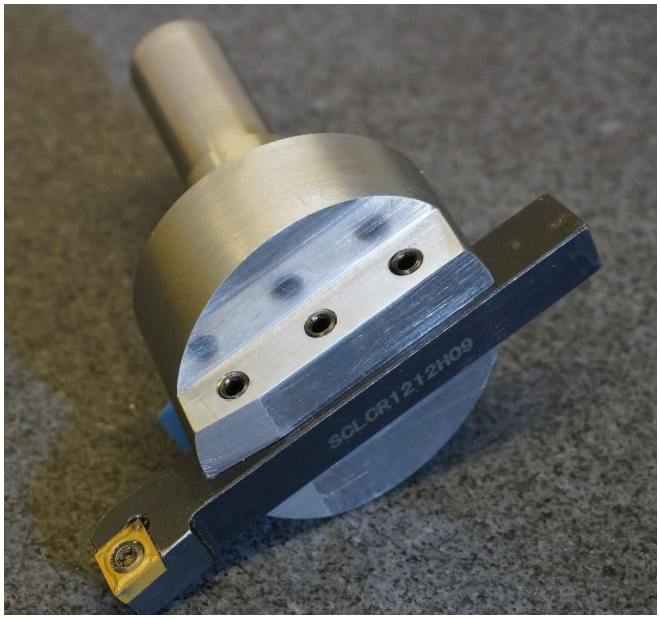


Εικόνα 15 διαφορετικά toolpaths για μετωπικό φρεζάρισμα

Καθώς επίσης και με διαφορετικά εργαλεία πχ. κονδύλια, φρεζοκεφαλές (εικόνα 16, 2015), flycutters (εικόνα 17, n.d.) κλπ.



Εικόνα 16 φρεζοκεφαλή



Εικόνα 17 flycutter

Η επιλογή του προγραμματιστή εξαρτάται από τις απαιτήσεις του τεμαχίου που κατασκευάζει. Η κάθε μέθοδος και το κάθε εργαλείο έχει τα πλεονεκτήματα και τα μειονεκτήματα του. Για παράδειγμα όταν ένα τεμάχιο έχει υψηλές απαιτήσεις σε γεωμετρικές ανοχές (επιπεδότητες – παραλληλίες κλπ.) μια καλή πρακτική είναι να προγραμματιστεί ένα toolpath που το εργαλείο θα κόβει προς το σταθερό μάγουλο της

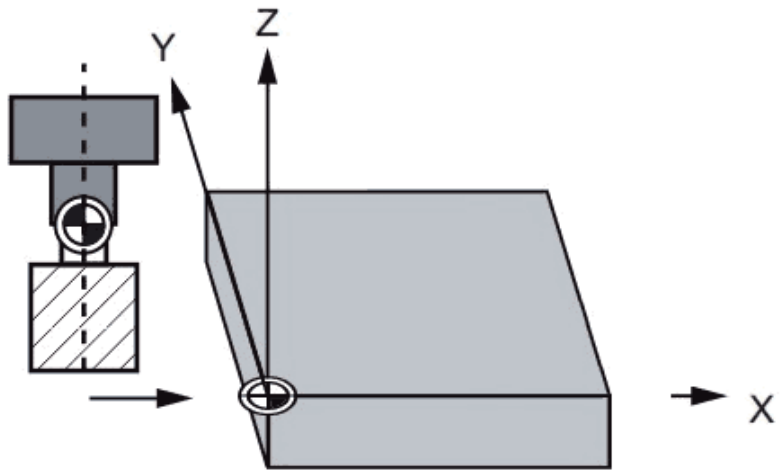
μέγγενης με ένα μικρό βάθος κοπής αξονικά (3-7% της διαμέτρου του κοπτικού) και ένα μέτριο βάθος ακτινικά (30-60% της διαμέτρου του κοπτικού) έχοντας υπόψιν ότι το υλικό που πρέπει να αφαιρεθεί θα μοιραστεί αν είναι δυνατόν στις δυο πλευρές του τεμαχίου που γίνεται το μετωπικό φρεζάρισμα.

Η κατασκευή μιας εξατομικευμένης ρουτίνας για το παραπάνω παράδειγμα προσφέρει στον προγραμματιστή την δυνατότητα να προγραμματίσει την κατεργασία μέσα σε ελάχιστο χρόνο πολύ πιο ευκολά από κάθε άλλη μέθοδο προγραμματισμού.

Τα δεδομένα που μεταβάλλονται σε μια τέτοια κατεργασία είναι τα εξής:

- Το εργαλείο
- Το ακτινικό βάθος κοπής (stepover)
- Το αξονικό βάθος κοπής
- Οι διαστάσεις του τεμαχίου

Όπως σε κάθε ρουτίνα είναι αναγκαίο να γίνουν κάποιες παραδοχές για να γίνεται πάντα σωστή χρήση της ρουτίνας από τον χειριστή και να αποφευχθούν τυχόν λάθη. Στην προκειμένη περίπτωση η μόνη παραδοχή που θα γίνει είναι ότι το σημείο μηδενισμού θα βρίσκεται στο κάτω αριστερό και ψηλότερο σημείο του τεμαχίου. (εικόνα 18, n.d.)



Εικόνα 18, σημείο μηδενισμού

%

O90010 (FACEMILL MACRO)

(Variables / Arguments)

#23= W_ (ΔΙΑΣΤΑΣΗ ΤΕΜ. ΚΑΤΑ Χ)

#11= H_ (Διάσταση τεμαχίου κατά Υ)

#26= Z_ (Βάθος κοπής)

#19= S_ (stepover %)

#18= R_ (Retract)

#9= F_ (Feedrate)

#105=#[2000+#4120] *(ακτίνα εργαλείου)*

#106=#4001 *(G00,01,02,03,33)*

#107=#4002 (G17,18,19)

#108=#4003 *(G90,91)*

#109=#4014 *(G54-59)*

#119= [#19/100] *[2*#105]] *(stepover σε χιλιοστά)*

G00 G90 X [-#105+#119] Y [-#105-5.] (X=stepover, Y=-5)

G00 Z[#18] (retract)

#120= [-#105+#119] (#120= μέχρι ποιο Χ έχει φρεζαριστεί η επιφάνεια)

WHILE [#120 LT #23] DO1 (όσο η επιφάνεια που έχει φρεζαριστεί είναι μικρότερη από το πλάτος του τεμαχίου…)

G00 X[#120+#119] Y [-#105-5]] (+ άλλο ένα stepover, Y αρχικό)

G00 Z#26 (βάθος κοπής)

G01 Y[#11+#105+5] F#9 (φρεζάρισμα με πρόωση μέχρι να βγει όλο το εργαλείο έξω από το τεμάχιο)

G00 Z[#18] (Retract)

#120=#120+#119 (προσαύξηση της #120 κατά ένα stepover)

END1 (όταν φρεζαριστεί όλο το πλάτος…)

```
G0 Z100. M09                    (απόσταση ασφαλείας, κλείσιμο ψυκτικού)

G#106 G#107 G#108 G#109

M99                             (επιστροφή στο βασικό πρόγραμμα)
```

Έχοντας δώσει το όνομα O90010 στο παραπάνω macro υπάρχει η δυνατότητα αντιστοίχισης του με την παράμετρο #6050. Αυτό σημαίνει ότι με την επιθυμητή τιμή στην παράμετρο αυτή, η κλήση του συγκεκριμένου Macro εκτός από τον κλασσικό τρόπο (G65/G66) θα μπορεί πλέον να γίνει και καλώντας την εντολή G με τον αριθμό της τιμής που δώσαμε στην παράμετρο #6050. Αναλυτικότερα:

1. Ορίζεται η τιμή 111 στην παράμετρο #6050
2. Κλήση macro , G111 W_ H_ Z_ S_ R_ F_

Όπου:

W= μήκος τεμαχίου κατά X

H= μήκος τεμαχίου κατά Y

Z= βάθος φρεζαρίσματος

S= rpm

R= επίπεδο ασφαλείας (Retract)

F= ταχύτητα πρόωσης (Feedrate)

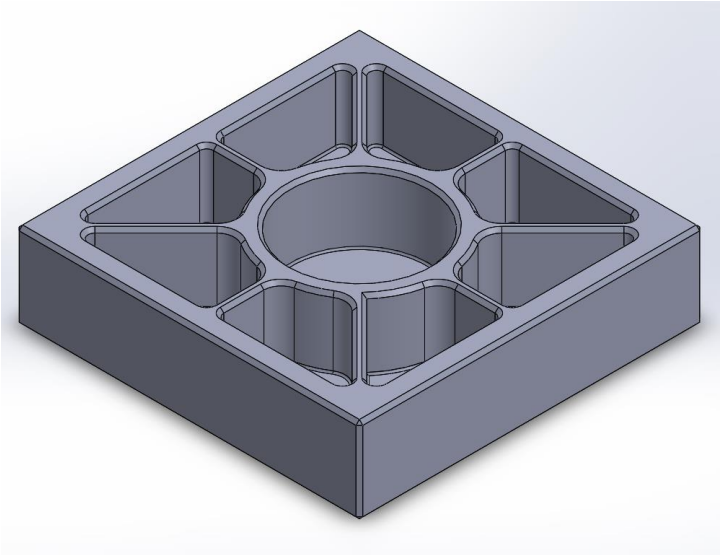
Ορίζοντας τις παραπάνω παραμέτρους (arguments) κατά την κλήση της εντολής G111 θα εκτελείται το παραπάνω παραμετρικό πρόγραμμα για μετωπικό φρεζάρισμα σε κομμάτια ανεξαρτήτου διαστάσεων. Αυτή είναι μία ακόμα περίπτωση που ο παραμετρικός προγραμματισμός υπερέχει έναντι οποιασδήποτε άλλης μεθόδου. Επιπλέον είναι φανερό ότι η αποδοτικότητα και η παραγωγικότητα του προγραμματιστή αλλά και της εργαλειομηχανής αυξάνονται κατακόρυφα αφού για τον προγραμματισμό μιας τέτοιας κατεργασίας δεν απαιτείται πλέον περισσότερο από μια γραμμή κώδικα.

21. Macro & δυναμικός μετρητικός έλεγχος

Η ανάγκη εξάλειψης απορριφθέντων τεμαχίων κατά την εκτέλεση μεγάλων παραγωγών ήταν πάντα ένα από τα μεγαλύτερα προβλήματα, ειδικά σε τεμάχια που απαιτούν γεωμετρικές ανοχές (ομοκεντρίες, καθετότητες κλπ.) που δεν μπορούν να μετρηθούν με όργανα χειρός (παχύμετρα, μικρόμετρα, ελεγκτήρες κλπ.) αλλά μόνο με μετρητικές διατάξεις (πχ. CMM).

Κάνοντας χρήση του παραμετρικού προγραμματισμού και αξιοποιώντας συσκευές για probing τις περισσότερες φορές ο προγραμματιστής καταφέρνει να δώσει λύση σε τέτοιου είδους προβλήματα μειώνοντας έτσι το κόστος της παραγωγής. Αυτό μπορεί να επιτευχθεί μειώνοντας τον αριθμό των τεμαχίων που απορρίπτονται αλλά και τον ανθρώπινο παράγοντα σε διαδικασίες μέτρησης και διόρθωσης των offset.

Σε παραγωγές που απαιτείται μεγάλος αριθμός τεμαχίων η διαδικασία που εκτελεί ο χειριστής από ένα σημείο και μετά εκτελείται μηχανικά και αυτό πολλές φορές κρύβει κινδύνους. Η ρουτίνα και ο χρόνος είναι ο χειρότερος εχθρός του χειριστή σε τέτοιες περιπτώσεις διότι από ένα εκτελεί την διαδικασία μηχανικά (έλλειψη συγκέντρωσης) για να να επιτύχει γρηγορότερες αλλαγές τεμαχίων ώστε να είναι πιο παραγωγικός και αποδοτικός (αυξημένη πίεση εργασίας). Στο παρακάτω παράδειγμα παρουσιάζεται αναλυτικά η ανάπτυξη ενός παραμετρικού προγράμματος που χρησιμοποιείται για τον έλεγχο του κρίσιμου σημείου ενός τεμαχίου.



Εικόνα 19,τεμάχιο με απαιτήσεις διαστατικής ακρίβειας

Το εικονιζόμενο τεμάχιο (εικόνα 19, n.d.) απαιτεί διαστατική ακρίβεια στην διάμετρο της κεντρικής ποκέτας και ο αριθμός τεμαχίων που θα παραχθεί είναι 25.000 τεμάχια. Η ανοχή που απαιτείται είναι -0.01 mm που σημαίνει ότι ένα παχύμετρο δεν θα ήταν καλή επιλογή οργάνου για μια τέτοια μέτρηση. Φυσικά θα μπορούσε να γίνει με μικρόμετρο οπής που είναι καταλληλότερο όργανο σε τέτοιες περιπτώσεις αλλά λόγο μεγάλου αριθμού τεμαχίων θα ήταν πολύ καλύτερο η μέτρηση αυτή να γινεί με ένα probe και αν είναι δυνατή η διόρθωση του τεμαχίου να αυτοματοποιηθεί ώστε ο χειριστής να έχει περισσότερο χρόνο για άλλες μετρήσεις λιγότερο κρίσιμες, απογρέζωση ή ακόμα και ταυτόχρονη εκτέλεση και άλλων παραγωγών παράλληλα. Όλα αυτά σημαίνουν κατακόρυφη αύξηση των κερδών για την εταιρεία και ταυτόχρονη μείωση των ευθυνών και πιο ξεκούραστη εργασία για τον χειριστή.

Το σημείο λοιπόν που αξίζει να γίνει χρήση παραμετρικού προγραμματισμού σε αυτή την περίπτωση είναι η κατασκευή της κεντρικής ποκέτας καθώς όλα τα υπόλοιπα στοιχεία μπορούν ευκολότερα να προγραμματιστούν με οποιονδήποτε άλλο τρόπο προγραμματισμού.

Η στρατηγική που ακολουθείται είναι η εξής:

- Προδιάτρηση οπής στο κέντρο της ποκέτας
- Διάνοιξη ποκέτας με κονδύλι (ξεχόνδρισμα)
- Φινίρισμα
- Έλεγχος βάθους
- Λήψη απόφασης για διόρθωση/απόρριψη/συνέχιση εκτέλεσης
- Έλεγχος διαμέτρου
- Λήψη απόφασης για διόρθωση/απόρριψη/τέλος προγράμματος

Οι παραδοχές που πρέπει να ληφθούν υπόψιν είναι οι εξής:

- Σημείο μηδενισμού στο κέντρο της ποκέτας και πάνω
- Η επιθυμητή διάμετρος είναι το κέντρο της ανοχής(40-(0.01/2)=39.995mm)
- Το υλικό που θα μείνει για τελικό φινίρισμα είναι 0,2 mm στην ακτίνα.

```
%
O00099 (macro & probing)
T01 M06 (centerdrill 10)
G54 G90
S3000 M03
G00 X0 Y0
G00 G43 H01 Z20. M08
G81 Z-4.5 R2. F80.
G80
G00 Z20. M09
;
T02 M06 (Drill 12)
G54 G90
S1100 M03
G00 X0 Y0
G00 G43 H02 Z20. M08
G83 Z-19.95 R2. F150. Q2.5
G80
G00 Z20. M09
;  (Roughing)
T03 M06 (CARBIDE ENDMILL 10 2FLUTE)
G54 G90
S4000 M03
G00 X0 Y0
G00 G43 H03 Z20. M08
```

G00 Z-15.

G01 Z-20 F100. (μειωμένο Feedrate εξαιτίας του υλικού που έχει μείνει από την μύτη του T2)

G13 I7. K19.797 Q1. D03 F400. (διάνοιξη pocket στην επιθυμητή διάμετρο -0.2 mm)

G00 Z20. M09

G00 Z100.

; (Finishing)

N5

T04 M06 (CARBIDE ENDMILL 10 3FLUTE)

G54 G90

S5000 M03

G00 X0 Y0

G00 G43 H04 Z20. M08

G00 Z-20.

G13 I19.997 D04 F350. (διάνοιξη pocket με διάμετρο ακριβώς το κέντρο της ανοχής)

G00 Z20. M09

G00 Z100.

; (Probing)

(VARIABLES)

#1=20. (θεωρητικό βάθος ποκέτας)

#11 (πραγματικό βάθος ποκέτας)

#2=#11-#1 (υπολογισμός διόρθωσης για το βάθος (διαφορά πραγματικού - θεωρητικού βάθους ποκέτας))

#4 (1ο σημείο διαμέτρου)

#5 (2ο σημείο διαμέτρου)

#16=#5-#4 (πραγματική διάμετρος ποκέτας)

#6=40. (θεωρητική διάμετρος ποκέτας)

#7=[#6-#16]/2 (υπολογισμός διόρθωσης για την διάμετρο (ημιδιαφορά θεωρητικής-πραγματικής διαμέτρου))

;

T25 M06 (PROBE)

G54 G90

G00 X10. Y-10. (εκτός κέντρου που έχει γίνει η προδιάτρηση για να εξασφαλισθεί οτι δεν έχει σπάσει το κονδύλι.)

G00 G43 H25 Z20.

G91

G31 Z-50. F100.

G00 Z5.

G31 Z-10 F50.

#11=#5063 (αποθηκεύει στην μεταβλητή #11 την τιμή του τρέχον Z

IF [#11 GE 0] GOTO N10 (αν το πραγματικό βάθος βγαίνει θετικό βγάλε alarm)

IF [#11 EQ #1] GOTO N15 (αν το πραγματικό βάθος ισούται με το θεωρητικό συνέχισε στον έλεγχο της διαμέτρου, αλλιώς…)

G10 G91 L10 P04 R#2 (διόρθωσε το tool offset height geometry για το T04 κατά #2)

GOTO N5

N15 (έλεγχος διαμέτρου)

G90 G00 Z-10

G00 X0 Y0

G31 G91 X-25. F100.

G00 X5.

G31 X-10 F50.

#4=#5061 (1ο σημείο διαμέτρου)

G31 X45. F100.

G00 X-5.

G31 X10. F50.

#5=#5061 (2ο σημείο διαμέτρου)

IF [#16 GT #6] GOTO N20 (αν η πραγματική διάμετρος είναι μεγάλη βγάλε alarm)

IF [#16 EQ #6] GOTO N25 (αν είναι ίση με την θεωρητική προχώρησε σε ασφαλή τερματισμό αλλιώς…)

G10 G91 L12 P04 R#7 (διόρθωση tool offset radius geometry για το T04 κατά #7)

GOTO N5 (επανεκτέλεση φινιρίσματος)

N25 (ασφαλής τερματισμός)

G90

G00 Z100.

G53 Y0.

M30

(ALARMS)

N10 #3000=100 (T04 IS BROKEN)

N20 #3000=101 (PART REJECTED)

(TROUBLESHOOTING)

(ERROR 100: REPLACE T04)

(ERROR 101: DIAMETER OFF TOLERANCE)

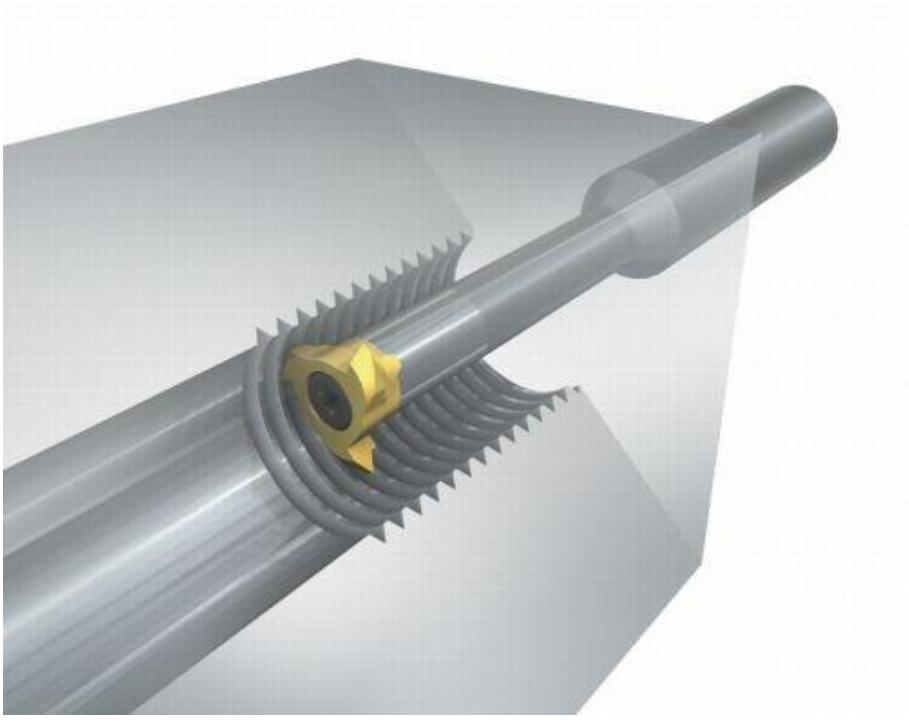
Συμπερασματικά είναι σαφές ότι αυτό που χαρακτηρίζει την αξία ενός προγράμματος αλλά και του ίδιου του προγραμματιστή είναι το κατά πόσο γρήγορα, ποιοτικά και κερδοφόρα παράγονται τα τεμάχια. Σίγουρα ο προγραμματισμός του μετρητικού ελέγχου είναι εφικτός και με τον συμβατικό προγραμματισμό αλλά η δυναμική διόρθωση των offset δεν είναι δυνατή χωρίς την χρήση παραμετρικού προγραμματισμού. Αυτό επειδή είναι η μόνη μέθοδος η οποία μπορεί να κάνει χρήση μεταβλητών αλλά και σταθερών παραμέτρων, να τις αξιοποιήσει κάνοντας πράξεις (μαθηματικές & λογικές) και να καταλήξει σε συμπεράσματα ώστε να ληφθεί μια απόφαση που θα εκτελεστεί χωρίς την ανάγκη του ανθρώπου.

22. Ρουτίνα Κωνικής εσωτερικής σπειρωτόμησης

Σχεδόν όλοι οι ελεγκτές των σύγχρονων εργαλειομηχανών αριθμητικού ελέγχου διαθέτουν ρουτίνες για τον προγραμματισμό διαφόρων κατεργασιών. Αυτή η ανάγκη δημιουργήθηκε από τα πρώτα κιόλας χρόνια της ιστορίας των CNC για δυο βασικούς λόγους. Για την αύξηση της παραγωγικότητας και τον ευκολότερο προγραμματισμό επαναλαμβανόμενων κατεργασιών. Φυσικά οι απαιτήσεις διαφέρουν για τον κάθε προγραμματιστή γι' αυτό το λόγο οι κατασκευαστές τον ελεγκτών έχουν κατασκευάσει ρουτίνες μόνο για τα πιο συνηθισμένα είδη κατεργασιών (σπειρωτόμηση, διάτρηση, pockets κλπ.) που χρησιμοποιούνται ευρέως απ' όλους δίνοντας όμως την δυνατότητα στον προγραμματιστή να μπορεί να κατασκευάσει τις δικές του ρουτίνες

Παρακάτω παρουσιάζεται αναλυτικά η ανάπτυξη μιας ρουτίνας για εσωτερική σπειρωτόμηση (internal thread milling) με δυνατότητα κατασκευής ακόμα και κωνικού σπειρώματος. Ο προγραμματισμός για ένα ελικοειδές toolpath με μεταβαλλόμενη ακτίνα δεν είναι ιδιαίτερα εύκολος και αν αυτό είναι συχνή απαίτηση για τον προγραμματιστή τότε σίγουρα συμφέρει η κατασκευή μιας τέτοιας ρουτίνας.

Τέτοιου είδους σπειρώματα χρησιμοποιούνται σε εφαρμογές που απαιτείτε στεγάνωση (δοχεία καυσίμου, λαδιού κλπ.) και αυτό σημαίνει ότι απαιτείται απόλυτη ακρίβεια στην κατασκευή. Για αυτό τον λόγο η στρατηγική που ακολουθείται συνήθως είναι η κατασκευή του σπειρώματος με φρέζα σπειρωτόμησης (εικόνα 20, n.d.) με ομόρροπο φρεζάρισμα.



Εικόνα 20, σπειρωτόμηση με φρέζα σπειρωμάτων

Τα μεγέθη που πρέπει να παραμετροποιήσουμε σε μια τέτοια περίπτωση είναι τα εξής:

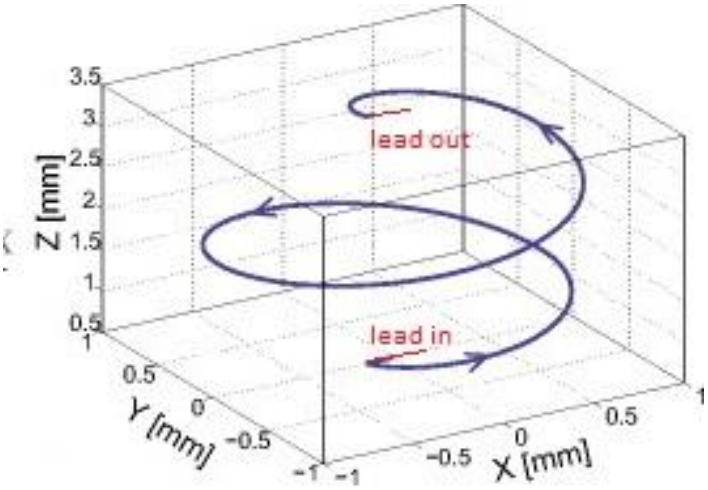
- Το τελικό βάθος του σπειρώματος
- Η διάμετρος του πυρήνα
- Το βήμα
- Η γωνία της κωνικότητας

Η στρατηγική κοπής που ακολουθείται για βέλτιστα αποτελέσματα είναι:

Τελικό βάθος > lead in > σπειρωτόμηση > lead out

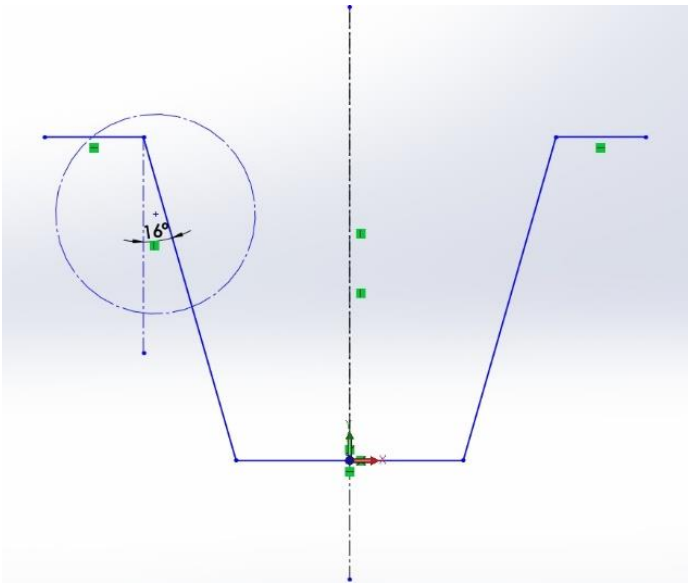
Επιλέγεται η κοπή από κάτω προς τα πάνω για να είναι ομόρροπο το φρεζάρισμα και επειδή μια από τις δυσκολίες σε κατεργασίες σπειρωτόμησης είναι η απομάκρυνση του γρεζιού. Με αυτό τον τρόπο το γρέζι απομακρύνεται μόνο του από το επίπεδο κοπής ακόμα και χωρίς την χρήση ψυκτικού υγρού αν το επιτρέπει το υλικό.

Το lead in/out (εικόνα 21, n.d.) είναι η κίνηση που γίνεται για να πάει/φύγει το κοπτικό στην θέση κοπής και αξιοποιείται για να ενεργοποιήσουμε/απενεργοποιήσουμε την αντιστάθμιση της ακτίνας του κοπτικού εργαλείου.



Εικόνα 21 ελικοειδές toolpath με lead in/out

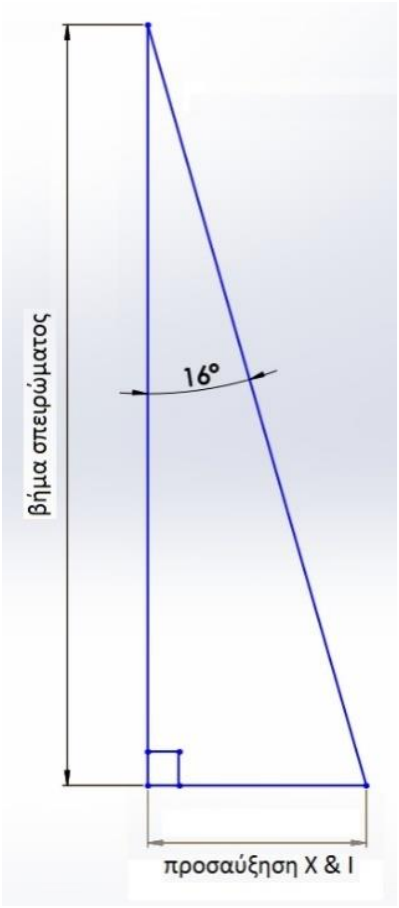
Η δυσκολία στον προγραμματισμό του συγκεκριμένου toolpath είναι στην μεταβαλλόμενη ακτίνα. Ο υπολογισμός της μεταβολής αυτής θα πρέπει να γίνεται δυναμικά με βάση την γωνίας της κωνικότητας. Πρακτικά αυτό που μεταβάλλεται στον κώδικα είναι οι παράμετροι X & I στην εντολή G03 κατά την ελικοειδή τροχιά. Η παράμετρος I προσαυξάνεται ακριβώς όσο προσαυξάνεται και η παράμετρος X. Αναλυτικότερα για τον υπολογισμό της προσαύξησης :



Εικόνα 22 τομή κωνικού σπειρώματος 16 μοιρών

Τριγωνομετρικά (εικόνα 22 & 23, n.d.) προκύπτει ότι:

Προσαύξηση X & I = βήμα * εφαπτομένη της γωνίας



Εικόνα 23 λεπτομέρεια τομής κωνικού σπειρώματος

```
%
O90011 (INTERNAL TAPERED THREAD)

(VARIABLES & ARGUMENTS)
#18=   R (RETRACT)
#17=   Q (PITCH)
#26=   Z (ΤΕΛΙΚΟ ΒΑΘΟΣ)
#22=  V (MIN DIAMETER)
#1=    A (ANGLE)
#9=   F (FEEDRATE)
#7=   D (TOOL COMPENSATION)
#10=[[#17*[TAN#1]] (προσαύξηση X)
#11=[[#22/2]]+#10] (προσαύξηση I + ακτίνα)
```

#12=**FUP**[#26/#17] (LOOPES)

G00 Z[#18] (RETRACT)

G00 Z[#26] (MAXIMUM DEPTH)

G91 G03 G41 D[#7] X[#22/2] R[#22/2] F[#9] (LEAD IN)

G03 X[#10] I-[#11] Z#17 L[#12] (THREADING)

G00 G90 G40 X0. Y0. (LEAD OUT)

G00 Z[#18] M09 (RETRACT)

M99 (RETURN)

Είναι εντυπωσιακό το πόσο μικρός είναι ο κώδικας που παράγεται στον παραμετρικό προγραμματισμό καθώς αυτό κάνει την ανάγνωση και την διόρθωση πολύ πιο εύκολη. Η ίδια κατεργασία με οποιαδήποτε άλλη μέθοδο προγραμματισμού θα είχε πολλές περισσότερες γραμμές και επιπλέον θα έπρεπε κάθε φορά που αλλάζει κάτι να γίνεται επαναπρογραμματισμός γιατί καμία άλλη μέθοδος δεν μπορεί να δημιουργήσει δυναμικά προγράμματα ή να κατασκευάσει εξατομικευμένες ρουτίνες.

Για να αποθηκευτεί η παραπάνω ρουτίνα στον ελεγκτή της μηχανής ακολουθείται η εξής διαδικασία:

- Ορίζουμε μια τιμή (έστω 112) στην παράμετρο #6051 (πρόγραμμα O90011)
- Κλήση ρουτίνας ορίζοντας τις επιθυμητές τιμές στα arguments

Παράδειγμα για την κατασκευή κωνικού σπειρώματος γωνίας 16 μοιρών, σε τελικό βάθος 35 mm, με διάμετρο πυρήνα 60mm και βήμα 0.5 mm αρκεί μόνο μια γραμμή κώδικα:

G112 A16. Z-35. V60. Q0.5 F250. R2. D05

23. Τελικά συμπεράσματα

Τα τελευταία χρόνια η κατασκευαστές των εργαλειομηχανών αριθμητικού ελέγχου έχουν καταφέρει να κατασκευάσουν ελεγκτές με πολλά περισσότερα χαρακτηριστικά από όσα θα χρειαζόταν ένας μέσος προγραμματιστής. Αυτό γίνεται για να παρέχει την δυνατότητα στον προγραμματιστή να προγραμματίσει απευθείας στον ελεγκτή χωρίς την ανάγκη κάποιου εξωτερικού λογισμικού που κοστίζει αρκετά χρήματα και πολλές φορές δεν είναι καν απαραίτητο. Δεν είναι λίγες οι φορές που η λύση του συμβατικού η του διαλογικού προγραμματισμού είναι η πιο γρήγορη και η πιο συμφέρουσα. Φυσικά ο προγραμματισμός με λογισμικά CAM συχνά είναι μονόδρομος και το μεγάλο πλεονέκτημα του offline προγραμματισμού καθώς και οι πολύ καλές προσομοιώσεις που διαθέτουν τέτοια λογισμικά είναι πολύ σοβαρά πλεονεκτήματα και αυτοί είναι και οι κυριότεροι λόγοι που τα λογισμικά CAM έχουν γίνει τόσο δημοφιλή.

Ο σκοπός της παρούσας εργασίας δεν είναι να συγκρίνει τις διαφορετικές μεθόδους αλλά να παρουσιάσει τα οφέλη του παραμετρικού προγραμματισμού που στην πραγματικότητα είναι κομμάτι του συμβατικού προγραμματισμού. Είναι μια επέκταση του συμβατικού προγραμματισμού που δημιουργήθηκε για να πολλαπλασιάσουμε τις δυνατότητες του.

Όλες οι μεθόδοι προγραμματισμού έχουν πλεονεκτήματα και μειονεκτήματα για αυτό τον λόγο όπως παρουσιάστηκε και στην παρούσα εργασία ο προγραμματιστής που αναπτύσσει το πρόγραμμα είναι υπεύθυνος να επιλέξει την ή τις κατάλληλες μεθόδους ώστε να υλοποιήσει τον στόχο του. Πρέπει να είναι σε θέση να γνωρίζει όλες τις μεθόδους και να μπορεί να αξιολογήσει ποια μέθοδος απαιτείται σε κάθε σημείο του προγράμματος που θέλει να φτιάξει. Η βέλτιστη χρήση που μπορεί να κάνει ένας προγραμματιστής είναι να αξιοποιεί όλες τις μεθόδους ώστε η μια να συμπληρώνει την άλλη.

24. Βιβλιογραφία

1. Sinha, S. K., 2005. *Cnc programming using Fanuc custom macro B.* s.l.:Mc Graw Hill.
2. Smid, P., 2005. *Fanuc Custom Macros.* s.l.:Industrial Press.
3. εικόνα 10, n.d. *Family parts example.* s.l.:s.n.
4. εικόνα 11, n.d. *Glacern Vices.* [Ηλεκτρονικό]
   Available at: https://i.vimeocdn.com/video/36906656.jpg?mw=900&mh=506
5. εικόνα 1, 2017. *Siemens Sinumerik.* [Ηλεκτρονικό]
   Available at:
   https://www.industry.siemens.com/topics/global/en/cnc4you/tips_and_tricks/PublishingImages/maximum-productivity/mehrkanalige-simulation_en.jpg
6. εικόνα 12, n.d. *Sandvik Coromant.* [Ηλεκτρονικό]
   Available at:
   https://www.sandvik.coromant.com/SiteCollectionImages/Technical%20guide/Pablo/D%20milling/090198.jpg
7. εικόνα 13, n.d. *Sandvik Coromant.* [Ηλεκτρονικό]
   Available at: https://i.ytimg.com/vi/dNPDTmmpcgE/maxresdefault.jpg
8. εικόνα 14, n.d. *Custom parts.* [Ηλεκτρονικό]
   Available at: http://www.custompartnet.com/wu/images/milling/chamfer-milling.png
9. εικόνα 15, n.d. *ResearchGate.* [Ηλεκτρονικό]
   Available at: https://www.researchgate.net/figure/Face-milling-a-square-face-with-cutting-primarily-in-a-X-axis-direction-b-Y-axis_fig4_314609464
10. εικόνα 16, 2015. *Iscar.* [Ηλεκτρονικό]
    Available at: http://img.directindustry.com/images_di/photo-g/5692-3297481.jpg
11. εικόνα 17, n.d. *Pinterest.* [Ηλεκτρονικό]
    Available at:
    https://i.pinimg.com/originals/cd/f8/c1/cdf8c1d3773b1aa9286a28c5cec7727b.jpg
12. εικόνα 17, n.d. *solidworks.* s.l.:s.n.
13. εικόνα 18, n.d. *Helman CNC.* [Ηλεκτρονικό]
    Available at: http://www.helmancnc.com/wp-content/uploads/2013/07/Work-Offset-Setting-on-Hermle-UWF-851-CNC-Mill-with-Sinumerik-Control-x-axis.gif
14. εικόνα 19, n.d. *Macro & Probing.* s.l.:s.n.
15. εικόνα 1, n.d. *Open Mind.* [Ηλεκτρονικό]
    Available at:
    https://d2.alternativeto.net/dist/s/mastercam_207474_full.jpg?format=jpg&width=1600&height=1600&mode=min&upscale=false

16. εικόνα 20, n.d. *Sandvik coromant.* [Ηλεκτρονικό]
Available at:
https://d2n4wb9orp1vta.cloudfront.net/cms/MMT_1215_Sandvik_2.jpg;width=560

17. εικόνα 21, n.d. *American society of mechanical engineers (ASME).* [Ηλεκτρονικό]
Available at:
http://manufacturingscience.asmedigitalcollection.asme.org/data/journals/jmsefk/28313/003002mae4.jpeg

18. εικόνα 22 & 23, n.d. *solidworks.* s.l.:s.n.

19. εικόνα 2, n.d. *Renishaw.* [Ηλεκτρονικό]
Available at: https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcQt7_ldKm5d6H2HFWG7XY4OVxhv-8t95oZAROq3Z-bdGlYXkq5tQw

20. εικόνα 3, n.d. *Renishaw.* [Ηλεκτρονικό]
Available at: http://img.directindustry.fr/images_di/photo-g/5200-9867393.jpg

21. εικόνα 4, n.d. *Renishaw.* [Ηλεκτρονικό]
Available at: http://www.renishawprobe.com/sc_images/products/751_large_image.jpg

22. εικόνα 5, n.d. *Fanuc.* [Ηλεκτρονικό]
Available at: http://www.cncmanuals.com/assets/fanuc/63322en/files/63322en%20-%200240_files/63322en%20-%200240-1.png

23. εικόνα 6, n.d. *sts industrials.* [Ηλεκτρονικό]
Available at: http://stsindustrial.com/wp-content/uploads/2013/11/G5-Dimensions-198.png