Ατομική Διπλωματική Εργασία

# ΕΦΑΡΜΟΓΕΣ ΡΟΜΠΟΤΙΚΟΥ ΒΡΑΧΙΟΝΑ

**Παναγιώτης Κυριάκου**

**ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ**

**ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ**

**Μάιος 2021**

# ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

## ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ


**Εφαρμογές Ρομποτικού Βραχίονα**


**Παναγιώτης Κυριάκου**


Επιβλέπων Καθηγητής
Μάριος Δικαιάκος

Η Ατομική Διπλωματική Εργασία υποβλήθηκε προς μερική εκπλήρωση των απαιτήσεων απόκτησης του πτυχίου Πληροφορικής του Τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου

Μάιος 2021

# Ευχαριστίες

Ευχαριστώ τον καθηγητή που με επέβλεπε καθ' όλη τη διάρκεια της διπλωματικής και με βοήθησε με όλα τα προβλήματα που παρουσιάστηκαν, τον κύριο Μάριο Δικαιάκο.

Ευχαριστίες θέλω να δώσω και στο κέντρο επιχειρηματικότητας του Πανεπιστημίου Κύπρου για τα εργαλεία και τον χώρο που μου διέθεσαν.

Επίσης πολλές ευχαριστίες θέλω να δώσω στον συμφοιτητή και φίλο μου Ανδρέα Κομή με τον οποίο συνεργαστήκαμε στα πλαίσια των εργασιών μας.

# Περίληψη

Σκοπός της διπλωματικής εργασίας είναι η δημιουργία μιας προσαρμοστικής και αποτελεσματικής εφαρμογής για τους ρομποτικούς βραχίονες Dobot Magician. Η αυτοματοποίηση και η λειτουργία εξ αποστάσεως, όλο και περισσότερο μπαίνουν στην ζωή μας, ειδικά στις μέρες μας που επικρατεί πανδημία. Στα πλαίσια της συγκεκριμένης διπλωματικής έχει αναπτυχθεί ένα πρόγραμμα με διάφορες εφαρμογές για την επίτευξη σεναρίων χρήσης ενός ρομποτικού βραχίονα, όπως η μεταφορά αντικειμένων και ο διαχωρισμός τους. Το πρόγραμμα έχει ενθυλακωθεί μέσα σε ένα docker container το οποίο λειτουργεί πάνω σε windows 10. Ο κύριος λόγος που προσθέσαμε το πρόγραμμα σε docker container ήταν για να μπορεί κάποιος να τρέξει και να επεξεργαστεί το πρόγραμμα από απόσταση. Για τον σκοπό αυτό χρησιμοποιήθηκαν δύο ρομποτικοί βραχίονες Dobot Magician, δύο Conveyor Belts και δύο κάμερες JeVois Smart Vision. Η ανάπτυξη του προγράμματος για τον Dobot Magician έχει γίνει σε γλώσσα προγραμματισμού Python, η δημιουργία και ανάπτυξη του docker container έγινε στα Windows 10 και τέλος, τα machine learning modules που τρέχουν στην JeVois Camera είναι υλοποιημένα σε γλώσσα C++.

Αρχικά, είχαν αναπτυχθεί δύο προγράμματα. Στο πρώτο το Dobot μετακινούσε αντικείμενα πάνω στο conveyor belt και στο δεύτερο το Dobot κατέβαζε τα αντικείμενα από το conveyor belt σε ένα άλλο σημείο. Παρατηρήθηκαν αρκετά προβλήματα συγχρονισμού και λειτουργικότητας στα δύο αρχικά προγράμματα, επίσης αφού δουλεύαμε με δύο ρομποτικούς βραχίονες δημιουργήθηκε η ανάγκη να δημιουργήσουμε νέες μεθόδους στο API του Dobot Magician, οι οποίες θα μας επέτρεπαν να συνδέσουμε τους δύο βραχίονες και να τους συγχρονίζουμε. Έτσι, στη συνέχεια βελτιώσαμε τα προγράμματα και αντιμετωπίσαμε κάποια από τα αρχικά μας προβλήματα. Οι επόμενες εκδοχές των προγραμμάτων επέτρεπαν στους δύο βραχίονες να επικοινωνούν άμεσα και να συγχρονίζουν τις κινήσεις τους μέσω του πολυνηματισμού και της παραλληλοποίησης που έγινε στο πρόγραμμα. Επίσης δημιουργήθηκε νέα μέθοδος η οποία επέτρεπε στην camera JeVois να ανιχνεύει τα αντικείμενα που βρίσκονται στο conveyor belt και επικοινωνώντας με τον βραχίονα να του στέλνει που πρέπει να τοποθετηθεί το κάθε αντικείμενο ανάλογα με το είδος του. Επιπρόσθετα, έχει δημιουργηθεί με 3D Printing μια βάση για την JeVois Camera, ώστε να μπορεί να τοποθετηθεί σε οποιοδήποτε σημείο δίπλα από τον ιμάντα και να είναι σταθερή. Τέλος, πραγματοποιήθηκε η ανάπτυξη μιας μεθόδου για την επικοινωνία της JeVois camera με τα Dobots, ώστε να είναι εφικτή η χρήση από τους ρομποτικούς βραχίονες και τις δυνατότητες της κάμερας.

# Περιεχόμενα

# Κεφάλαιο 1

## Εισαγωγή



### 1.1 Ο κλάδος της ρομποτικής

Η ρομποτική είναι ένα διεπιστημονικό πεδίο που ενσωματώνει την επιστήμη των υπολογιστών και τη μηχανική. Ο κόσμος τα τελευταία χρόνια φαίνεται να είναι εντυπωσιασμένος με τα ρομπότ. Καθώς η τεχνολογία εξελίσσεται, το ίδιο ισχύει και για τη ρομποτική. Τα ρομπότ αποκτούν μηχανικές δυνατότητες κάτι που μας κάνει να πιστεύουμε ότι στο μέλλον μπορεί να αποκτήσουν αυτονομία και κρίση. Με την απόκτηση αυτών των δυνατοτήτων θα μπορεί να καταλαβαίνει πότε να ενεργήσει για να παρέχει βοήθεια και πότε όχι. Όλα τα ρομπότ αποτελούνται από κάποιο είδος μηχανικής κατασκευής. Τα μηχανικά χαρακτηριστικά ενός ρομπότ το βοηθούν να ολοκληρώσει εργασίες στο περιβάλλον για το οποίο έχει σχεδιαστεί. Σήμερα, βλέπουμε έναν εξελιγμένο ορισμό της ρομποτικής που περιλαμβάνει την ανάπτυξη, τη δημιουργία και τη χρήση ρομπότ που εξερευνούν ακόμη και τις πιο σκληρές συνθήκες της Γης, ρομπότ που βοηθούν στην επιβολή του νόμου, ακόμη και ρομπότ που βοηθούν στον τομέα της υγείας με μεγάλο αντίκτυπο στην αποκατάσταση. Η βιομηχανία ρομποτικής είναι ακόμα νέα, αλλά έχει ήδη κάνει καταπληκτικά βήματα. Από τα βάθη των ωκεανών μας, έως τα υψηλότερα ύψη των βουνών, μέχρι και στο διάστημα. Τα ρομπότ μπορούν να εκτελούν εργασίες που οι άνθρωποι δεν μπορούσαν να ονειρευτούν πως θα επιτύχουν.

Τα ρομπότ έχουν τεράστιες δυνατότητες τις οποίες προσπαθούμε σιγά σιγά να εκμεταλλευτούμε. Μέχρι τώρα προσπαθούσαμε να βάζουμε τα ρομπότ να αναλαμβάνουν σκληρές και επικίνδυνες δουλειές, εργασίες που είναι επαναλαμβανόμενες και χρειάζονται μεγάλη ακρίβεια, δηλαδή πράγματα τα οποία μπορεί να κάνει μια μηχανή πολύ πιο

αποδοτικά από ένα άνθρωπο. Μια τακτική επαναλαμβανόμενη εργασία όπως το σφίξιμο μιας βίδας σε μια συσκευή είναι προτιμότερο να γίνεται από ρομπότ παρά από ανθρώπους, διότι τα ρομπότ εκτελούν επαναλαμβανόμενες ενέργειες με τεράστια ακρίβεια. Η εκτέλεση τέτοιου είδους εργασιών από τα ρομπότ συμβάλει στην οικονομία του χρόνου και επιτυγχάνεται με καλύτερη απόδοση και χωρίς να υπάρχει το πρόβλημα τραυματισμού κάποιου ανθρώπου. Σήμερα, τα ρομπότ χρησιμοποιούνται σε διάφορες εφαρμογές που βοηθούν την ανθρωπότητα.

### 1.2 Ο ρόλος της ρομποτικής στις ζωές μας

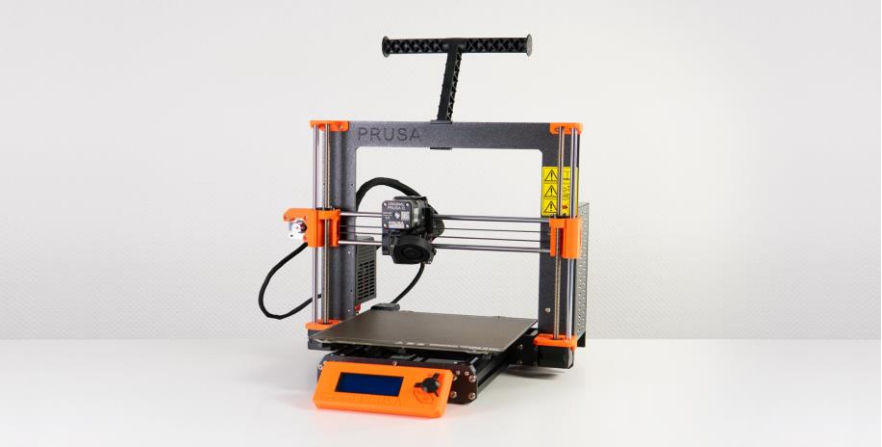
Τα θεμέλια για τον προγραμματισμό και τον έλεγχο κίνησης των ρομπότ αναπτύχθηκαν αρχικά με γνώμονα τις βιομηχανικές εφαρμογές. Αυτές οι εφαρμογές αξίζουν ιδιαίτερης προσοχής προκειμένου να κατανοήσουμε την προέλευση της ρομποτικής επιστήμης και να εκτιμήσουμε τα άλυτα προβλήματα που εξακολουθούν να εμποδίζουν την ευρύτερη χρήση ρομπότ στα κατασκευαστικά περιβάλλοντα.

Η ρομποτική βοηθά την αυτοματοποίηση[1] πολλών λειτουργιών και λύνει τεράστια προβλήματα στην καθημερινότητα μας που υπό άλλες συνθήκες θα δημιουργούσαν πονοκεφάλους στον καθένα μας. Για παράδειγμα, όταν έχουμε οποιονδήποτε τραυματισμό και χρειάζεται αποκατάσταση της πληγής στο σώμα μας, όλα γίνονται πιο εύκολα με την ρομποτική αποκατάστασης και πράγματα που μπορούν να μας παρέχουν.[2] Επίσης, υπάρχουν ρομπότ που χρησιμοποιούνται για την γεωργία εφαρμόζοντας λιπάσματα, φυτοφάρμακα και νερό στα χωράφια. Υπάρχει ο τομέας της θαλάσσιας ρομποτικής με την βοήθεια αλγορίθμων πλοήγησης και ελέγχου για επιφανειακά και υποβρύχια οχήματα. Επίσης υπάρχουν τα εξερευνητικά ρομπότ όπως το mars rover τα οποία στέλνονται στο διάστημα για εξερεύνηση. Με λίγα λόγια μέσω της ρομποτικής η ανθρωπότητα αγγίζει νέους ορίζοντες που δεν θα μπορούσαμε ποτέ να φανταστούμε. Σε αυτή την εργασία θα ασχοληθούμε με ένα βιομηχανικό ρομπότ αξιοποιώντας τις δυνατότητες του για να δημιουργήσουμε ένα όσο το δυνατό πιο ρεαλιστικό σενάριο χρήσης του στην πραγματική βιομηχανία.

### 1.3 Κατηγορίες Ρομποτικών βραχιόνων

Υπάρχουν πολλοί διαφορετικοί τύποι ρομποτικών βραχιόνων που διατίθενται στη σημερινή αγορά, καθένας από τους οποίους έχει σχεδιαστεί με σημαντικές βασικές ικανότητες και λειτουργίες που κάνουν συγκεκριμένους τύπους ιδιαίτερα κατάλληλους για συγκεκριμένους ρόλους. Η πλειοψηφία των ρομποτικών βραχιόνων έχει έως και 6 αρθρώσεις που συνδέουν 7 τμήματα, από τα οποία τα περισσότερα από αυτά οδηγούνται από κινητήρες και ελέγχονται από υπολογιστή. Αυτό επιτρέπει την ακριβή κίνηση και τοποθέτηση του βραχίονα, πράγμα το οποίο στις περισσότερες βιομηχανικές χρήσεις είναι κάποιο εξειδικευμένο εργαλείο σχεδιασμένο να εκτελεί μια συγκεκριμένη επαναλαμβανόμενη ενέργεια.
Ως επί το πλείστων, η βασική διάκριση μεταξύ διαφορετικών ειδών ρομποτικών βραχιόνων φαίνεται στον τρόπο με τον οποίο οι αρθρώσεις τους είναι σχεδιασμένες να κινούνται, δηλαδή το εύρος των κινήσεων και των λειτουργιών που μπορούν να εκτελέσουν. Το άλλο κριτήριο είναι ο τύπος πλαισίου που υποστηρίζεται το ρομπότ. Μερικοί από τους πιο συχνά χρησιμοποιούμενους τύπους προγραμματιζόμενων ρομπότ που χρησιμοποιούνται σε όλες τις βιομηχανίες.[16]

1) Υπάρχουν οι καρτεσιανοί ρομποτικοί βραχίονες οι οποίοι παίρνουν την ονομασία τους από το καρτεσιανό σύστημα συντεταγμένων. Οι καρτεσιανές συντεταγμένες είναι ουσιαστικά αυτό που μας δίνει το σύστημα αξόνων X, Y και Z που βλέπουμε σχεδόν πάντα σε γραφικές παραστάσεις. Αυτοί οι ρομποτικοί βραχίονες χρησιμοποιούνται για μεταφορά αντικειμένων στον τρισδιάστατο χώρο. Μια άλλη χρήση αυτού του είδους ρομποτικών βραχιόνων είναι το 3D Printing, το οποίο απαιτεί ακρίβεια στον τρισδιάστατο χώρο. Αυτή η ακρίβεια σε συνδυασμό με την ταχύτητα κάνουν τους καρτεσιανούς βραχίονες να είναι απαραίτητοι στον τομέα του 3D Printing. Έγινε χρήση του καρτεσιανού ρομποτικού βραχίονα 3D printer Original Prusa i3 MK3S στα πλαίσια της εργασίας.




**Εικόνα 1.3.1 Καρτεσιανός ρομποτικός βραχίονας**

2) Οι κυλινδρικοί βραχίονες ρομπότ, σε αντίθεση με τους καρτεσιανούς βραχίονες, είναι αυτοί των οποίων οι άξονες σχηματίζουν ένα κυλινδρικό σύστημα συντεταγμένων δηλαδή κινούνται σε κυλινδρικό χώρο. Αυτοί οι τύποι βραχιόνων χρησιμοποιούνται περισσότερο για εργασίες συναρμολόγησης και συγκόλλησης, όπου οι περιστροφικοί σύνδεσμοι τους δίνουν περιστροφική και γραμμική κίνηση.

**Εικόνα 1.3.2 Κυλινδρικός ρομποτικός βραχίονας**

3) Πολικοί ή Σφαιρικοί ρομποτικοί βραχίονες έχουν ένα συγκεκριμένο σφαιρικό χώρο εργασίας στον οποίο μπορούν να κινηθούν ελεύθερα. Ο πολικός ρομποτικός βραχίονας συνδέεται στη βάση του μέσω ενός στρεφόμενου συνδέσμου ο οποίος τον κάνει χρήσιμο για την εκτέλεση παρόμοιων ρόλων όπως κυλινδρικοί ρομποτικοί βραχίονες.

**Εικόνα 1.3.3 Σφαιρικός ρομποτικός βραχίονας**

4) SCARA ρομποτικοί βραχίονες είναι ο κλασικός τύπος που υπάρχει στις γραμμές παραγωγής εργοστασίων. Οι βραχίονες αυτοί χρησιμοποιούνται περισσότερο σε εφαρμογές συναρμολόγησης και συλλογής και τοποθέτησης αντικειμένων. Δεν είναι όσο ευέλικτοι είναι οι άλλοι βραχίονες, όμως έχουν την δυνατότητα να εισάγουν εξαρτήματα σε στενούς χώρους με μεγάλη ευκολία.

**Εικόνα 1.3.4 Ρομποτικός βραχίονας SCARA**

5) Αρθρωτοί ρομποτικοί βραχίονες έχουν σφαιρική βάση η οποία περιστρέφεται όπως και οι κυλινδρικοί και σφαιρικοί βραχίονες. Επίσης έχουν αρθρώσεις οι οποίες επιτρέπουν να κινούνται σαν το χέρι. Αυτό τους κάνει ιδιαίτερα χρήσιμους σε σενάρια παραλαβής και μετακίνησης αντικειμένων όπως το σενάριο που ασχολούμαστε σε αυτή την διπλωματική. Ένας αρθρωτός βραχίονας είναι και αυτός που επιλέξαμε να χρησιμοποιήσουμε ο Dobot Magician.

**Εικόνα 1.3.5 Αρθρωτός ρομποτικός βραχίονας**

6) Ρομποτικοί βραχίονες Delta. Μοιάζουν με αράχνη και αποτελούνται από αρθρωτά παραλληλόγραμμα συνδεδεμένα σε μια κοινή βάση. Χρησιμοποιούνται έντονα στις βιομηχανίες τροφίμων, φαρμακευτικών και ηλεκτρονικών. Είναι κατάλληλα για λεπτές και ακριβές κινήσεις.

**Εικόνα 1.3.4 Ρομποτικός βραχίονας DELTA**

Οι τυπικοί ρομποτικοί βραχίονες διαθέτουν έξι άξονες κίνησης με έξι βαθμούς ελευθερίας. Αυτός ο σχεδιασμός επιτρέπει μέγιστη ευελιξία. Τέτοιου είδους βραχίονες είναι ιδανικοί για δουλειές τύπου συγκολλήσεις, χειρισμούς υλικών ή εργαλείων, φροντίδα μηχανών και άλλα.

### 1.4 Σκοπός της εργασίας

Η διπλωματική εργασία έχει γίνει με σκοπό την ανάπτυξη εφαρμογών για τον ρομποτικό βραχίονα Dobot Magician. Οι εφαρμογές που δημιουργήθηκαν είχαν σκοπό την υλοποίηση ενός πραγματικού σεναρίου χρήσης σε μια βιομηχανία με την χρήση του ρομποτικού βραχίονα. Επίσης οι εφαρμογές υλοποιήθηκαν έτσι ώστε να επιτρέπουν την εισαγωγή και την επεξεργασία δεδομένων για την αποτελεσματική χρήση του βραχίονα. Για την επίτευξη της εργασίας δημιουργήθηκε πρόγραμμα σε γλώσσα Python το οποίο λειτουργεί σαν ένας controller που επεξεργάζεται τα δεδομένα που δέχεται από τον χρήστη και την κάμερα που λειτουργεί σαν sensor και δίνει τις ανάλογες εντολές για την μετακίνηση του κάθε βραχίονα. Πιο αναλυτικά, στο πρόγραμμα έχει χρησιμοποιηθεί συνάρτηση που είναι υπεύθυνη για την επικοινωνία της κάμερας με τον βραχίονα που κάνει την ταξινόμηση, έτσι όταν αναγνωρίζει κάποιο αντικείμενο η κάμερα πάνω στον ιμάντα τότε να ενημερώνει τον βραχίονα με το όνομα του αντικειμένου, για να μπορεί στην συνέχεια ο βραχίονας με ακρίβεια να μπορεί να το μεταφέρει στην σωστή βάση. Συνοψίζοντας, το πρόγραμμα που υλοποιήθηκε τρέχει πάνω σε υπολογιστή με λειτουργικό windows 10, το οποίο είναι ενωμένο με τους δύο βραχίονες και τις δύο κάμερες JeVois για να επιτρέπει την άμεση επικοινωνία και συγχρονισμό μεταξύ τους. Η προσπάθεια αποσκοπούσε στο να δημιουργηθεί ένα ρεαλιστικό σενάριο χρήσης των δύο ρομποτικών βραχιόνων και την αξιοποίηση των χαρακτηριστικών τους στον μέγιστο βαθμό.

### 1.5 Δομή διπλωματικής εργασίας

Στο πρώτο κεφάλαιο αναφέρονται κάποια γενικά πράγματα για την ρομποτική και τον ρόλο που παίζει στην ζωή μας. Στη συνέχεια, γίνεται αναφορά στους ρομποτικούς βραχίονες που χρησιμοποιήθηκαν, τις λειτουργίες, τα εξαρτήματα και τους τρόπους χρήσης τους. Μετά θα μιλήσουμε για το API, τον τρόπο που χρησιμοποιήθηκε και τις βασικές εντολές του στο κεφάλαιο 2. Στο κεφάλαιο 3 ασχοληθήκαμε με την JeVois camera. Αιτιολογούμε γιατί επιλέξαμε την συγκεκριμένη κάμερα και τις δυνατότητες που έχει. Αναφέρουμε κάποια προβλήματα που αντιμετωπίσαμε και τα scripts που αρχικοποιούν τις παραμέτρους που χρειαζόμαστε για να τρέξουν τα machine learning modules της JeVois. Στο τέλος του κεφαλαίου μιλούμε για την εκπαίδευση της κάμερας και την σύνδεση με το python πρόγραμμα μας. Στο κεφάλαιο 4 γίνεται αναφορά στο Docker και τα docker containers, τις χρήσεις τους και το πως μπορούν να χρησιμοποιηθούν από μελλοντικούς χρήστες. Στη συνέχεια, στο κεφάλαιο 5 αναλύουμε την τελική διάταξη της διπλωματικής, τα προγράμματα που εκτελούνται, τους κώδικες των προγραμμάτων, κάποια προβλήματα που

αντιμετωπίστηκαν και τα αποτελέσματα. Στα αποτελέσματα και μετρήσεις αναφέρονται διάφορες παρατηρήσεις σε σχέση με την απόδοση του προγράμματος. Στο κεφάλαιο 6 παρουσιάζουμε κάποια συμπεράσματα και στο τέλος έχουμε τις εισηγήσεις για την μελλοντική εργασία και την βιβλιογραφία.

# Κεφάλαιο 2

## Περιγραφές Συστημάτων



### 2.1 Dobot Magician

Το Dobot Magician είναι ένας αρθρωτός ρομποτικός βραχίονας πολλαπλών λειτουργιών για πρακτική εκπαίδευση [3]. Εγκατεστημένο με διαφορετικά εργαλεία, το Dobot Magician μπορεί να πραγματοποιήσει διάφορες λειτουργίες όπως τρισδιάστατη εκτύπωση, χάραξη με λέιζερ, γραφή και σχέδιο. Το DOBOT έχει την δυνατότητα επίσης να σηκώνει πράγματα και να τα μεταφέρει με δύο τρόπους, είτε με το gripper, είτε με το suction cup το οποίο θα χρησιμοποιήσουμε. Μπορεί να αναπτυχθεί με διάφορες γλώσσες προγραμματισμού, ROS, PLC, Microcontroller και Arduino, όμως εμείς στην συγκεκριμένη εργασία θα ασχοληθούμε με ανάπτυξη σε γλώσσα Python. Επίσης, μπορεί να ελεγχθεί μέσω προγραμματισμού όπως κάναμε εμείς με την βοήθεια της γλώσσας Python, το ίδιο μπορεί να γίνει και με την βοήθεια Bluetooth, WiFi, ποντικιού και άλλων. Το Dobot Magician είναι συμβατό για την ανάπτυξη εφαρμογών. Επίσης υποστηρίζει API και διάφορα πρωτόκολλα που έχουν κυκλοφορήσει. Αξιοποιήθηκε ένα σεβαστό ποσό τον εντολών του API που παρέχεται για την υλοποίηση του προγράμματος. Επίσης το Dobot έχει την δυνατότητα να ελέγξει τον ιμάντα αντικειμένων και να συνδυάζει τις κινήσεις του μαζί του. Ο βραχίονας Dobot Magician δημιουργήθηκε για σκοπούς διδασκαλίας και εκπαίδευσης. Στην εργασία εκμεταλλευτήκαμε τον βραχίονα και δημιουργήσαμε ένα τεχνητό σενάριο που χρησιμοποιεί διάφορα χαρακτηριστικά του βραχίονα σε συνδυασμό με άλλες μηχανές για να δείξουμε τις δυνατότητες του ή των παρόμοιων βραχιόνων στον τομέα της βιομηχανίας.

## 2.2 Λειτουργίες και εξαρτήματα του βραχίονα

Ο βραχίονας έρχεται μαζί με διάφορα εξαρτήματα που το καθένα μπορεί να κάνει τις δικές του λειτουργίες[8].

Κάποια από τα εξαρτήματα του είναι :

1) Χάραξη με λέιζερ : Επιτρέπει την χάραξη ειδικών επιφανειών με λέιζερ.
2) Τρισδιάστατη εκτύπωση : Επιτρέπει την τρισδιάστατη εκτύπωση με την χρήση του πλαστικού Polylactic Acid.
3) Πένα : Μπορούμε να αναπτύξουμε εφαρμογές με τις οποίες ο βραχίονας να σχεδιάζει.
4) Gripper : Η πιάστρα μας δίνει την δυνατότητα μεταφοράς αντικειμένων.
5) Suction cup : Με την βεντούζα μας δίνετε η δυνατότητα μεταφοράς αντικειμένων.
6) Linear Rail Kit: Η γραμμική ράγα μας δίνει την δυνατότητα να μετακινούμε γραμμικά σε μια ευθεία τον βραχίονα.
7) Conveyor Belt Kit: Συμπεριλαμβάνονται ο ιμάντας μεταφοράς αντικειμένων μαζί με τον αισθητήρα χρωμάτων ο οποίος αναγνωρίζει τα τρία βασικά χρώματα και τον photoelectric αισθητήρα, για εντοπισμού αντικειμένου σε συγκεκριμένο σημείο.
8) Controller: Μοχλός ελέγχου του βραχίονα.
9) Wi-Fi kit : Περιλαμβάνει το εξάρτημα που επιτρέπει την ασύρματη σύνδεση του βραχίονα με τον υπολογιστή.
10) Bluetooth kit : Περιλαμβάνει το εξάρτημα που επιτρέπει την Bluetooth σύνδεση του βραχίονα με τον υπολογιστή.

## 2.3 Τρόποι χρήσης του βραχίονα

Ο βραχίονας μπορεί να προγραμματιστεί και να χρησιμοποιηθεί με διάφορους τρόπους. Αρχικά, ο ποιο κοινός τρόπος αξιοποίησης και ελέγχου του βραχίονα είναι μέσω του Dobot Magician SDK, δηλαδή το επίσημο λογισμικό της εταιρίας. Μέσω του συγκεκριμένου λογισμικού μας δίνεται η ευκαιρία να προγραμματίσουμε σε περιβάλλον φιλικό προς εμάς τις λειτουργίες του βραχίονα. Μέσω του λογισμικού μπορούμε επίσης να επιλέξουμε τις διάφορες κινήσεις του βραχίονα με το Teaching & Playback το οποίο απαιτεί από εμάς να δίνουμε συγκεκριμένες συντεταγμένες στον βραχίονα, όπως με το να τον τοποθετούμε και να τις αναγνωρίζει έτσι ώστε να μπορεί στην συνέχεια να μιμηθεί την κίνηση και να μεταβεί στις ακριβές συντεταγμένες που του ορίσαμε. Επιπρόσθετα έχουμε την δυνατότητα να ελέγξουμε και τα διάφορα εργαλεία που έχει, όπως για παράδειγμα το suction cup. Έτσι μέσω του Teaching & Playback μπορούμε να κάνουμε κάποιες ολοκληρωμένες κινήσεις.

Μέσω του Write & Draw μπορούμε να ζωγραφίσουμε γράμματα, λέξεις και εικόνες με την βοήθεια του βραχίονα. Για την σωστή λειτουργία του Write & Draw πρέπει να προσκομιστεί μια εικόνα ή να γραφτούν οι λέξεις για να παράξει το αποτέλεσμα ο βραχίονας. Μπορούμε να προσαρμόσουμε το αποτέλεσμα στις ανάγκες μας προτού δημιουργηθεί από τον βραχίονα. Μέσω του Blockly και του Script μπορεί ουσιαστικά να γίνει η ίδια δουλειά, η οποία είναι ο προγραμματισμός με την βοήθεια γλώσσας προγραμματισμού. Η διαφορά είναι ότι μέσω του Blockly δίνονται έτοιμες κάποιες εντολές του API για να γίνεται πιο εύκολη η χρήση τους. Επίσης δίνεται πιο γραφική αναπαράσταση των εντολών για να καταλαβαίνει κάποιος τον τρόπο που τρέχουν οι εντολές και λειτουργεί ο βραχίονας. Μέσω Script μπορούμε πιο παραδοσιακά να γράψουμε ένα πρόγραμμα σε γλώσσα που υποστηρίζει ο βραχίονας και να το τρέξουμε.

Υπάρχει επίσης το Leap Motion που έρχεται μαζί με τον motion sensor του. Το Leap Motion λειτουργεί σαν ένα ανεξάρτητο λογισμικό που ελέγχει πλήρως τον βραχίονα μέσα από τις κινήσεις της παλάμης του χεριού. Τέλος, το Mouse λειτουργεί σαν ένας controller στον οποίο πρέπει να δώσουμε τις απαραίτητες παραμέτρους και θα μπορεί να υπάρξει κάποιος έλεγχος, ο οποίος είναι περιορισμένος.

Όλες οι λειτουργίες του βραχίονα μπορούν να τρέξουν και να ελεγχθούν μέσω του iOS app control, Arduino control through UART, Qt creator control, STM32 control, Visual basic control, καθώς επίση και μέσω της Python.


**Εικόνα 2.3.1 Dobot Studio**

## 2.4 Dobot Magician API

Το API χρησιμοποιήθηκε για την δημιουργία των εφαρμογών του βραχίονα. Υπάρχουν δύο δυνατότητες εντολών επικοινωνίας κατά την επικοινωνία με τον Dobot controller[4] :

1) Όλες οι εντολές μετά την εκτέλεση τους επιστραφούν στον controller. Υποστηρίζονται εντολές που ορίζουν(set commands) π.χ. SetPTPJointParams, στις οποίες ο controller ορίζει κάποιες παραμέτρους με διαφορετική αξία. Επίσης υπάρχουν και εντολές που παίρνουν(get commands), π.χ. GetPTPJointParams στις οποίες ο controller επιστρέφει την αξία που ζητήθηκε.
2) Ο Dobot controller υποστηρίζει δύο είδη εντολών:
**Άμεση εντολή** : Ο Dobot controller θα επεξεργαστεί την εντολή μόλις ληφθεί ανεξάρτητα από το εάν υπάρχουν διεργασίες με άλλες εντολές οι οποίες τρέχουν ακόμη ή όχι στον controller.
**Εντολή ουράς**: Ο Dobot controller λαμβάνει την εντολή ουράς, η εντολή μπαίνει στην ουρά εσωτερικής εντολής του ελεγκτή. Ο controller θα εκτελέσει οδηγίες με τη σειρά με την οποία η εντολή προωθήθηκε στην ουρά.

Κάποιες από τις βασικές εντολές του API που χρησιμοποιήθηκαν ήταν :

- SetPTPCmdEx : Μεταφορά του βραχίονα από ένα σημείο σε άλλο σημείο δίνοντας συγκεκριμένες συντεταγμένες.
- SetEndEffectorSuctionCup : Ενεργοποιεί και απενεργοποιεί το Suction Cup του βραχίονα.
- SetEMotorSex : Ενεργοποιεί την κίνηση του ιμάντα αντικειμένων για συγκεκριμένο χρονικό διάστημα.
- SetEMotorS : Ενεργοποιεί ή απενεργοποιεί την κίνηση του ιμάντα αντικειμένων.

Για την χρήση του API έγιναν κάποιες αλλαγές οι οποίες ήταν απαραίτητες για την διευκόλυνση μας και για την διευκόλυνση οποιουδήποτε θέλει να ασχοληθεί με τον ρομποτικό βραχίονα Dobot. Πιο συγκεκριμένα δημιουργήθηκαν οι μέθοδοί loadX() , ConnectDobotX() , DisconnectDobotX() . Αρχικά η κάθε μέθοδος αντικατέστησε την load() , ConnectDobot() , DisconnectDobotX() αντίστοιχα. Η load() κατά την εκκίνηση δημιουργούσε μια μόνο σύνδεση με αποτέλεσμα να μπορούμε να επικοινωνούμε μόνο με ένα Dobot στο πρόγραμμα μας μέχρι να τελειώσουμε να κλείσουμε την σύνδεση και να ξεκινήσουμε μια νέα.(εικόνα 2.4.1 Αρχικός κώδικας λειτουργίας σύνδεσης με API) Οι αλλαγές που έγιναν στην μέθοδο load(), ήταν ότι προστέθηκε το connection DLL και των

δύο Dobot έτσι ώστε να μπορούμε να ενωνόμαστε και με τα δύο Dobot και να τρέχουμε εντολές ταυτόχρονα χωρίς καμία διακοπή. Η συγκεκριμένη λειτουργία φαίνεται ξεκάθαρα στην εικόνα 2.4.2 Τελικός κώδικας λειτουργίας σύνδεσης με API. Στη συνέχεια στη μέθοδο ConnectDobot() αλλάχτηκαν οι παράμετροι και πλέον αντί να δέχεται την βιβλιοθήκη που φορτώνεται από το load(), τώρα αρχικοποιείται με την μέθοδο loadX() με την οποία πλέον μπορούμε να ενώσουμε περισσότερα από ένα Dobots. Η νέα μέθοδος loadX() καλείται μέσα από την νέα ConnectDobotX(). Η τελευταία αλλαγή που έγινε ήταν στην μέθοδο DisconnectDobotX() με την οποία αποσυνδέουμε τα Dobots. Η διαφορά από την DisconnectDobot() είναι απλά ότι αντί να αποσυνδέει το ενωμένο Dobot, αποσυνδέει 1 ή περισσότερα Dobots που μπορεί ανά πάσα στιγμή να είναι ενωμένα.

```
# Load Dll
api = dType.load()

# Connect Dobot
state = dType.ConnectDobot(api, "", 115200)[0]
print("Connect status:", CON_STR[state])
```

**Εικόνα 2.4.1: Αρχικός κώδικας λειτουργίας σύνδεσης με API**

```
# Load Dll and get the CDLL object
dobot0, state1 = dType.ConnectDobotX("COM4")
dobot1, state2 = dType.ConnectDobotX("COM3")

# Connect Dobot
print("Connect status1:", CON_STR[state1[0]])
print("Connect status2:", CON_STR[state2[0]])
```

**Εικόνα 2.4.2: Τελικός κώδικας λειτουργίας σύνδεσης με API**

Οι αλλαγές στο API έγιναν από τον Ανδρέα Κομή στα πλαίσια της διπλωματικής εργασίας του.

# Κεφάλαιο 3

## JeVois Camera

---

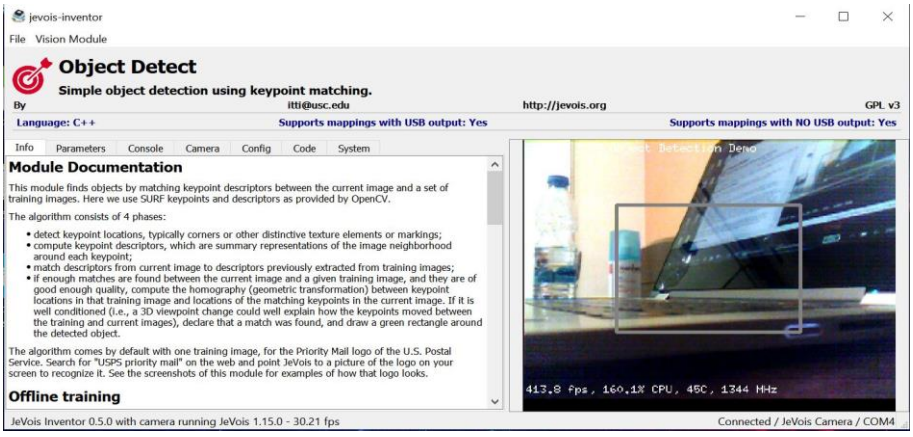


---

### 3.1 Επιλογή JeVois κάμερας

Η JeVois Camera επιλέχθηκε γιατί έχει την ικανότητα να εκτελεί βαθιά νευρωνικά δίκτυα που μπορούν να αναγνωρίσουν με πολλούς διαφορετικούς τύπους αντικειμένων σε πραγματικό χρόνο. Χρησιμοποιήθηκε με σκοπό να κάνει αναγνώριση των αντικειμένων που κινούνται στο conveyor belt και να επικοινωνεί μέσω τις εφαρμογής, η οποία στην συνέχεια θα στέλνει τις απαραίτητες εντολές ανάλογα με το αντικείμενο και το χρώμα του στον βραχίονα. Αρχικά για την ρύθμιση της κάμερας χρειάστηκε να κατεβάσουμε το λογισμικό Etcher το οποίο μας βοήθησε στο να κάνουμε flash το λογισμικό μέσα στην SD card της κάμερας. Στη συνέχεια έγινε εγκατάσταση του JeVois Inventor 1.0 λογισμικού που χρησιμοποιήθηκε με σκοπό την ρύθμιση της κάμερας και την εισαγωγή των παραμέτρων που χρειάζονται για να λειτουργεί σωστά και να καλύπτει τον σκοπό της εργασίας.


**Εικόνα 3.1.1 JeVois Camera**




**Εικόνα 3.1.2 JeVois Inventor**

### 3.2 Βάση κάμερας

Παρατηρήσαμε ότι λόγω του ανεμιστήρα ψύξης της κάμερας, η κάμερα δεν έμενε σταθερή κατά την διάρκεια της εκτέλεσης του προγράμματος και αυτό προκαλούσε την λανθασμένη αναγνώριση αντικειμένων. Για την αντιμετώπιση του συγκεκριμένου προβλήματος χρησιμοποιήθηκε το 3D printer Original Prusa i3 MK3S για την εκτύπωση βάσης η οποία θα κρατούσε σταθερή την κάμερα σε συγκεκριμένο σημείο έτσι ώστε να μην χρειάζεται να επαναλαμβάνουμε κάθε φορά την διαδικασία εκπαίδευσης της κάμερας για το κάθε αντικείμενο για να γίνεται σωστά η αναγνώριση. Η βάση που φαίνεται στην εικόνα 3.2.1 έχει την δυνατότητα να βιδωθεί σε οποιαδήποτε επιφάνεια για να παραμένει σταθερή κατά την λειτουργία των εργασιών της.

**Εικόνα 3.2.1 JeVois Camera 3D printed Base**

### 3.3 JeVois Inventor scripts για αρχικοποίηση και σωστή χρήση

Στο JeVois Inventor προσθέσαμε κάποια scripts. Το πρώτο script που προσθέσαμε ήταν το initscript.cfg το οποίο τρέχει πάντα κατά την εκκίνηση της κάμερας και κάνει όλες τις απαραίτητες αρχικοποιήσεις, όπως την επιλογή του module που θα χρησιμοποιήσουμε και τον τρόπο που εξάγονται οι πληροφορίες από την κάμερα, στην περίπτωσή μας στέλνει όλα τα δεδομένα στο usb port. Επίσης γίνονται αρχικοποιήσεις σε παραμέτρους όπως το στυλ των μνημάτων που εξάγει και τον τόπο που βρίσκονται οι εικόνες που χρησιμοποιήθηκαν για την εκπαίδευση. Το δεύτερο script που χρησιμοποιήθηκε ήταν το videomappings.cfg το οποίο είναι περιέχει όλα τα modules που μπορούν να χρησιμοποιηθούν. Η μόνη αλλαγή που χρειάστηκε να γίνει για να δουλεύει το module ήταν του USBmode σε NONE και η

διόρθωση των frames της κάμερας. Χρησιμοποιήθηκαν ακόμα δύο scripts τα οποία ήταν υπεύθυνα για τις αρχικοποιήσεις των παραμέτρων του επιλεγμένου module. Το ένα είναι το modulesparams.cfg το οποίο αρχικοποιεί κάποιες βασικές παραμέτρους του module. Το άλλο είναι το modulesscript.cfg το οποίο θέτει τις παραμέτρους serout και serlog σε USB. Και οι δύο εντολές προωθούν strings στα serial ports που δίνονται σαν παράμετρος δηλαδή το USB από το οποίο παίρνουμε τα σειριακά μηνύματα και τα επεξεργαζόμαστε με την PySerial στο πρόγραμμα μας.

```
JeVois smart camera operation modes and video mappings
# Format is: <USBmode> <USBwidth> <USBheight> <USBfps>
<CAMmode> <CAMwidth> <CAMheight> <CAMfps> <Vendor> <Module>
#From this : YUYV 320 252 30.0 YUYV 320 240 30.0 JeVois
ObjectDetect
#To this : NONE 0 0 0 YUYV 320 240 29.0 JeVois ObjectDetect
```

Με αυτές τις ρυθμίσεις η JeVois camera κάνει τις αρχικοποιήσεις που χρειάζεται την ώρα του start up έτσι μπορούμε εμείς κατευθείαν να τρέξουμε το πρόγραμμα μας και να πάρουμε τα αποτελέσματα από την κάμερα.

Επίσης πολύ σημαντικό είναι να αναφέρουμε ότι για να μπορούμε να δεχόμαστε μηνύματα από την JeVois στο πρόγραμμα Python αλλάξαμε τις ρυθμίσεις για να στέλνει δεδομένα σε 2D normal standardized messages. Υπάρχουν διάφοροι τύποι standardized μηνυμάτων. Τα 2D normal standardized messages είναι τύπου «N2 ID X Y W H» , όπου N2 είναι το είδος των μηνυμάτων δηλαδή κανονικά δισδιάστατα, ID είναι το όνομα του αντικειμένου που εντοπίστηκε από το Module, X-Y είναι οι συντεταγμένες του αντικειμένου και W-H είναι οι διαστάσεις του αντικειμένου.

| serstyle | message |
|---|---|
| Terse | `T2 x y` |
| Normal | `N2 id x y w h` |
| Detail | `D2 id x1 y1 x2 y2 x3 y3 x4 y4 extra` |
| Fine | `F2 id n x1 y1 ... xn yn extra` |

**Εικόνα 3.3.1 Πίνακας standardized message**

### 3.4 Εκπαίδευση και σύνδεση JeVois με το πρόγραμμα

Για να αξιοποιήσουμε την κάμερα εκμεταλλευτήκαμε την μέθοδο ObjectDetection που κάνει χρήση του Module Object Detection[10], η οποία καθορίζει το είδος του αντικειμένου που βρίσκεται πάνω στον ιμάντα αντικειμένων για να σταλεί στην συνέχεια η κατάλληλη εντολή στον βραχίονα που εκτελεί την Sorting μέθοδο και να ταξινομήσει το κάθε αντικείμενο αναλόγως με το είδος και το χρώμα στο κατάλληλο μέρος.[2] Για την σωστή χρήση του συγκεκριμένου module απαιτείται εκπαίδευση. Η εκπαίδευση μπορεί να γίνει εκτός σύνδεσης με την πρόσθεση ορισμένων φωτογραφιών των αντικειμένων που θέλουμε να μας αναγνωρίζει, στην κάρτα μνήμης της κάμερας. Επίσης μπορεί να γίνει και ζωντανή εκπαίδευση, δηλαδή να φωτογραφίζουμε τις εικόνες τις οποίες επιθυμούμε να αναγνωρίζει από την JeVois κάμερα. Η σύνδεση της κάμερας γίνεται μέσω της Python. Η κάμερα είναι ενωμένη σε serial port του υπολογιστή και επικοινωνεί με σειριακά μηνύματα σε ζωντανή ώρα. Μέσω του προγράμματος παίρνουμε αυτά τα σειριακά μηνύματα και ενημερώνουμε τους βραχίονες για τις επόμενες τους ενέργειες. Αυτό ισχύει και για τις δύο κάμερες που είναι ενωμένες και στους δύο ιμάντες αντικειμένων.

# Κεφάλαιο 4

## Docker



### 4.1 Εισαγωγή στο Docker

Το Docker είναι ένα εργαλείο για τη δημιουργία, ανάπτυξη και εκτέλεση εφαρμογών σε κοντέινερ. Το λογισμικό είναι δημοφιλές στους προγραμματιστές καθώς επιταχύνει τη διαδικασία ανάπτυξης και δεν χρησιμοποιεί πολλούς πόρους. Τα Docker containers είναι ελαφριά, ειδικά σε σύγκριση με εικονικές μηχανές. Αυτός είναι και ένας από τους κύριους λόγους που επιλέξαμε τα Docker containers.

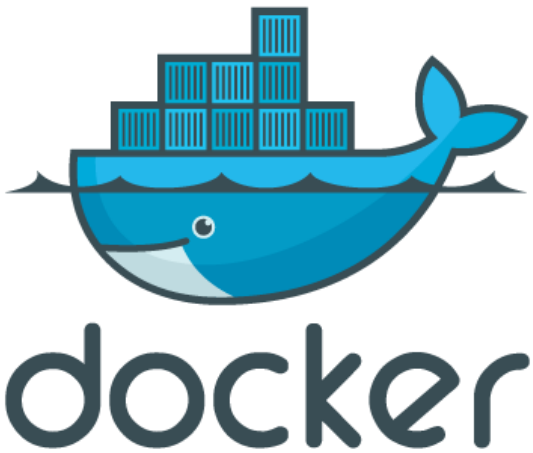
Κάποιοι από τους λόγους για τους οποίους επιλέξαμε το Docker είναι ότι μας παρέχει ευελιξία στις κινήσεις μας. Με την λέξη ευελιξία αναφερόμαστε στη λήψη λειτουργιών και ενημερώσεων πολύ γρήγορα. Επίσης το περιβάλλον του Docker είναι συνεπές και απομονωμένο, πράγμα που μας δίνει την δυνατότητα σαν προγραμματιστές να μπορούμε να δημιουργήσουμε προβλέψιμα περιβάλλοντα που είναι απομονωμένα από άλλες εφαρμογές. Αυτό σημαίνει ότι κάποιος που θέλει να τρέξει το πρόγραμμα μας σε άλλη συσκευή δεν χρειάζεται να ταλαιπωρείται με την εγκατάσταση οποιονδήποτε εξαρτήσεων που χρειάζεται το πρόγραμμα να τρέξει. Το μόνο που χρειάζεται να γίνει, είναι να έχει εγκατεστημένο το Docker και να κατεβάσει το docker container από το docker hub και να το τρέξει. Στόχος της εργασίας είναι το πρόγραμμα να γίνει όσο φορητό και εύκολο στην διαχείριση σε άλλα συστήματα γίνεται και εδώ παίζει τεράστιο ρόλο το Docker.

### 4.2 Εγκατάσταση Docker στον υπολογιστή

Για την εγκατάσταση του Docker ακολουθήσαμε τα εξής βήματα:

1) Βεβαιωθήκαμε ότι το μηχάνημά μας πληρεί τις απαιτήσεις για την εγκατάσταση.
2) Βεβαιωθήκαμε ότι τρέχουμε στο μηχάνημα λειτουργικό σύστημα που συμπίπτει στις κατηγορίες Windows 10 64-bit: Home, Pro, Enterprise, or Education, version 1903 (Build 18362 or higher).
3) Ενεργοποιήσαμε τη δυνατότητα WSL 2 στα Windows. Ο λόγος που χρειάζεται να ενεργοποιήσουμε το WSL 2 είναι για την εκτέλεση binary executable Linux εγγενώς στα Windows[6].
4) Ελάχιστα χαρακτηριστικά μηχανήματος :
    - 64-bit processor με Second Level Address Translation (SLAT)
    - 4GB system RAM
    - BIOS-level hardware virtualization support να είναι ενεργοποιημένο στα BIOS settings
5) Να κατεβάσουμε και να εγκαταστήσουμε το Linux kernel update package.

Η εγκατάσταση του Docker Desktop[5] περιλαμβάνει το Docker Engine, Docker CLI client, Docker Compose, Notary, Kubernetes και Credential Helper.




**Εικόνα 4.1.1 docker logo**

### 4.3 Δημιουργία docker container και docker image

Για την δημιουργία του docker container γράφτηκε ένα Dockerfile. Με αυτό τον τρόπο εισάγουμε και το πρόγραμμα μας μέσα στο docker container[9].

```
# set base image (host OS)

FROM python:3

# set the working directory in the container

WORKDIR /usr/src/Dobot

# copy the dependencies file to the working directory

COPY requirements.txt ./

# install dependencies

RUN pip install --no-cache-dir -r requirements.txt

# copy the content of the local directory to the working directory

COPY . .

# Execute program

CMD [ "python", "./control.py" ]
```

Για την εύρεση των εξαρτήσεων χρησιμοποιήσαμε την εντολή pip freeze η οποία τυπώνει την λίστα των πακέτων του τρέχων περιβάλλοντος. Το αρχείο requirements περιέχει όλες τις εξαρτήσεις του προγράμματος οι οποίες θα εγκατασταθούν :

```
pylint==2.6.0

python-dateutil==2.8.1

PyYAML==5.4.1

pyparsing==2.4.7

pyserial==3.5

threadpool==1.3.2
```

Στη συνέχεια για την δημιουργία του docker container τρέξαμε την εντολή :

```
docker build -t Dobot .
```

Με αυτό τον τρόπο δημιουργούμε το docker image. Το docker build θα κατεβάσει αυτόματα όλα τα απαραίτητα αρχεία και εξαρτήσεις που θα χρειαστούμε για να τρέξουμε το πρόγραμμα μας.
Για την έναρξη του docker container χρησιμοποιούμε την εντολή :

```
docker run -ti Dobot
```

Με το flag -ti εννοούμε image name για να ορίσουμε με το όνομα πιο container να τρέξει. Μπορούμε επίσης να τρέξουμε docker ps -a να δούμε το ID του docker container και να το τρέξουμε με το ID.

Για την εισαγωγή του python προγράμματος μέσα στο docker container[14] χρησιμοποιούμε το Dockerfile. Αυτό γίνεται στο COPY . . όπου αντιγράφουμε τα περιεχόμενα του local directory στο directory που δουλεύουμε. Μετά με την εντολή CMD ["python", "./control.py"] τρέχουμε το πρόγραμμα μας μέσα στο docker container.

### 4.4 Σκοπός του docker στην εργασία

Το docker χρησιμοποιήθηκε με απώτερο σκοπό το πρόγραμμα μας να είναι φορητό και εύκολο στη χρήση από κάποιο νέο χρήστη. Μας προσφέρει, εκτός από ευχρηστία και αξιοπιστία στην ανάπτυξη του προγράμματος, και συνοχή διότι δεν απαιτεί από τον κάθε διαφορετικό χρήστη να εγκαταστήσει τα διαφορετικά πακέτα και εξαρτήσεις που χρησιμοποιούνται. Το ίδιο ισχύει και για τις συμβατές εκδοχές τους, επειδή είναι όλα ήδη εγκατεστημένα και έτοιμα για τρέξιμο. Το docker container είναι ανεβασμένο στην ιστοσελίδα του docker hub στον λογαριασμό με το όνομα pkyria14(το link στην βιβλιογραφία)[11]. Από την ιστοσελίδα μπορεί κανείς με την εντολή docker pull pkyria14/diplomatiki:dobot1 να το κατεβάσει και να το ξεκινήσει στον δικό του υπολογιστή. Επίσης δεν χρειάζεται να μεταφερθεί το πρόγραμμα στο docker container επειδή είναι ήδη εγκατεστημένο μαζί με τις απαραίτητες εξαρτήσεις. Με αυτό τον τρόπο μπορεί οποιοσδήποτε να αξιοποιήσει την εργασία και να την αναπτύξει ή να την προσαρμόσει για δική του χρήση. Παρομοίως η εργασία είναι και στο GitHub και μπορεί κάποιος να κατεβάσει και να εκμεταλλευτεί απευθείας των κώδικα.

# Κεφάλαιο 5

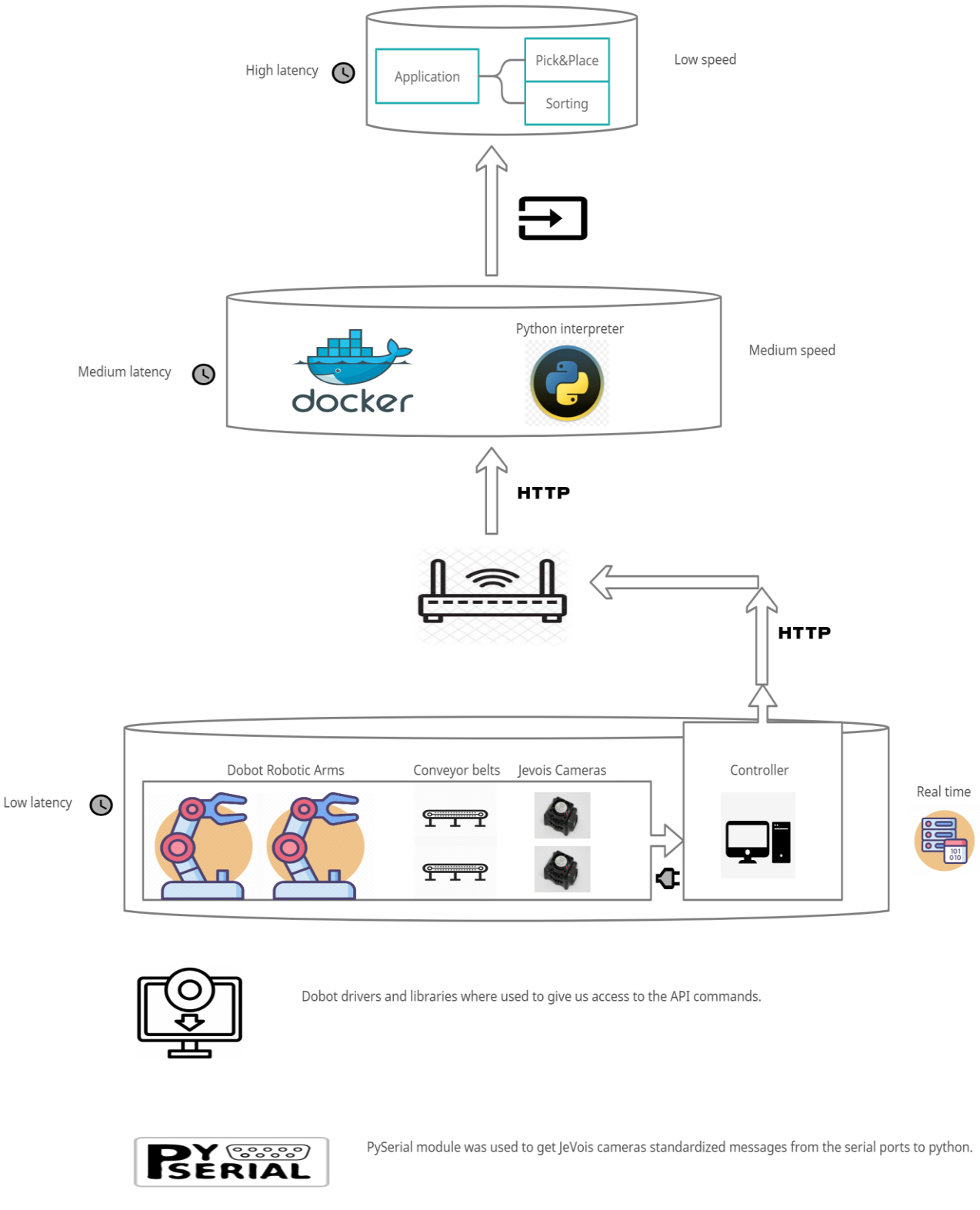
## Τελική διάταξη και πρόγραμμα python



### 5.1 Τελική διάταξη

Η τελική διάταξη της διπλωματικής έχει ως εξής. Αρχικά τοποθετήσαμε τα 2 Dobots το ένα απέναντι από το άλλο και το καθένα δίπλα από τους ιμάντες αντικειμένων που ελέγχουν. Οι 2 κάμερες JeVois τοποθετήθηκαν στα δεξιά και αριστερά του Dobot που τρέχει τον αλγόριθμο Sorting δηλαδή το Dobot το οποίο παραλαμβάνει τα αντικείμενα από τον ιμάντα αντικειμένων και τα διαχωρίζει. Οι κάμερες τοποθετήθηκαν σε αυτά τα σημεία για να έχουν καλή οπτική γωνία στα αντικείμενα και να μπορούν με ακρίβεια να δώσουν τα αποτελέσματα. Οι ιμάντες αντικειμένων βρίσκονται δίπλα από τους βραχίονες για να γίνεται εύκολη η τοποθέτηση και παραλαβή των αντικειμένων. Ο υπολογιστής-controller που τρέχει το πρόγραμμα μας είναι τοποθετημένος δίπλα στις 2 κάμερες και το Dobot.

## 5.2 Διάγραμμα και ανάλυση συστήματος




**Εικόνα 6.2 σχεδιάγραμμα συστήματος**

Οι ρομποτικοί βραχίονες, οι ιμάντες αντικειμένων, οι JeVois κάμερες και ο υπολογιστής είναι το hardware. Το hardware επικοινωνεί με το software μέσω του υπολογιστή που λειτουργεί σαν controller στο σύστημα και δίνει εντολές για τις κινήσεις που πρέπει να γίνουν. Ο υπολογιστής είναι ενωμένος στο διαδίκτυο και έχει εγκατεστημένο το docker το οποίο είναι ενθυλακωμένο το πρόγραμμα μας. Ο υπολογιστής μας τρέχει το λειτουργικό σύστημα Windows 10 και πληροί τα ελάχιστα κριτήρια για να τρέχει το docker με επιτυχία. Για να τρέξει το πρόγραμμα στο λειτουργικό αυτό χρειάζεται να γίνει εγκατάσταση των εξής modules:

1. Dobot driver from Dobot studio
2. PySerial from Python
3. Threading from Python

Αυτά τα modules είναι απαραίτητα για την λειτουργία διαφόρων σταδίων του προγράμματος. Το Dobot driver είναι υπεύθυνο για όλες τις λειτουργίες που έχουν να κάνουν με το API του Dobot, όπως τις διάφορες κινήσεις του βραχίονα και του ιμάντα αντικειμένων. Το PySerial είναι η βιβλιοθήκη της Python που μας επιτρέπει την εξαγωγή σειριακών μηνυμάτων από την JeVois camera στο πρόγραμμα μας και την επεξεργασία τους. Τέλος το threading είναι και αυτό μια βιβλιοθήκη της Python η οποία μας δίνει την δυνατότητα να τρέξουμε παράλληλα το πρόγραμμα μας. Με την παραλληλία αυτή εκμεταλλευόμαστε όλο τον χαμένο χρόνο και αυξάνουμε την ταχύτητα εκτέλεσης του προγράμματος μας.[7]

Στην εικόνα 6.2 παρατηρούμε τρία διαφορετικά layers που χωρίζονται ανάλογα με την ταχύτητα εκτέλεσης των λειτουργιών τους και με την ταχύτητα μεταφοράς δεδομένων. Στο πρώτο layer στο οποίο βρίσκεται το hardware έχουμε πολύ μικρές ταχύτητες για μεταφορά δεδομένων, όμως μεγάλους χρόνους και αργές ταχύτητες για την εκτέλεση των λειτουργιών που τρέχουν σε πραγματικό χρόνο(αρκετά δευτερόλεπτα). Στο δεύτερο επίπεδο έχουμε μικρές ταχύτητες για την εκτέλεση των λειτουργιών και για την μεταφορά δεδομένων. Σε αυτό το επίπεδο γίνονται οι απαραίτητες λειτουργίες για να τρέχει το docker container το python πρόγραμμα μας. Το docker χρησιμοποιεί αρχιτεκτονική client-server όπου ο client μιλά με τον daemon που κάνει όλη την βαριά δουλειά. Η επικοινωνία μεταξύ client και daemon γίνεται μέσω των sockets. Στο τρίτο επίπεδο βρίσκεται το πρόγραμμα μας το οποίο τρέχει σε γλώσσα προγραμματισμού python. Οι ταχύτητες επικοινωνίας σε αυτό το επίπεδο είναι μεγάλες, όπως και η εκτέλεση και μεταφορά δεδομένων. Ο λόγος είναι γιατί το πρόγραμμα εκτελεί απλές μεθόδους που δεν χρειάζονται ιδιαίτερη υπολογιστική δύναμη για να εκτελεστούν.

### 5.3 Εκτέλεση και επεξήγηση κώδικα προγράμματος

Η διαδικασία εκτέλεσης του προγράμματος αρχίζει από την ένωση των βραχιόνων με το πρόγραμμα. Αν γίνει επιτυχής ένωση, τότε τρέχουμε κάποιους ελέγχους για την ασφάλεια του προγράμματος και των βραχιόνων. Ελέγχουμε τις παραμέτρους που δόθηκαν από τον χρήστη αν ανήκουν στα επιτρεπτά όρια και μετά αρχικοποιούμε τις μεταβλητές. Οι παράμετροι που δέχεται το πρόγραμμα μας είναι ο αριθμός των αντικειμένων που θα μεταφερθούν, η απόσταση που θα μεταφερθούν τα αντικείμενα πάνω στον ιμάντα και οι ταχύτητες των αρθρώσεων των βραχιόνων. Οι έλεγχοι είναι απαραίτητοι γιατί δεν μπορεί να δοθεί αρνητικός αριθμός αντικειμένων για μεταφορά, ούτε τεράστιος αριθμός μεταφοράς των αντικειμένων πάνω στον ιμάντα. Επίσης δεν μπορούν να δοθούν μεγάλες ταχύτητες κίνησης των αρθρώσεων του βραχίονα γιατί θα δημιουργήσουν πρόβλημα. Στη συνέχεια τρέχει η μέθοδος PickandPlace που αναλαμβάνει την μετακίνηση των αντικειμένων από ένα σταθερό σημείο, πάνω στους ιμάντες αντικειμένων. Την στιγμή που ο βραχίονας τοποθετήσει το πρώτο αντικείμενο στον πρώτο ιμάντα, τότε το νήμα που τρέχει την μέθοδο belt_job αρχίζει να κινεί τον πρώτο ιμάντα αντικειμένων καθώς ο βραχίονας μεταφέρει το δεύτερο αντικείμενο στον άλλο ιμάντα αντικειμένων. Μόλις τοποθετηθεί το δεύτερο αντικείμενο στον ιμάντα τρέχει ξανά η μέθοδος belt_job για να ενεργοποιηθεί ο άλλος ιμάντας και να μετακινήσει το αντικείμενο στην εμβέλεια του δεύτερου βραχίονα. Μετά ο δεύτερος βραχίονας ο οποίος αναλαμβάνει την μεταφορά των αντικειμένων από τους ιμάντες αντικειμένων σε σταθερά σημεία, περιμένει την λειτουργία της κάμερας JeVois. Αν η κάμερα εντοπίσει ότι το αντικείμενο είναι κύβος, τότε ο βραχίονας το τοποθετεί σε ένα συγκεκριμένο σημείο. Αν εντοπιστεί ότι είναι οτιδήποτε άλλο τότε τοποθετείτε σε ξεχωριστό σημείο. Ο δεύτερος βραχίονας λειτουργεί την μέθοδο sorting και μεταφέρει τα αντικείμενα στα σημεία που πρέπει ανάλογα με το είδος τους, που προσδιορίζεται από τίς JeVois κάμερες. Η διαδικασία αυτή γίνεται και από τις δύο πλευρές και με τους δύο ιμάντες αντικειμένων και με τις δύο κάμερες JeVois, μέχρι να τελειώσει ο αριθμός αντικειμένων που δόθηκε στο πρόγραμμα.

Η main του προγράμματος είναι εδώ που γίνονται οι ενώσεις του προγράμματος με τα 2 Dobots και το API. Αρχικά παίρνουμε 3 παραμέτρους που είναι ο αριθμός αντικειμένων που θα μεταφερθούν, ταχύτητα των αρθρώσεων των Dobot και απόσταση που θα μετακινηθούν τα αντικείμενα πάνω στον ιμάντα. Οι παράμετροι στη συνέχεια περνούν από ελέγχους που δεν μας επιτρέπουν να δώσουμε παράμετρο εκτός ορίων διότι μπορεί να προκαλέσουμε σοβαρό πρόβλημα στα Dobots. Μετά γίνεται η ένωση των Dobot και η δημιουργία των states τους μέσω της εντολής dType.ConnectDobotX("COMX") του API. Τρέχουμε τον αλγόριθμο

PickandPlace ο οποίος καλεί τον αλγόριθμο Sorting και όταν τελειώσουν όλες οι λειτουργίες, αποσυνδεόμαστε με την εντολή dType.DisconnectAll(). [Παράρτημα Α – Κώδικας main]

```
Main(){
    get arguments
    check arguments
    connectdobot0()
    connectdobot1()

    PickandPlace()

    DisconnectAll()
}
```

**Εικόνα 5.3.1 Ψευδοκώδικας main**

Για την σωστή λειτουργία του προγράμματος γράφτηκαν οι μέθοδοι INITIALIZE_PickandPlace και INITIALIZE_Sorting οι οποίες κάνουν όλες τις απαραίτητες αρχικοποιήσεις μεταβλητών που χρειαζόμαστε. Η INITIALIZE_PickandPlace αρχικοποιεί τις παραμέτρους που χρειάζεται η μέθοδος PickandPlace, όπως τις μεταβλητές με τις συντεταγμένες των 4 σημείων που ανακτώνται τα αντικείμενα αρχικά και τις μεταβλητές με τις συντεταγμένες των 2 σημείων πάνω στους 2 ιμάντες αντικειμένων. Η INITIALIZE_Sorting αρχικοποιεί της μεταβλητές με τις συντεταγμένες των 2 σημείων που παραλαμβάνει τα αντικείμενα από πάνω από τους 2 ιμάντες αντικειμένων και τις μεταβλητές με τις συντεταγμένες των σημείων που τοποθετούνται τα αντικείμενα.

Στη συνέχεια έχουμε τους 2 κύριους αλγορίθμους του προγράμματος. Η δουλειά χωρίζεται στο πρόγραμμα ως εξής. Αρχικά τρέχει η μέθοδος PickandPlace τρέχει την μέθοδο INITIALIZE_PickandPlace και μετά δημιουργεί 2 νήματα τα οποία θα τρέχουν τις μεθόδους items_job και belt_job. Το πρώτο νήμα τρέχει την δουλειά η οποία είναι υπεύθυνη να μεταφέρει τα αντικείμενα από ένα σταθερό σημείο πάνω στους ιμάντες αντικειμένων. Το δεύτερο νήμα τρέχει την δουλειά η οποία είναι υπεύθυνη για την ενεργοποίηση του μοτέρ των ιμάντων. Στη συνέχεια η PickandPlace αρχίζει την λειτουργία των 2 νημάτων με κάποια δευτερόλεπτα διαφορά για λόγους συγχρονισμού. Συνεχίζοντας, κάθε δύο αντικείμενα που φτάνουν στο τέλος του ιμάντα και είναι έτοιμα για διαχωρισμό, ξεκινά η λειτουργία των JeVois καμερών δηλαδή η μέθοδος ObjectDetection. Τέλος μετά την αναγνώριση των

αντικειμένων από την μέθοδο ObjectDetection παίρνοντας τα αποτελέσματα τρέχει ο αλγόριθμος Sorting. [Παράρτημα Β – Κώδικας PickandPlace]

```
PickandPlace(){
    initialize_pickandplace()
    while (i < numberofitems){
        create_thread1(items_job)
        create_thread2(belt_job)
        if(every_two_times){
            Object_Detection(jevois1)
            Object_Detection(jevois2)
            Sorting()
        }
        sync_threads()
    }

}
```

**Εικόνα 5.3.2 Ψευδοκώδικας PickandPlace**

Στη συνέχεια ακολουθεί η μέθοδος ObjectDetection με την οποία αξιοποιούμε τις κάμερες JeVois. Η σύνδεση των δύο γίνεται μέσω των ports του υπολογιστή, στην περίπτωση μας η σύνδεση φαίνεται με τις εντολές (serdev = 'COM13'  και serdev2 = 'COM17'). Η μέθοδος παίρνει  σαν παραμέτρους τα ports των καμερών, έτσι χρειάζεται να καλεστεί 1 φορά για την κάθε κάμερα. Αρχικά η μέθοδος διαβάζει την γραμμή με δεδομένα που στέλνει η κάμερα με 2D standardized messages και τα αποθηκεύει στον πίνακα tok. Στη συνέχεια επιστρέφουμε το όνομα του αντικειμένου που έχει εντοπίσει η κάμερα. Γίνονται αρκετοί έλεγχοι όπως για παράδειγμα αν το μήνυμα που επέστρεψε η κάμερα έχει μήκος μεγαλύτερου του 1 και εάν έχει την σωστή μορφή δηλαδή κανονικό δισδιάστατο(normal 2D). [Παράρτημα Γ – Κώδικας Object Detection]

```
ObjectDetection(){
    with serial(jevois){
        while true{
            receive input
            handle input
            check input
            decode input
            return input
        }
    }
}
```

**Εικόνα 5.3.3 Ψευδοκώδικας Object Detection**

Ακολουθεί ο αλγόριθμος Sorting ο οποίος είναι υπεύθυνος για την μεταφορά και ταξινόμηση των αντικειμένων από τους ιμάντες σε σταθερά σημεία. Πρώτα, όπως και ο αλγόριθμος PickandPlace το ίδιο και ο Sorting αρχίζει με την αρχικοποίηση των μεταβλητών με την μέθοδο INITIALIZE_Sorting. Στη συνέχεια, αφού τρέξει ο αλγόριθμος ObjectDetection για την αναγνώριση του αντικειμένου που βρίσκεται πάνω στον ιμάντα, τότε παίρνοντας σαν είσοδο το είδος του αντικειμένου ο αλγόριθμος ορίζει που θα τοποθετηθεί το αντικείμενο. Για παράδειγμα, εάν το αντικείμενο είναι κύβος τότε θα τοποθετηθεί σε διαφορετικό σημείο από τα άλλα αντικείμενα με σκοπό να ξεχωρίσουμε όλους τους κύβους. [Παράρτημα Δ – Κώδικας Sorting]

```
Sorting(){
    initialize_sorting()
    while(i < 2){
        set home_point
        set place_point
        set grap_point
        set sunction_start
        move grap_point
        sunction_start()

        if(object == cube){
            move place_point
            set sunction_stop()
        }else{
            move place_point
            set sunction_stop()
        }

        move home_point
    }
}
```

**Εικόνα 5.3.4 Ψευδοκώδικας Sorting**

Τέλος έχουμε τις δύο δουλειές που κάνουν τα νήματα που δημιουργούμε. Η πρώτη είναι το items_job η οποία ελέγχεται από την PickandPlace και αναλαμβάνει την μεταφορά των αντικειμένων από ένα σταθερό σημείο σε άλλο. Η δεύτερη δουλειά που κάνουν τα νήματα είναι το belt_job το οποίο ελέγχει την λειτουργία του ιμάντα αντικειμένων και των παραμέτρων του όπως απόσταση κτλ. Το belt_job ενώνεται με τους δύο βραχίονες διότι οι βραχίονες είναι αυτοί που ελέγχουν τον κάθε ιμάντα αντικειμένων και μέσω του βραχίονα προωθούνται οι εντολές για την κίνηση του κάθε ιμάντα. [Παράρτημα Ε – Κώδικας νημάτων]

```
belt_job(){
    set conveyor_belt_velocity
    set right_motor
    set left_motor

    if(side == right){
        start right_motor
        set change_side
    }else{
        start left_motor
        set change_side
    }
}
```

**Εικόνα 5.3.5 Ψευδοκώδικας belt job**

```
items_job(){
    set home_point
    set grap_points
    set place_points
    move grap_points
    set sunction_start
    if(side == right){
        move place_points
        set change_side
    }else{
        move place_point
        set change_side
    }

    move home_point
}
```

**Εικόνα 5.3.6 Ψευδοκώδικας items job**

### 5.4 Προβλήματα και επεξήγηση

Αντιμετωπίστηκαν αρκετά προβλήματα κατά την ανάπτυξη του προγράμματος. Το πρώτο πρόβλημα που παρατηρήθηκε ήταν η σύνδεση των βραχιόνων. Ο λόγος που στην αρχή δημιουργήθηκαν δύο διαφορετικά προγράμματα ήταν επειδή η ταυτόχρονη σύνδεση ήταν αδύνατη στους βραχίονες. Για παράδειγμα, όταν ενωνόμασταν στον πρώτο βραχίονα δεν μπορούσαμε να τρέξουμε καμία εντολή από τον δεύτερο βραχίονα, διότι απαιτούσε αποσύνδεση και επανασύνδεση με τον άλλο βραχίονα. Το πρόβλημα αυτό αντιμετωπίστηκε με την δημιουργία της μεθόδου ConnectDobotX() η οποία επέτρεπε την σύνδεση πολλών συσκευών και την ταυτόχρονη χρήση εντολών του API από αυτές. Άλλο πρόβλημα που είχαμε ήταν στην εκπαίδευση των καμερών JeVois οι οποίες έχουν ένα ιδιαίτερο τρόπο εκπαίδευσης για να καταλαβαίνουν τις εικόνες των αντικειμένων που τους παρουσιάζουμε. Χρειάστηκε να εμβαθύνουμε αρκετά και να αλλάξουμε διάφορα πράγματα στα configuration files των καμερών για να μας παράγουν το αποτέλεσμα που απαιτούσαμε. Επίσης ένα σημαντικό πρόβλημα που είχαμε ήταν ότι οι κάμερες JeVois επηρεάζονται πάρα πολύ από την φωτεινότητα του δωματίου, πράγμα που μπορεί να αντιμετωπιστεί μόνο αν τοποθετήσουμε τις κάμερες σε σημεία με καλό φωτισμό. Εκτός από την φωτεινότητα οι κάμερες δεν έστελναν με επιτυχία τα 2D standardized messages. Χρειάστηκε αρκετή μελέτη για να ανακαλύψουμε ότι ο τρόπος επιλογής του module που τρέχει είχε λάθος αρχικοποιήσεις και έγιναν αλλαγές στα configuration files για την αντιμετώπιση του προβλήματος. Σχετικά με την παραλληλοποίηση και τον πολυνηματισμό που χρησιμοποιήσαμε στην εργασία, αντιμετωπίσαμε αρκετά προβλήματα κυρίως συγχρονισμού των δύο βραχιόνων τα οποία διαχειριστήκαμε με ελέγχους και τεχνητές καθυστερήσεις(sleep). Μια δυσκολία που έγινε αντιληπτή μετά την ανάπτυξη των νημάτων του προγράμματος ήταν η απόσταση στην μεταφορά αντικειμένων από τους ιμάντες. Αρχικά χρησιμοποιούσαμε την εντολή dType.SetEMotorEx() η οποία ξεκινούσε το μοτοράκι του ιμάντα και έπρεπε να ξανακαλεστεί για να σταματήσει να κινείται ο ιμάντας. Όταν προσπαθούσαμε να ορίσουμε την απόσταση μέσω του χρόνου που θα λειτουργούσε το μοτοράκι δεν παίρναμε σταθερά αποτελέσματα. Όταν αλλάξαμε την μέθοδο dType.SetEMotorEx() και χρησιμοποιήσαμε dType.SetEMotorSEx() σταματήσαμε να έχουμε αυτό το πρόβλημα διότι η dType.SetEMotorSEx() δέχεται παράμετρο απόστασης και αντί να λειτουργεί σαν on/off όπως η άλλη, λειτουργεί για μια προκαθορισμένη απόσταση. Διάφορα προβλήματα σαν αυτό δημιουργούνταν κατά την ανάπτυξη του προγράμματος.

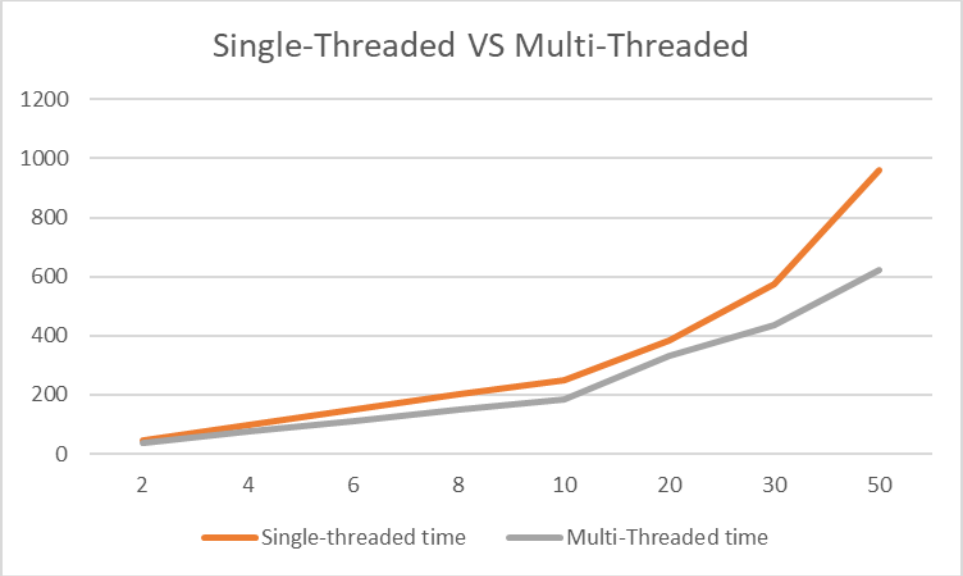
### 5.5 Αποτελέσματα και μετρήσεις

Το πρόγραμμα μετρήθηκε σε διάφορα στάδια για να προχωρήσουμε στην τελική βελτιστοποιημένη υλοποίηση του. Στο πρώτο στάδιο είχαμε δύο ξεχωριστά προγράμματα τα οποία έτρεχαν τους αλγορίθμους pickandplace και sorting το καθένα αντίστοιχα. Σε δεύτερο στάδιο έγινε η ένωση των προγραμμάτων των οποίων έτρεχαν με 1 νήμα. Σε αυτό το στάδιο παρατηρήθηκε η ανάγκη της αξιοποίησης του πολυνηματισμού που προσφέρει η Python[13] για την βελτίωση της ταχύτητας εκτέλεσης των δύο αλγορίθμων και κατ' επέκταση του προγράμματος. Στο τρίτο στάδιο ακολούθησε η δημιουργία νημάτων και η ανάθεση τους σε δουλειές. Έγιναν μετρήσεις στο πρόγραμμα στο δεύτερο και τρίτο στάδιο ανάπτυξης και παρατηρήθηκαν σημαντικά πράγματα.

Σχετικά με τις μετρήσεις για την Multi-Threaded εκτέλεση του προγράμματος, με την πρώτη ματιά παρατηρήσαμε ότι ο χρόνος εκτέλεσης του προγράμματος αυξάνεται σταθερά με την αύξηση των αντικειμένων. Αυτό γίνεται διότι απαιτείται συγχρονισμός στην εκτέλεση και δεν μπορούμε να κάνουμε τέλεια παραλληλοποίηση του προγράμματος γιατί μετά δεν θα προλαβαίνουν οι βραχίονες να κάνουν τις απαραίτητες ενέργειες. Η σταθερή αύξηση είναι ένδειξη που δεν μας χαροποιεί και ιδιαίτερα αφού περιμέναμε να βλέπουμε βελτίωση στον χρόνο για πολλά αντικείμενα. Δηλαδή, θα έπρεπε να παρατηρούμε ότι για περισσότερα αντικείμενα το πρόγραμμα μας είναι πιο αποδοτικό παρά για λιγότερα. Το συμπέρασμα αυτό μπορεί να ισχύει για αριθμούς αντικειμένων μεγαλύτερους του 100-200.


**Εικόνα 5.5.1 Γραφική παράσταση multi-threaded program execution times (x-items , y-seconds)**

Σε αυτή την γραφική 5.5.2 βλέπουμε ξεκάθαρα πως αυτά που υποστηρίξαμε πιο πάνω ισχύουν. Στα πολλά αντικείμενα η απόδοση του προγράμματος είναι καλύτερη σε αναλογία με τα λίγα αντικείμενα. Εδώ φαίνεται το τεράστιο πλεονέκτημα της παραλληλίας και του πολυνηματισμού που χρησιμοποιήθηκε για την υλοποίηση του προγράμματος. Το πρόγραμμα για 2 αντικείμενα χρειάζεται 38.3 δευτερόλεπτα για να εκτελεστή, πράγμα που σημαίνει πως αν συνεχίσουμε να υπολογίσουμε τον χρόνο με την πληροφορία αυτή σαν δεδομένο τότε καταλήγουμε στο ότι τα 20 αντικείμενα θα έπρεπε να έχουν χρόνο περίπου 383 δευτερόλεπτα , τα 30 αντικείμενα χρόνο περίπου 575 δευτερόλεπτα και τα 50 αντικείμενα περίπου χρόνο 959 δευτερόλεπτα. Με τις μετρήσεις που έγιναν βλέπουμε ότι αυτό δεν ισχύει διότι έχουμε χρόνους 331, 434 και 621 δευτερόλεπτα αντίστοιχα για 20,30 και 50 αντικείμενα.




**Εικόνα 5.5.2 Γραφική παράσταση Steady time VS Our time increase**

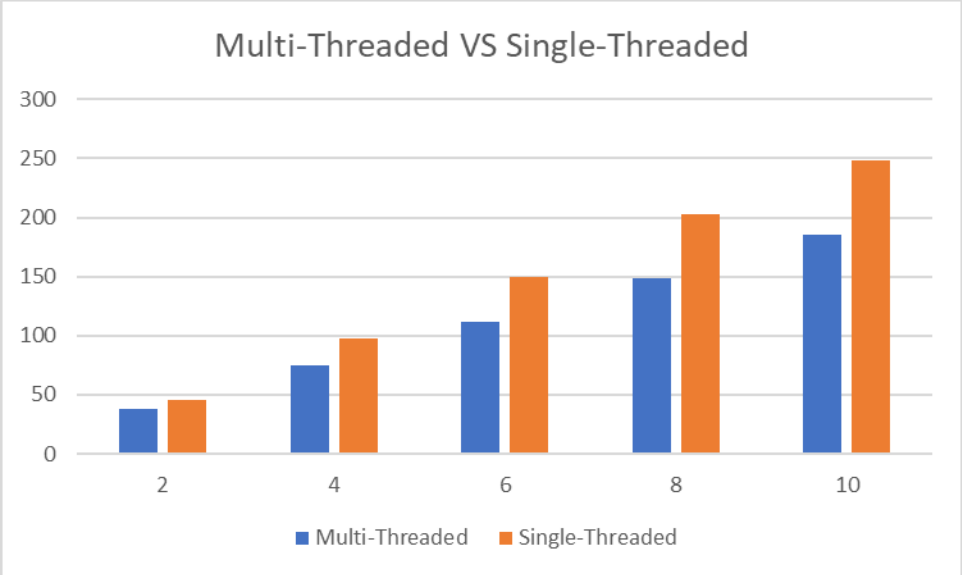
Για την γραφική παράσταση που παριστάνει η εικόνα 5.5.3 εκτελέστηκε το πρόγραμμα τρέχοντας αρχικά με 1 νήμα όλες τις λειτουργίες και στη συνέχεια με πολλά νήματα. Παρατηρείται μεγάλη διαφορά στον χρόνο εκτέλεσης του προγράμματος, παρόλο που το τρέξαμε μόνο μέχρι 10 αντικείμενα. Επίσης φαίνεται πως η αύξηση της διαφοράς του χρόνου είναι ανάλογη με την αύξηση των αντικειμένων, πράγμα που δείχνει την σημασία της παραλληλοποίηση στο multi-threaded πρόγραμμα. Επιπλέον κάτι άλλο που παρατηρήθηκε κατά την εκτέλεση των δύο προγραμμάτων είναι οι μεγάλοι νεκροί χρόνοι της τάξης των 3-5 δευτερολέπτων που υπάρχουν στο single-threaded πρόγραμμα και οι ανεκμετάλλευτοι πόροι. Με την παραλληλοποίηση εκμεταλλεύονται πολύ καλύτερα οι πόροι που έχουμε στην διάθεση μας, με αποτέλεσμα να μην υπάρχουν έντονα μεγάλοι νεκροί χρόνοι και το

πρόγραμμα να τρέχει πολύ πιο γρήγορα. Αυτά γίνονται μέσω της python και του module multithreading το οποίο εκμεταλλευόμαστε[12]. Πιο συγκεκριμένα η παραλληλοποίηση φαίνεται στο παράρτημα Β-2 στον αλγόριθμο PickandPlace[15], μέσω του οποίου δημιουργούνται τα νήματα που τρέχουν τις δουλειές του προγράμματος με τις εντολές :

t1 = Thread(target = items_job, args=(dobot1,i,side,))

t2 = Thread(target = belt_job, args=(belt,distance,dobot0,dobot1,))

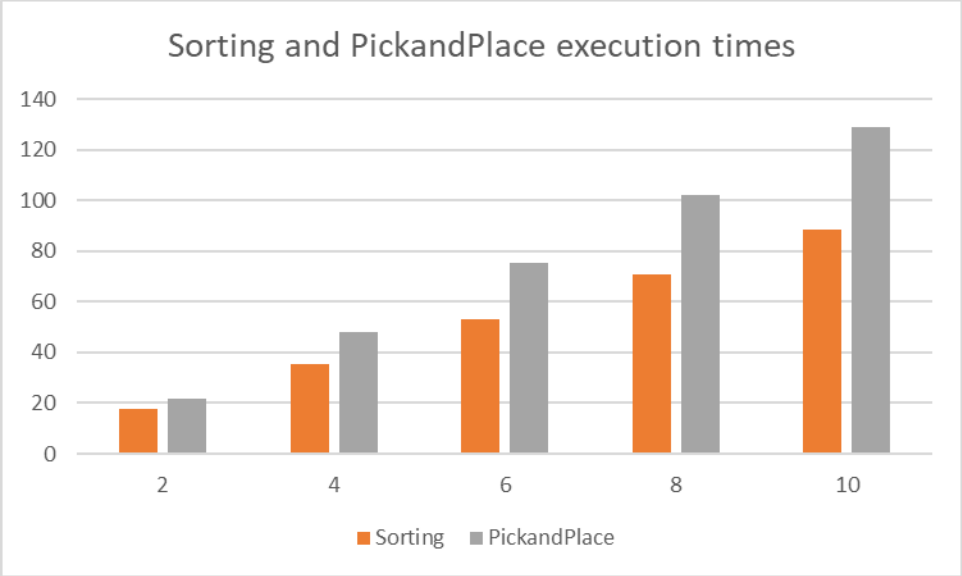
Επίσης στο τέλος κάθε νήματος για λόγους συγχρονισμού κάνουμε join τα νήματα, δηλαδή περιμένουμε να τελειώσουν και τα δύο την δουλειά τους για να αρχίσουν την επόμενη. Θα μπορούσαμε να μην κάνουμε join τα νήματα και να συνεχίζαμε να τα τρέχουμε, αλλά θα απαιτούνταν περεταίρω έλεγχοι συγχρονισμού του προγράμματος, διότι αυξάνονται οι πιθανότητες να γίνουν λάθη. Αν φύγουμε τα join και γίνει διαχείριση των προβλημάτων που θα προκύψουν σίγουρα το πρόγραμμα μπορεί να γίνει ακόμα πιο αποδοτικό.




**Εικόνα 5.5.3 Γραφική παράσταση Multi-threade VS Single-Threaded execution times**

Έγιναν μετρήσεις στον χρόνο εκτέλεσης του κάθε αλγορίθμου. Σημαντικό να σημειώσουμε ότι κατά την διάρκεια των μετρήσεων δεν λήφθηκαν υπόψη οι χρόνοι που χρειάζονται οι κάμερες JeVois για την αναγνώριση του αντικειμένου και ενημέρωση του προγράμματος. Από την γραφική παράσταση στην εικόνα 5.5.4 παρατηρούμε ότι οι χρόνοι εκτέλεσης του αλγορίθμου PickandPlace είναι σημαντικά μεγαλύτεροι από τους χρόνους εκτέλεσης του αλγορίθμου Sorting. Είναι πολύ σημαντικό να το επιβεβαιώσουμε, γιατί με αυτό τον τρόπο καταλαβαίνουμε που χρειάζεται περισσότερο χρόνο το πρόγραμμα μας και που καταναλώνει τους περισσότερους πόρους. Η παρατήρηση μας βοηθά να στρέψουμε το βλέμμα μας εκεί που χρειάζεται για να βελτιώσουμε το πρόγραμμα. Στην περίπτωση μας για περαιτέρω

βελτίωση του ολικού χρόνου εκτέλεσης του προγράμματος μας είναι ορθόδοξο να ασχοληθούμε με την βελτίωση του αλγορίθμου PickandPlace και τον τρόπο που μπορούμε να τον βελτιώσουμε, διότι θα έχει περισσότερο αντίκτυπο στον ολικό χρόνο παρά η βελτίωση του αλγορίθμου Sorting.




**Εικόνα 5.5.4 Γραφική παράσταση Sorting and PickandPlace algorithm execution times**

# Κεφάλαιο 6

## Συμπεράσματα και Μελλοντική εργασία



### 6.1 Συμπεράσματα

Με την ολοκλήρωση της διπλωματικής εργασίας έχουμε καταφέρει να δημιουργήσουμε ένα πραγματικό σενάριο μεταφοράς αντικειμένων από τον τομέα της βιομηχανίας, το οποίο δείχνει πόσο εύκολη έγινε η ζωή μας με την ανάπτυξη της τεχνολογίας. Αντιληφθήκαμε τον ρόλο που παίζουν οι ρομποτικοί βραχίονες και την ταχύτητα που δίνουν στην διεκπεραίωση εργασιών. Οι ρομποτικοί βραχίονες έχουν γίνει ένα αναπόσπαστο κομμάτι της βιομηχανίας και της ανάπτυξης της. Μέσω της εργασίας καταλάβαμε ότι δεν αρκεί να έχουμε τα απαραίτητα εξαρτήματα για να υλοποιήσουμε μια εργασία για να είναι αποδοτική, αλλά και ο τρόπος που θα τα αξιοποιήσουμε αυτά τα εξαρτήματα ανεξάρτητα είδους. Στο σενάριο μας κάναμε διάφορες μετρήσεις και παρατηρήσαμε πολλά ενδιαφέροντα πράγματα. Ένα από αυτά είναι ότι πάντα μπορείς, έστω και λίγο να βελτιώσεις τον τρόπο εργασίας σου. Είτε αυτό είναι να δημιουργήσεις περισσότερα νήματα και να βελτιώσεις την παραλληλία στο πρόγραμμα, είτε να προσθέσεις περισσότερους βραχίονες για να μεταφέρουν πιο πολλά αντικείμενα μαζί. Οποιαδήποτε ανάπτυξη συστήματος γίνεται είναι αναμενόμενο να παρουσιαστούν προβλήματα, αυτό είναι κάτι που δεν μπορούμε ποτέ να το αποφύγουμε. Όμως τα προβλήματα αντιμετωπίζονται με τον ένα ή τον άλλο τρόπο. Στην συγκεκριμένη διπλωματική εργασία αντιμετωπίστηκαν αρκετά προβλήματα τα οποία συζητούμε στο κεφάλαιο 5.4 με περισσότερη λεπτομέρεια και το πως το καθένα από αυτά αντιμετωπίστηκε. Τα προβλήματα τα οποία δεν καταφέραμε να αντιμετωπίσουμε είναι αυτά που μας έδειξαν τα περιθώρια βελτίωσης και ανάπτυξης του προγράμματος μας. Εν τέλει, ο βραχίονας που χρησιμοποιήθηκε εκπλήρωσε τον σκοπό δημιουργίας του που ήταν η εκπαίδευση και ελπίζω να μπορέσει και η εργασία μου να εκπαιδεύσει κάποιον άλλο στο μέλλον.

### 6.2 Μελλοντική εργασία

Το πρόγραμμα που γράφτηκε στα πλαίσια αυτής της διπλωματικής εργασίας προσπάθησε να θίξει ένα από τα πολλά σενάρια τα οποία μπορεί να υπάρχουν στον τομέα της βιομηχανίας με ρομποτικούς βραχίονες. Ένας από τους λόγους που το πρόγραμμα τοποθετήθηκε σε docker container είναι για να μπορεί οποιοσδήποτε στο μέλλον να το δανειστεί και να το επεξεργαστή με οποιοδήποτε τρόπο επιθυμεί. Σχετικά με τις βελτιώσεις που θα μπορούσαν να γίνουν στο πρόγραμμα, θα μπορούσε ο μελλοντικός χρήστης να ψάξει και να βρει έξυπνες κάμερες οι οποίες θα λειτουργούσαν με οποιαδήποτε φωτεινότητα έχει ο χώρος ή θα μπορούσαν να τρέξουν πολλαπλά machine learning modules για αναγνώριση χρώματος, απόστασης και άλλα εκτός από αναγνώριση αντικειμένων. Επίσης, μπορεί να υπάρχουν κάμερες οι οποίες έχουν δικό τους API, με το οποίο θα μπορούσε κάποιος να τις προγραμματίζει δυναμικά αλλάζοντας παραμέτρους καθώς τρέχουν το πρόγραμμα. Μια καλή ιδέα θα ήταν να προστεθούν αισθητήρες απόστασης ή χρώματος που θα πρόσθεταν άλλες λειτουργίες στο σύστημα. Εδώ αξίζει να αναφερθεί ότι αισθητήρας χρώματος έρχεται μαζί με το Dobot Magician απλά είναι προβληματικός.

Σχετικά με ανάπτυξη του υλοποιημένου προγράμματος στο μέλλον θα μπορούσε κάποιος να προσθέσει μετακίνηση των αντικειμένων εκτός του κλειστού συστήματος των δύο βραχιόνων και των ιμάντων αντικειμένων, δηλαδή να προστεθεί ένα κινούμενο ρομπότ το οποίο θα μεταφέρει τα αντικείμενα σε διαφορετικό χώρο. Αυτό θα ήταν και πιο ρεαλιστικό σενάριο που αντιπροσωπεύει και τον πραγματική βιομηχανία.

Σε σχέση με τον βραχίονα θα μπορούσε κάποιος να εκμεταλλευτεί και κάποιες από της υπόλοιπες λειτουργίες του που δεν αγγίξαμε στα πλαίσια αυτής της διπλωματικής. Για παράδειγμα θα μπορούσε να χρησιμοποιήσει την χάραξη laser σε διάφορες επιφάνειες και να δημιουργήσει κάποιον αυτοματοποιημένο τρόπο δημιουργίας πλακών με laser engraving.

# Βιβλιογραφία


[1] Avanse Education Loan. 2021. robotics and industrial automation. [online] Available at: https://www.avanse.com/blog/robotics-industrial-automation-good-career-path/#:~:text=Robotics%20%26%20industrial%20automation%20refers%20to,%2C%20speed%2C%20and%20overall%20performance

[2] Bacharach. n.d. *Robotics Rehabilitation Therapy for Neurological Patients | Bacharach*. [online] Available at: <https://www.bacharach.org/robotics/>

[3] DOBOT | Robotics Solution Provider for STEAM Education, Industry & Businesses. 2021. *DOBOT*. [online] Available at: <https://www.dobot.cc/>

[4] Dobot.it. 2021. [online] Available at: <http://www.dobot.it/wp-content/uploads/2018/03/dobot-api-en.pdf>

[5] Docker Documentation. 2021. *Install Docker Desktop on Windows*. [online] Available at: <https://docs.docker.com/docker-for-windows/install/>

[6] Docs.microsoft.com. 2021. *Install WSL on Windows 10*. [online] Available at: <https://docs.microsoft.com/en-us/windows/wsl/install-win10>

[7] Docs.python.org. 2021. *Thread-based parallelism*. [online] Available at: <https://docs.python.org/3/library/threading.html>

[8] Generationrobots.com. 2021. [online] Available at: <https://www.generationrobots.com/media/Dobot-Magician-User-Manual-V1.2.4.pdf>

[9] Iordache, A., Inman, B., Oro, D. and Arbezzano, G., 2021. *Containerized Python Development - Part 1 - Docker Blog*. [online] Docker Blog. Available at: <https://www.docker.com/blog/containerized-python-development-part-1/>

[10] Itti, L., 2021. [online] Jevois.org. Available at: <http://jevois.org/moddoc/ObjectDetect/modinfo.html>

[11] Kyriakou, P., 2021. *Docker Hub*. [online] Hub.docker.com. Available at: <https://hub.docker.com/r/pkyria14/diplomatiki>

[12] Martelli, A., 2021. *Threading in Python*. [online] Stack Overflow. Available at: <https://stackoverflow.com/questions/2846653/how-can-i-use-threading-in-python>

[13] Python, R., 2021. *An Intro to Threading in Python*. [online] Realpython.com. Available at: <https://realpython.com/intro-to-python-threading/>

[14] Runnable Docker Guides. 2021. *Dockerize your Python Application*. [online] Available at: <https://runnable.com/docker/python/dockerize-your-python-application>

[15] Tutorialspoint.com. 2021. *Python - Multithreaded Programming* - Tutorialspoint. [online] Available at: <https://www.tutorialspoint.com/python/python_multithreading.htm>

[16] Uk.rs-online.com. 2021. *Robotic Arms | RS Components*. [online] Available at: <https://uk.rs-online.com/web/generalDisplay.html?id=ideas-and-advice/robotic-arms-guide>

# Παράρτημα Α-1

Κώδικας main

```
def main(argv):
    # default values
    numberofitems = 2
    speed = 300
    distance = 8
    print("number of items : ", numberofitems)
    print("speed of robotic arm : " , speed)
    print("distance the item will travel on conveyor belt : ", distance)

    try:
        opts, args = getopt.getopt(argv, "hn:p:s:d:", ["numberofitems=",
"speed=", "distance="])
    except getopt.GetoptError:
        print('control.py -n <numberofitems> -s <speed>')
        sys.exit(2)
    for opt, arg in opts:# runs with default values
        if opt == '-h': # print usage of program
            print('USAGE : control.py -n <numberofitems> -s <speed>')
        elif opt in ("-n", "--numberofitems"):
            if (numberofitems > 0):
                numberofitems = int(arg)
        elif opt in ("-d", "--distance"):
            distance = int(arg)
            if (distance > 10 or distance < 0):
                print("wrong distance, defaulting to 8\n")
                distance = 8
        elif opt in ("-s", "--speed"):
            speed = int(arg)
            if (speed < 321 and speed > 0):
                printf("Outside speed limits, defaulting to 300")
                speed = 300
    print('Number of items is ', numberofitems)
    print('Speed of joints : ', speed)
    print("distance the item will travel on conveyor belt : ", distance)

    CON_STR = {
        dType.DobotConnect.DobotConnect_NoError: "DobotConnect_NoError",
        dType.DobotConnect.DobotConnect_NotFound: "DobotConnect_NotFound",
```

```
    dType.DobotConnect.DobotConnect_Occupied: "DobotConnect_Occupied"
}

# Load Dll and get the CDLL object
dobot0, state1 = dType.ConnectDobotX("COM9") # pick and place
dobot1, state2 = dType.ConnectDobotX("COM3") # sorting

# Connect Dobot
print("Connect status1:", CON_STR[state1[0]])
print("Connect status2:", CON_STR[state2[0]])

# Main program algorithm
PickandPlace(numberofitems,speed,distance,dobot0,dobot1)

# Disconnect All Dobots
dType.DisconnectAll()
```

# Παράρτημα Β-2

Κώδικας PickandPlace

```
def PickandPlace(numberofitems, speed, distance, dobot0, dobot1):

    print('Number of items is ', numberofitems)
    print('Speed of joints : ', speed)
    print('Distance of conveyot belt : ', distance)

    # initialize valus before running
    INITIALIZE_PickandPlace(dobot1, speed)
    print('START PICK&PLACE')
    stop = 0
    exectime = 0 # for time metrics
    side = False
    belt = 1 # which conveyor belt to run each time
    i = 0  # for which cube to pick up

    while stop < numberofitems:

        # create threads[15]
        t1 = Thread(target = items_job, args=(dobot1,i,side,))
        t2 = Thread(target = belt_job, args=(belt,distance,dobot0,dobot1,))

        # start threads
        t1.start()
        time.sleep(10)  # to synchronise the procedure
        t2.start()

        stop = stop + 1 # every 2 run JeVois and Sorting
        i = i + 1
        belt = belt - 1 # change belt
        side = not side # changes side to put one left one right each time

        if(stop > 0 and stop % 2 == 0):
            # Identify object with JeVois Camera
            print("JEVOIS TIME\n")
            # Find first item
            objectid = objectDetection(serdev)
            print("object id : ", objectid)
            # Find second item
```

```
            objectid2 = objectDetection(serdev2)
            print("object id 2 : ", objectid2)

            Sorting(numberofitems, speed,  objectid, objectid2, dobot0,
dobot1)


        t1.join()
        t2.join()
```

# Παράρτημα Γ-3

Κώδικας Object Detection

```
serdev = 'COM13'  # serial device of JeVois 1
serdev2 = 'COM17'  # serial device of Jevois 2

# JeVois Camera Object Detection

def ObjectDetection(serdev):
    with serial.Serial(serdev, 115200, timeout=1) as ser:
        while 1:
            # Read a whole line and strip any trailing line ending
character:
            line = ser.readline().rstrip()
            print("received: {}".format(line))
            # Split the line into tokens:
            tok = line.split()
            # Skip if timeout or malformed line:
            if len(tok) < 1:
                print("len(tok) < 1")
                continue
            # Skip if not a standardized "Normal 2D" message:
            # See http://jevois.org/doc/UserSerialStyle.html
            if tok[0].decode('utf-8') != 'N2':
                print("tok[0] != N2")
                continue
            # From now on, we hence expect: N2 id x y w h
            if len(tok) != 6:
                print("length !=6")
                continue
            # Assign some named Python variables to the tokens:
            key, id, x, y, w, h = tok
            id = (id.decode('utf-8'))
            #print(id)
            return id
```

# Παράρτημα Δ-4

Κώδικας Sorting

```
def Sorting(numberofitems, speed, objectid, objectid2, dobot0, dobot1):
    placement = "right"
    INITIALIZE_Sorting(dobot0, speed)
    print('START SORTING')
    stop = 0
    while stop < 2:
        if (placement == "left"):
            placement = "right"
            # ChangePos(placement)
            # Grap from left
            dType.SetPTPCmdEx(dobot0, 0, Grab_X_L, Grab_Y_L, Grab_Z_L, 0,
1)
            dType.SetEndEffectorSuctionCupEx(dobot0, 1, 1)
        else:
            placement = "left"
            # ChangePos(placement)

            # Grap from right
            dType.SetPTPCmdEx(dobot0, 0, Grab_X_R, Grab_Y_R, Grab_Z_R, 0,
1)
            dType.SetEndEffectorSuctionCupEx(dobot0, 1, 1)
        if (stop == 0 and objectid == "blue.png"):
            print("FIRST CUBE IS BLUE! ", objectid , "stop: ", stop)
            dType.SetPTPCmdEx(dobot0, 0, Place_X_cube, Place_Y_cube,
Place_Z_cube, 0, 1)
        elif(stop == 0 and objectid == "green.png"):
             print("FIRST CUBE IS GREEN! ",objectid, "stop: ", stop)
             dType.SetPTPCmdEx(dobot0, 0, Place_X_else, Place_Y_else,
Place_Z_else, 0, 1)
        elif(stop == 1 and objectid2 == "green.png"):
            print("SECOND CUBE IS GREEN! ", objectid2, "stop: ", stop)
            dType.SetPTPCmdEx(dobot0, 0, Place_X_else, Place_Y_else,
Place_Z_else, 0, 1)
        else:
            print("SECOND CUBE IS BLUE! ", objectid2 , "stop: ", stop)
            dType.SetPTPCmdEx(dobot0, 0, Place_X_cube, Place_Y_cube,
Place_Z_cube, 0, 1)
        stop = stop + 1
```

```
    dType.SetEndEffectorSuctionCupEx(dobot0, 0, 1)

# go to home position after placing item
dType.SetPTPCmdEx(dobot0, 0, HOME_X, HOME_Y, HOME_Z, 0, 1)
# dType.SetPTPCmdEx(dobot1, 0, Place_2X, Place_2Y, Place_2Z, 0, 1)
```

# Παράρτημα Ε-5

Κώδικας νημάτων

```
def items_job():
    print('Items Job')
    if (i == 0):
        dType.SetPTPCmdEx(dobot1, 0, Grab_X, Grab_Y, Grab_Z, 0, 1)
    elif (i == 1):
        dType.SetPTPCmdEx(dobot1, 0, Grab_X1, Grab_Y1, Grab_Z1, 0, 1)
    elif (i == 2):
        dType.SetPTPCmdEx(dobot1, 0, Grab_X2, Grab_Y2, Grab_Z2, 0, 1)
    elif (i == 3):
        dType.SetPTPCmdEx(dobot1, 0, Grab_X3, Grab_Y3, Grab_Z3, 0, 1)

    i = i + 1
    dType.SetEndEffectorSuctionCupEx(dobot1, 1, 1)

    if (side):
        dType.SetPTPCmdEx(dobot1, 0, Place_2X, Place_2Y, Place_2Z, 0, 1)
    else:
        dType.SetPTPCmdEx(dobot1, 0, Place_X, Place_Y, Place_Z, 0, 1)
    dType.SetEndEffectorSuctionCupEx(dobot1, 0, 1)


    # go to home position after placing item
    dType.SetPTPCmdEx(dobot1, 0, HOME_X, HOME_Y, HOME_Z, 0, 1)

def belt_job(belt,):
    print("Belt_job")
    # belt0 = right belt1 = left

    if (belt == 0):
        time_start = dType.gettime()[0]
        STEP_PER_CRICLE = 360.0 / 1.8 * 10.0 * 16.0
        MM_PER_CRICLE = 3.1415926535898 * 36.0
        vel = float(50) * STEP_PER_CRICLE / MM_PER_CRICLE
        dType.SetEMotorEx(dobot1, 0, 1, int(vel), 1)
        while True:
            # velocity of conveyor belt
            if (dType.gettime()[0]) - time_start >= distance:
                STEP_PER_CRICLE = 360.0 / 1.8 * 10.0 * 16.0
                MM_PER_CRICLE = 3.1415926535898 * 36.0
                vel = float(0) * STEP_PER_CRICLE / MM_PER_CRICLE
                dType.SetEMotorEx(dobot1, 0, 0, int(vel), 1)
                belt = belt + 1
                break
    else:
        time_start = dType.gettime()[0]
        STEP_PER_CRICLE = 360.0 / 1.8 * 10.0 * 16.0
        MM_PER_CRICLE = 3.1415926535898 * 36.0
        vel = float(50) * STEP_PER_CRICLE / MM_PER_CRICLE
        dType.SetEMotorEx(dobot0, 0, 1, int(vel), 1)
        while True:
            # velocity of conveyor belt
            if (dType.gettime()[0]) - time_start >= distance:
                STEP_PER_CRICLE = 360.0 / 1.8 * 10.0 * 16.0
                MM_PER_CRICLE = 3.1415926535898 * 36.0
                vel = float(0) * STEP_PER_CRICLE / MM_PER_CRICLE
                dType.SetEMotorEx(dobot0, 0, 0, int(vel), 1)
```

```
            belt = belt - 1
            break
```