# Θεωρητικό Υπόβαθρο

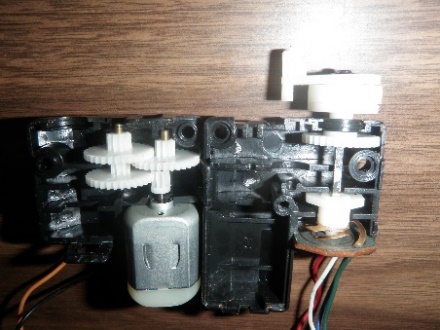
## Μοτέρ Servo

### Αρνητική Ανάδραση

Ένα servo, ή σερβο-μηχανισμός είναι ένα σύστημα το οποίο λειτουργεί με βάση την αρχή της αρνητικής ανάδρασης η οποία προσπαθεί, μέσω κάποιας ενέργειας, να αναγκάζει την έξοδο του συστήματος, να ακολουθεί την είσοδο [1].

### Δομή

To μοτέρ servo, είναι ένα ηλεκτρικό μοτέρ το οποίο στην πιο βασική του μορφή αποτελείται από τα εξής μέρη [2]:



Εικόνα 1‑1 Το εσωτερικό ενός servo motor.

* Γρανάζι κινητήρα
* Κινητήρας DTC
* Κύκλωμα ελέγχου
* Ποτενσιόμετρο
* Άξονας εξόδου

Μπορούμε να συνδέσουμε εξαρτήματα επάνω στον άξονα εξόδου προκειμένου να τα θέσουμε σε περιστροφική κίνηση.

### Λειτουργία

Το κύκλωμα ελέγχου είναι υπεύθυνο για την περιστροφή του κινητήρα DTC μέσω ενός σήματος ελέγχου. Έτσι κάθε σήμα αντιστοιχεί σε μία επιθυμητή γωνιακή μετατόπισητου κινητήρα. Ο κινητήρας με την σειρά του περιστρέφει τον άξονα εξόδου μέσω του γραναζιού με σκοπό να τον περιστρέψει. Όσο μεγαλύτερο είναι το γρανάζι τόσο αυξάνεται η ροπή του τελικού άξονα. Το ποτενσιόμετρο παράγει ένα σήμα ανατροφοδότησης το οποίο αντιπροσωπεύει την πραγματική γωνιακή μετατόπιση του άξονα. Τέλος, συγκρίνοντας την πραγματική με την επιθυμητή γωνιακή μετατόπιση, πραγματοποιείται αρνητική ανάδραση, η οποία αποσβένει το σφάλμα

### Μοντέλο PWM

Μία συχνή μέθοδος ελέγχου των μοτέρ servo είναι η χρήση του μοντέλου PWM (Pulse-Width-Modulation) όπου σύμφωνα με αυτό κάθε χρονικό διάστημα ένας παλμός πλάτους τροφοδοτείται στο μοτέρ [2]. Θα ξεκινήσουμε την πιο αναλυτική περιγραφή του μοντέλου ορίζοντας κάποιες ποσότητες.

Ορίζουμε τον Duty Cycle () από την σχέση:

Η γωνιακή συχνότητα ενός κύκλου PWM ορίζεται ως:

Το servo-μοτέρ είναι σχεδιασμένο έτσι ώστε να υποστηρίζει παλμούς με πλάτος μεταξύ μίας ελάχιστης και μέγιστης τιμής και αντίστοιχα. Αν τροφοδοτηθεί με ένα σήμα το οποίο έχει πλάτος εντός αυτών των ορίων, ο άξονας εξόδου θα αποκτήσει ανάλογη περιστροφή.

Η τιμή του είναι μεταβλητή και μπορεί να χρησιμοποιηθεί για να κατασκευαστεί ένας υποστηριζόμενο παλμός πλάτους ως ο οποίος θα δίνεται από την σχέση:

Το μοτέρ καταλήγει να αποκτά περιστροφή ίση με:

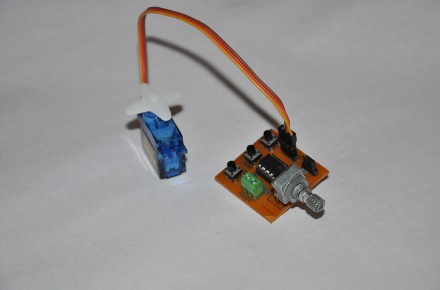
Όπου η ελάχιστη και μέγιστη περιστροφή που υποστηρίζει το servo. Αυτές καθορίζονται από τα και αντίστοιχα.

Συνοπτικά, εάν η επιθυμητή γωνία περιστροφής του servo είναι , τότε πρώτα βρίσκουμε τον D από την σχέση:

και ύστερα υπολογίζουμε το αντίστοιχο πλάτος από την σχέση **(2.1)**. Τέλος τροφοδοτούμε στο μοτέρ ένα σήμα με παλμό πλάτους μέσω ενός μικροελεγκτή PWM.

### Μικροελεγκτής PWM

Οι μικροελεγκτές PWM τροφοδοτούν τα μοτέρ servo με το επιθυμητό σήμα PWM μέσω των καναλιών που διαθέτουν. Κάθε κανάλι αποτελείται από τρεις ακροδέκτες (pins) που βρίσκονται σε σειρά. Οι ακροδέκτες είναι υπεύθυνοι για να παρέχουν στο servo ένα από τα τρία απαραίτητα σήματα:



Εικόνα 1‑2 Ένας μικρός μικροελεγκτής PWM συνδεδεμένος με ένα μοτέρ servo.

1. Τάση λειτουργίας
2. Γείωση
3. Σήμα PWM

Οι μικροελεγκτές συνήθως συνδέονται με έναν υπολογιστή, ο οποίος καθορίζει την μορφή του σήματος PWM και το κανάλι στο οποίο θα σταλεί. Θεωρητικά, ο έλεγχος των servo θα μπορούσε να γίνει απευθείας από τον υπολογιστή. Ωστόσο, οι μικροελεγκτές PWM εμπεριέχουν ρυθμιστικά κυκλώματα τα οποία εξασφαλίζουν πως όλα τα μοτέρ δέχονται την σωστή τάση λειτουργίας και ακόμα παράγουν σήματα με λιγότερο θόρυβο.

## Raspberry Pi

### Τι είναι το Raspberry Pi;

Το raspberry pi, στην πιο απλή του μορφή, είναι ένας υπολογιστής μικρών διαστάσεων [3]. Με την πάροδο του χρόνου, αναπτύσσονται νέα μοντέλα Raspberry Pi, έτσι ώστε να μένουν ενημερωμένα με τις τελευταίες εξελίξεις της τεχνολογίας και του Hardware.



Εικόνα 1‑3 Raspberry Pi 4

Οι κύριες διαφορές μεταξύ του Raspberry Pi και ενός τυπικού, προσωπικού υπολογιστή (PC) εμφανίζονται τόσο στην ισχύ, όσο και στο κόστος. Οι συσκευές Raspberry Pi είναι σημαντικά πιο οικονομικές από ένα PC επειδή βασίζονται σε τεχνολογίες hardware κόστους και έχουν μειωμένη κατανάλωση ενέργειας. Ωστόσο, δεν μπορούν να ανταποκριθούν στις ίδιες απαιτήσεις με έναν κοινό υπολογιστή και έτσι χρησιμοποιούνται σε εφαρμογές όπου δεν απαιτούνται υψηλές επιδόσεις όπως σε μικροελεγκτές, servers, έξυπνες συσκευές, ερασιτεχνικές δραστηριότητες και άλλα.

Το λειτουργικό σύστημα ενός υπολογιστή Raspberry Pi ονομάζεται Raspberry Pi OS και βασίζεται επάνω στο Linux/Debian [3]. Τέλος, έχει προ-εγκατεστημένη την γλώσσα προγραμματισμού “python” την οποία μπορούμε να χρησιμοποιήσουμε για να εγκαταστήσουμε εφαρμογές ή βιβλιοθήκες και να επικοινωνήσουμε με το hardware του του υπολογιστή ή με εξαρτήματα/συσκευές που είναι συνδεδεμένες μαζί του.

### Κεφαλή GPIO

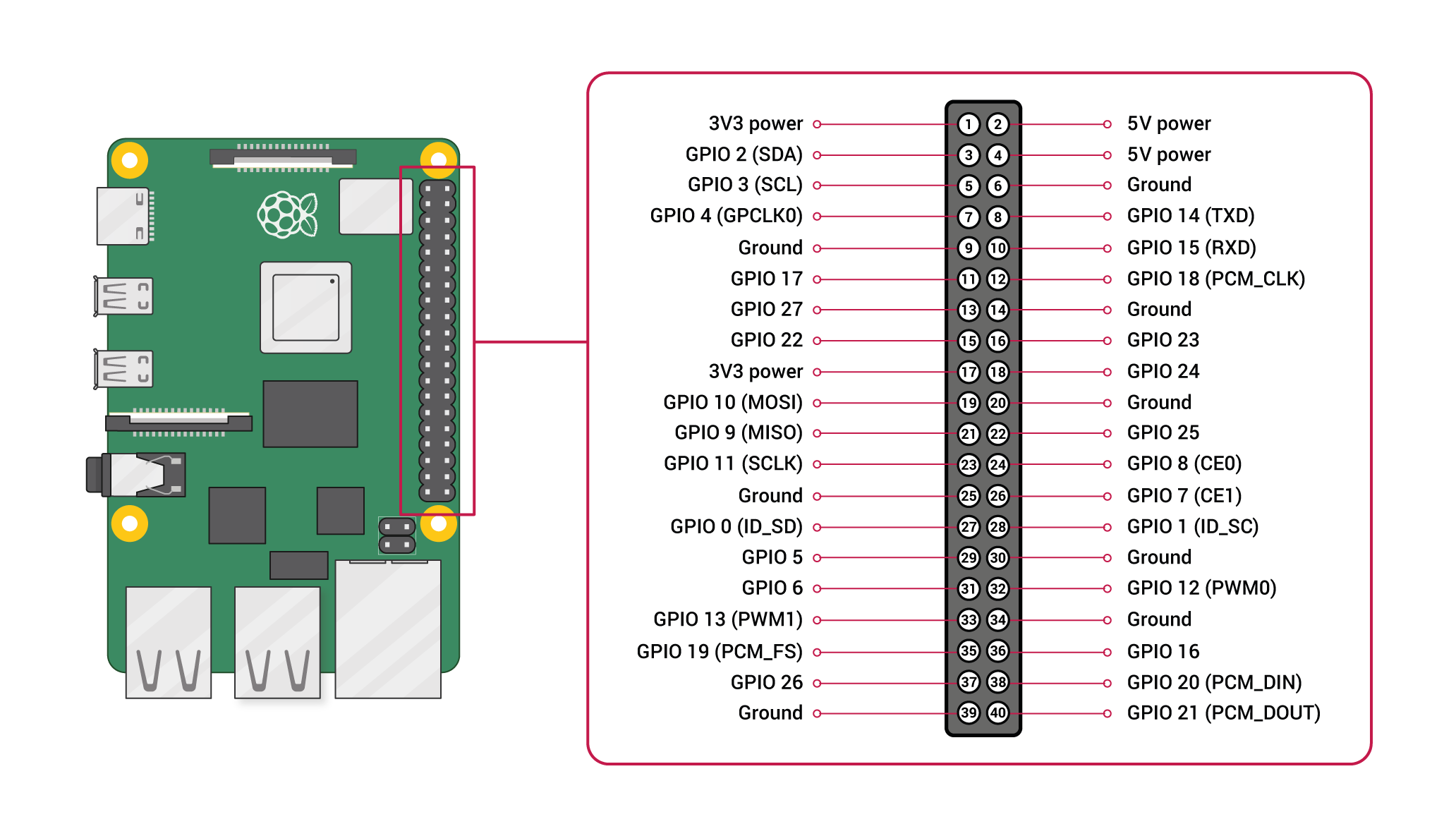
To Raspberry Pi έχει μία κεφαλή GPIO (general-purpose input/output) με 40 ακροδέκτες (pins) οι οποίοι χρησιμοποιούνται για την αποστολή και παραλαβή σημάτων με συσκευές όπως αισθητήρες, μοτέρ, LED και άλλα. Κάθε ακροδέκτης μπορεί να ανήκει σε μία από αυτές τις κατηγορίες:

* 3.3V
* 5V
* Ground
* ID EEPROM
* GPIO
  + General Purpose Input
  + General Purpose Output
  + Ειδικές Επιλογές (Ορίζονται από τον αριθμό του Pin)
    - PWM
    - SPI
    - I2C
    - Serial

Οι κατηγορίες 3.3V, 5V και Ground υπάρχουν απλά για να παρέχουν τροφοδοσία 3.3V, 5V και 0V αντίστοιχα. Ο τύπος ID EEPROM αντιστοιχεί σε δύο pins τα οποία είναι υπεύθυνα για την αναγνώριση εξαρτημάτων ΗAT, αν και δεν είναι πάντα αναγκαία η χρήση τους.

Τέλος οι ακροδέκτες που ανήκουν σε μια από ειδικές επιλογές (PWM, SPI, I2C, SERIAL) χρησιμοποιούνται για την λήψη και αποστολή δεδομένων από και σε συσκευές μέσω των πρωτοκόλλων που αυτές υποστηρίζουν. Τα HATs (hardware attached on top) είναι απλά εξαρτήματα το οποία συνδέονται επάνω στην κεφαλή GPIO, καλύπτοντας κάποιους ή όλους τους ακροδέκτες της.

Το πρωτόκολλο I2C για παράδειγμα χρησιμοποιείται σε αυτή την εργασία για την επικοινωνία μεταξύ του Raspberry Pi και ενός HAT το οποίο λειτουργεί σαν μικροελεγκτής PWM. Για την ανταλλαγή πληροφοριών, υπεύθυνοι είναι οι ακροδέκτες SDA (GPIO 2) και SCL (GPIO 3). Ο ακροδέκτης SDA (Serial Data) μεταφέρει δεδομένα μεταξύ των συσκευών, ενώ ο SCL (Serial Clock) καθορίζει τον χρόνο αποστολής και ανάγνωσης δεδομένων από το Raspberry Pi και τις συνδεδεμένες συσκευές.



Εικόνα 1‑4 Διάγραμμα GPIO header pins για το Raspberry Pi.

## Ρομποτικοί Βραχίονες

### Η εξέλιξη των ρομπότ στην βιομηχανία

Η πρώτη παρέμβαση των ρομπότ στην βιομηχανία έγινε το 1954, όταν δύο μηχανικοί, ο George Devol και ο Joseph Engelberger ίδρυσαν την εταιρία «Unimation» [4]. Το 1961, η Unimation κατασκεύασε αυτό που θεωρείται σήμερα το πρώτο βιομηχανικό ρομπότ με όνομα «Unimate»[5]. To Unimate ήταν ένα ρομπότ υδραυλικής πίεσης, κάτι που το καθιστούσε κατάλληλο για εργασίες με μεγάλο φορτίο. Η επιτυχία του στον τομέα της αυτοκινητοβιομηχανίας οδήγησε αργότερα στην ίδρυση και άλλων εταιριών κατασκευής ρομπότ. Γενικά, Τα ρομπότ που χαρακτήριζαν εκείνη την δεκαετία χρησιμοποιούνταν κυρίως στην επεξεργασία υλικών και τις γραμμές παραγωγής. Ωστόσο, επειδή ο επαναπρογραμματισμός τους ήταν μία αρκετά δύσκολη διαδικασία, η κίνησή τους ήταν συγκεκριμένη και διατελούσαν μία και μόνο εργασία.

Ένα ακόμα μεγάλο ορόσημο για την ρομποτική ήταν η κατασκευή το «Stanford Arm» από τον Victor Scheinman [6]. Το Stanford Arm ήταν το πρώτο ολοκληρωτικά ηλεκτρικά-ελεγχόμενο ρομπότ αφού χρησιμοποιούσε έναν μικροεπεξεργαστή (PDP-6) για την λειτουργία του. Επίσης, ήταν εξοπλισμένο με ταχύμετρα και τενσιόμετρα που καθιστούσαν δυνατή την μέτρηση θέσης και ταχύτητας του, παρέχοντας έτσι μια πιο λεπτομερής και ελεγχόμενη κίνηση. Τα μοντέλα που ακολούθησαν μπορούσαν να αναλάβουν εργασίες οι οποίες απαιτούσαν μια πιο ακριβής κίνηση όπως η ήταν για παράδειγμα η συναρμολόγηση στην αυτοκινητοβιομηχανία.

Τα ρομπότ μεταξύ των χρονολογιών 1980 και 2000 ήταν εξοπλισμένα με servo μοτέρ και έτσι μπορούσαν να εκτελέσουν κινήσεις τόσο από σημείο σε σημείο, όσο και επάνω σε ένα συνεχές μονοπάτι [5]. Ο προγραμματισμός τους γινόταν με την χρήση teach box, PLC, ή υπολογιστή. Συνεπώς δεν περιορίζονταν σε μία συγκεκριμένη κίνηση αφού μπορούσαν να προγραμματιστούν εκ νέου για διαφορετικές λειτουργίες και να χρησιμοποιούν την επεξεργαστική ικανότητα ενός υπολογιστή ώστε να προσαρμόζονται σε περιβαλλοντικούς παράγοντες.

Σήμερα, οι ικανότητες ενός ρομπότ περιορίζονται κυρίως από το λογισμικό του υπολογιστή ο οποίος το ελέγχει. Γενικά, ένα ρομπότ της εποχής μας μπορεί να είναι συνδεδεμένο με διαφορετικούς αισθητήρες και έτσι να έχει επίγνωση του περιβάλλοντός του σε βαθύ επίπεδο. Τα δεδομένα που συλλέγει, με την κατάλληλη επεξεργασία, οδηγούν σε φιλοσοφημένες και εξατομικευμένες για κάθε εφαρμογή κινήσεις.

### Ο ρομποτικός βραχίονας

Πλέον υπάρχουν πολλά διαφορετικά είδη ρομπότ, το κάθε ένα με ικανότητες σχεδιασμένες για να ανταποκρίνονται σε συγκεκριμένες εφαρμογές. Παραδείγματα περιλαμβάνουν τα ανθρωποειδή ρομπότ, τα drones, τους 3D εκτυπωτές και τους ρομποτικούς βραχίονες.

Οι ρομποτικοί βραχίονες έχουν παρόμοια φυσική δομή και κινησιολογία με το ανθρώπινο χέρι, καθιστώντας τους κατάλληλους σε εφαρμογές οι οποίες απαιτούν ακρίβεια και ευελιξία. Όπως και το ανθρώπινο χέρι αποτελούνται από:

* Συνδέσμους (links): Αντιστοιχούν στα οστά του ανθρώπινου χεριού και αποτελούν την φυσική δομή του βραχίονα.
* Αρθρώσεις (joints): Όπως υποδεικνύει το όνομά τους, έχουν ρόλο παρόμοιο με αυτόν των ανθρώπινων αρθρώσεων, δηλαδή να περιστρέφουν τους συνδέσμους.

Ένας ρομποτικός βραχίονας κατηγοριοποιείται με βάση των βαθμών ελευθερίας του (degrees of freedom, DOF). Κάθε ανεξάρτητη άρθρωση η οποία επηρεάζει είτε την περιστροφική ή την γραμμική θέση του [7]. Όσο περισσότεροι είναι οι βαθμοί ελευθερίας του συστήματος, τόσο πιο ευέλικτος είναι ο βραχίονας.

Στην εικόνα 2-5 παρουσιάζεται ένας ρομποτικός βραχίονας τεσσάρων βαθμών ελευθερίας, στον οποίο κάθε άρθρωση περιστρέφεται μέσω σερβοκινητήρα. Οι τρεις πρώτες αρθρώσεις είναι υπεύθυνες για την μετακίνηση της δαγκάνας στο επιθυμητό σημείο στον τρισδιάστατο χώρο, ενώ η τέταρτη άρθρωση ρυθμίζει το άνοιγμα και το κλείσιμο της, απαρτίζοντας συνολικά τέσσερεις βαθμούς ελευθερίας.

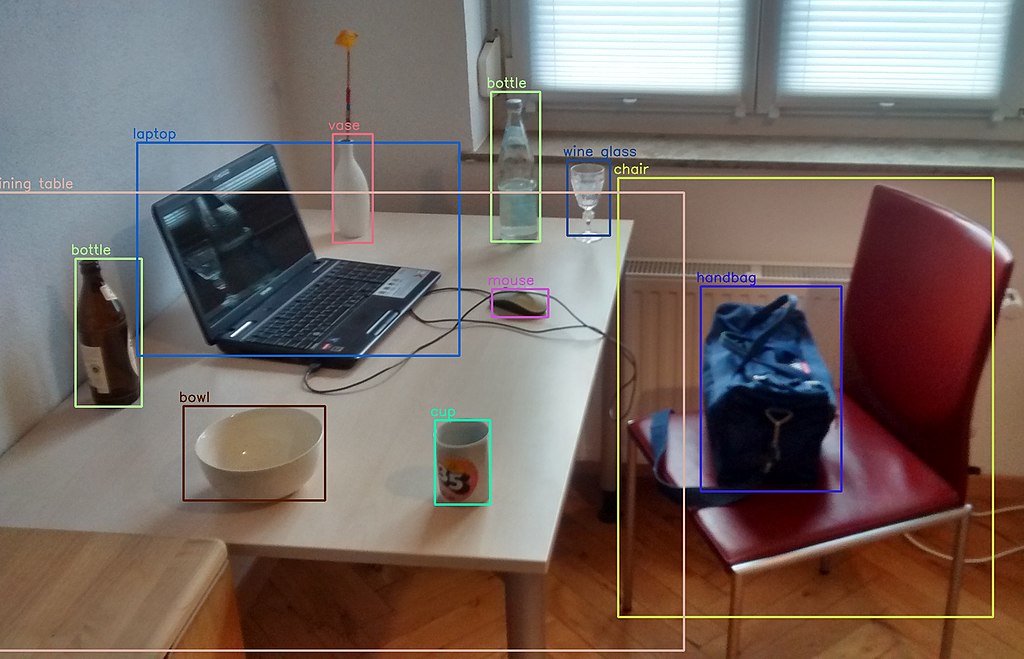


Εικόνα 1‑5 Ρομποτικός βραχίονας "Waveshare Robot Arm for Pi" με 4 βαθμούς ελευθερίας.

## Αναγνώριση αντικειμένων με το μοντέλο Yolo

### Ανίχνευση αντικειμένων σε μία εικόνα

Η ανίχνευση της θέσης και του τύπου ενός αντικειμένου σε μία ψηφιακή εικόνα ή βίντεο γίνεται μέσω ενός μοντέλου μηχανικής μάθησης. Το μοντέλο δέχεται ως είσοδο μία εικόνα και παράγει ως έξοδο το είδος (κλάση), ένα πλαίσιο οριοθέτησης (bounding box) και ένα βαθμό βεβαιότητας (confidence score) για κάθε αντικείμενο που ανιχνεύει. Ο βαθμός βεβαιότητας εκφράζει το πόσο σίγουρο είναι το μοντέλο για τον τύπο του αντικειμένου όσο και για το πόσο σωστά είναι τα όρια του πλαισίου οριοθέτησης. Ένα παράδειγμα πλαισίων οριοθέτησης φαίνεται στην Εικόνα 1‑6.



Εικόνα 1‑6 Ανιχνευμένα αντικείμενα που προέκυψαν από το μοντέλο «Yolo». Μπορούμε να διακρίνουμε το πλαίσιο οριοθέτησης που αντιστοιχεί σε κάθε αντικείμενο.

### Ανίχνευση με YOLO

Το YOLO είναι ένα μοντέλο μηχανικής μάθησης το οποίο αναπτύχθηκε το 2015 από τον Joseph Redmon [8]. Σε αντίθεση με προηγούμενες μεθόδους οι οποίες ελέγχουν μεμονωμένες περιοχές της εικόνας και σε διάφορες διαστάσεις, το YOLO (You Only Look Once) εφαρμόζει ένα νευρωνικό δίκτυο σε όλη την εικόνα, μία φορά και παράγει bounding boxes με ξεχωριστά confidence scores.

Το σύστημα χωρίζει την εικόνα σε ένα πλέγμα διαστάσεων . Για κάθε κελί αυτού του πλέγματος, ένα νευρωνικό δίκτυο παράγει bounding boxes με συγκεκριμένο βαθμό βεβαιότητας. Κάθε bounding box εκφράζεται από ένα διάνυσμα 5 μεταβλητών και μία ακόμα μεταβλητή η οποία αντιστοιχεί στην κλάση του αντικειμένου που ανιχνεύθηκε. Τα αναπαριστούν το κέντρο του αντικειμένου σε σχέση με το κελί, ενώ τα εκφράζουν το πλάτος και το ύψος του σε σχέση με ολόκληρη την εικόνα αντίστοιχα. Τέλος, το εκφράζει τον βαθμό βεβαιότητας της ανίχνευσης.

### Εκπαίδευση του μοντέλου YOLO

Πριν μπορέσουμε να χρησιμοποιήσουμε το YOLO και να ανιχνεύσουμε τα αντικείμενα που μας ενδιαφέρουν σε μία εικόνα, θα πρέπει να το εκπαιδεύσουμε κατάλληλα.

Σε ένα υψηλό επίπεδο, η εκπαίδευση του μοντέλου γίνεται με τα παρακάτω βήματα:

1. Συγκεντρώνουμε φωτογραφικό υλικό από εικόνες οι οποίες εμπεριέχουν αντικείμενα τις ίδιας κατηγορίας (κλάσης) με αυτά που θέλουμε να ανιχνεύσουμε.
2. Σε ένα κατάλληλο λογισμικό σήμανσης δεδομένων (labeling data software), σημειώνουμε τις περιοχές στις οποίες εντοπίζουμε κάθε αντικείμενο επάνω σε όλες τις φωτογραφίες και παράλληλα ορίζουμε την κλάση στην οποία ανήκουν. Η διαδικασία αυτή ονομάζεται labeling process.
3. Από το ίδιο λογισμικό, εξάγουμε τα τις πληροφορίες σχετικά με την σήμανση των φωτογραφιών σε μία μορφή την οποία το μοντέλο μπορεί να αναγνωρίσει (για παράδειγμα .txt για το YOLO). Τα δεδομένα αυτά ονομάζονται label data.
4. Χρησιμοποιούμε έναν αλγόριθμο ο οποίος τροφοδοτεί το μοντέλο με την τοποθεσία (path) των label data και των φωτογραφιών και τελικά ξεκινάει την εκπαίδευσή του μοντέλου.

Μόλις ολοκληρωθεί η εκπαίδευση, παράγεται ένα αρχείο τύπου το οποίο περιέχει όλες τις πληροφορίες που χρειάζεται το μοντέλο για να λειτουργήσει. Η ακρίβεια του μοντέλου καθορίζεται από την ποιότητα της εκπαίδευσης του, έτσι, για να πετύχουμε το καλύτερο δυνατό αποτέλεσμα, φροντίζουμε οι εικόνες που θα χρησιμοποιήσουμε κατά την εκπαίδευση να βρίσκονται υπό πολλές διαφορετικές συνθήκες φωτισμού και τα αντικείμενα να απεικονίζονται με πληθώρα περιστροφών και θέσεων στο χώρο.

# Hardware

Σε αυτό το κεφάλαιο θα παρουσιαστεί η γενική διάταξη και λειτουργία του συστήματος. Θα αναφέρουμε ονομαστικά τον εξοπλισμό από τον οποίο αποτελείται και ύστερα θα γίνει μία λεπτομερείς περιγραφή κάθε στοιχείου αυτού. Τελικά, θα αναφερθούμε στα προγράμματα και τις βιβλιοθήκες που χρησιμοποιήθηκαν, καθώς και οποιοδήποτε κομμάτι κώδικα το οποίο προέρχεται από εξωτερικές πηγές (third party).

## Φυσική Διάταξη

Για την λειτουργία αυτής της εφαρμογής χρησιμοποιήθηκε ο παρακάτω εξοπλισμός:

* Ρομποτικός Βραχίονας “Waveshare Robot Arm for Pi”
* Κάμερα “Logitech C505e HD 720p Webcam”
* Βάση κάμερας σε σχήμα “Π”.
* Βάση τρίλιζας
* Πιόνια τρίλιζας

Στην Εικόνα 2‑1 φαίνεται ο τρόπος που τοποθετούνται τα παραπάνω στοιχεία στον χώρο.

A robot holding a robot arm over a chess board

AI-generated content may be incorrect.

Εικόνα 2‑1 Φυσική διάταξη του συστήματος

Σε μια βάση πάχους τοποθετούνται τα πιόνια και ένα χαρτί Α4 με εκτυπωμένη την τρίλιζα. Ο ρομποτικός βραχίονας τοποθετείται πίσω από τη βάση σε μια σταθερή απόσταση, η οποία είναι ίδια σε κάθε παιχνίδι. Για να εξασφαλίσουμε πως η θέση του βραχίονα σε σχέση με την βάση είναι πάντα η ίδια, ένα σχέδιο σε χαρτί Α4 έχει τοποθετηθεί πίσω από την βάση.

Επάνω από την τρίλιζα τοποθετείται η βάση «Π» της κάμερας, έτσι ώστε να διασφαλίζει ένα καθαρό οπτικό πεδίο του πλέγματος. To Raspberry Pi, λειτουργεί ως ελεγκτής του ρομποτικού βραχίονα και είναι προσαρμοσμένο επάνω του, επομένως, όλο το σύστημα θα πρέπει να βρίσκεται κοντά σε μία οθόνη υπολογιστή και κάποια παροχή ρεύματος.

## Λειτουργία

Σε πρώτη φάση, το σύστημα περνάει από μία διαδικασία αρχικοποίησης, κατά την οποία ανιχνεύεται το πλέγμα της τρίλιζας. Αφού σιγουρευτούμε πως κανένα πιόνι δεν βρίσκεται επάνω σε κελί, εκτελούμε τον κύριο αλγόριθμο (main loop) και το παιχνίδι ξεκινά. Η κάμερα καταγράφει την τρίλιζα και μετά από επεξεργασία προκύπτουν τα όρια του πλέγματος.



Εικόνα 2‑2 Παράδειγμα του πλαισίου ορίων-πλέγματος που δημιουργείται κατά την αρχικοποίηση

Ο ανθρώπινος παίκτης (χρήστης) κάνει την πρώτη κίνηση τοποθετώντας ένα πιόνι με το σύμβολο «X» στο επιθυμητό κελί. Η σειρά του ολοκληρώνεται όταν αυτός πατήσει το κουμπί «Enter» στο πληκτρολόγιο.

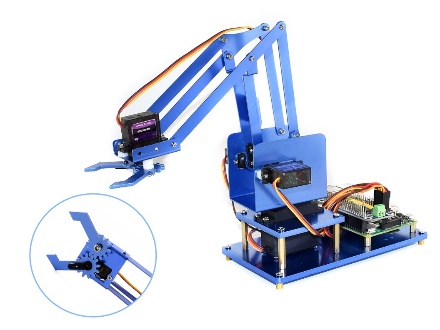
Κάθε φορά που ολοκληρώνεται η σειρά οποιοδήποτε παίκτη (ρομπότ ή άνθρωπος), η κάμερα καταγράφει την τρίλιζα, επεξεργάζεται την εικόνα και μετατρέπει την πληροφορία σε ψηφιακή μορφή, συγκεκριμένα σε μία λίστα python.

Όταν έρθει η σειρά του ρομπότ, αυτό υπολογίζει την καλύτερη κίνηση που μπορεί να κάνει και ο βραχίονας μεταφέρει ένα από τα πιόνια «Ο» στο κατάλληλο κελί. Το παιχνίδι ολοκληρώνεται όταν προκύψει ισοπαλία ή εάν ένας από τους δύο παίκτες κερδίσει.

## Περιγραφή του Hardware

### Ρομποτικός βραχίονας «Waveshare Robot Arm for Pi»

Το μοντέλο του βραχίονα που χρησιμοποιήθηκε ονομάζεται «Waveshare Robot Arm for Pi». Έχει τέσσερις βαθμούς ελευθερίας και τέσσερα μοτέρ σέρβο αντίστοιχα. Τα τρία πρώτα μοτέρ είναι υπεύθυνα για την περιστροφική κίνηση των αρθρώσεων, ενώ το τελευταίο χρησιμοποιείται για τον έλεγχο μίας δαγκάνας που είναι ενσωματωμένη στο άκρο του.



Εικόνα 2‑3 Waveshare Robot Arm for Pi

Το προϊόν διατίθεται ως πακέτο συναρμολόγησης και περιέχει:

* Τη μεταλλική δομή του βραχίονα
* 4 μοτέρ servo
* Έναν μικροελεγκτή PWM σε μορφή HAT (Servo Driver HAT)

Για την λειτουργία του βραχίονα, απαιτείται αρχικά η συναρμολόγηση όλων των εξαρτημάτων. Η σχεδίαση του συστήματος επιτρέπει την εύκολη τοποθέτηση ενός «Raspberry Pi 4 model B» στην ειδική βάση σχηματίζεται. Τρία μοτέρ MG90S χρησιμοποιήθηκαν για τον έλεγχο της δαγκάνας: το ένα ελέγχει την σύσφιξη ενώ τα άλλα δύο ρυθμίζουν το ύψος και το βάθος της θέσης της. Ένα επιπλέον μοτέρ MG996R τοποθετήθηκε στην βάση για τον καθορισμό της αζιμουθιακής γωνίας του βραχίονα.

Ο μικροελεγκτής είναι ενσωματωμένος με 16 κανάλια και έτσι μπορεί να ελέγξει μέχρι και 16 μοτέρ ταυτόχρονα, ενώ παρέχει ανάλυση 4096 βημάτων. Αυτό σημαίνει πως η διαφορά του μικροελεγκτή μπορεί να χωριστεί σε 4096 διακριτά επίπεδα τα οποία αναγνωρίζει ως είσοδο. Τέλος, ο ελεγκτής βασίζεται στο ολοκληρωμένο κύκλωμα PCA9896 για το οποίο υπάρχει σχετική βιβλιοθήκη python.

### Βάση «Π» και Κάμερα

Η κάμερα που χρησιμοποιήθηκε μία απλή webcam υπολογιστή «Logitech C505e HD 720p». Είναι συνδεδεμένη με το Raspberry Pi και στηρίζεται επάνω στην βάση «Π» με τον τρόπο που φαίνεται στην ΧΧ.

H βάση της κάμερας είναι μία διάταξη μεταλλικών ράβδων που σχηματίζει ένα «Π». Είναι σχεδιασμένη ώστε να παραμένει σταθερή, ενώ το ύψος της εξασφαλίζει ένα μεγαλύτερο πεδίο όρασης. Η κάμερα συγκρατείται σε αυτή μέσω του κλίπ που διαθέτει.

### Βάση Τρίλιζας και Πιόνια

Η βάση της τρίλιζας αποτελείται από μία επιφάνεια διογκωμένης πολυστερίνης (φελιζόλ) πάχους τριών εκατοστών, και μία ξύλινη επιφάνεια, πάχους 1.5 εκατοστών, που έχει τοποθετηθεί επάνω της. Τα δύο στοιχεία είναι ενωμένα με ανθεκτική κολλητική ταινία. Το φελιζόλ επιλέχθηκε λόγω της οικονομικής του τιμής και του χαμηλού βάρους του, ενώ η ξύλινη επιφάνεια χρησιμοποιήθηκε κυρίως για αισθητικούς λόγους.

Ένα φύλλο Α4 στο οποίο είναι εκτυπωμένη η διάταξη της τρίλιζας έχει στερεωθεί στην ξύλινη επιφάνεια με την χρήση κόλλας PVA. Το ξύλο συνεργάζεται άριστα με την κόλλα PVA και το χαρτί, γεγονός που αποτέλεσε ακόμα έναν λόγο να επιλεγεί.

Πίσω από την βάση, είναι τοποθετημένο ένα εκτυπωμένο διάγραμμα το οποίο καθορίζει την θέση του ρομποτικού βραχίονα σε σχέση με την βάση της τρίλιζας. Αυτό λειτουργεί σαν οδηγός είναι σημαντικό να τοποθετείται σωστά πριν ξεκινήσει το παιχνίδι.

Τέλος, τα πιόνια είναι κυλινδρικά κομμάτια πλαστικού, ύψους περίπου 2 εκατοστών, στα οποία έχουν προσκολληθεί εκτυπωμένα σύμβολα τρίλιζας, με

κόλλα σιλικόνης. Συνολικά, υπάρχουν πέντε πιόνια με το σύμβολο «Χ» και τέσσερα με το σύμβολο «0».

|  |  |  |
| --- | --- | --- |
| Sforza Castle Milan in autumn | Skyscrapers shown from view looking up | Hairdresser using comb and hair-cutting shears to trim ends of man's hair |
| (a) | (b) | (c) |

Εικόνα 2‑4 (a) … (b) … (c) …

## Κατασκευή Hardware

### Συναρμολόγηση Ρομποτικού Βραχίονα

Η συναρμολόγηση του ρομποτικού βραχίονα έγινε σύμφωνα με τις οδηγίες που αναφέρονται στην ιστοσελίδα του προϊόντος[9]. Τα μοτέρ servo εγκαταστάθηκαν ταυτόχρονα με την συναρμολόγηση του σκελετού ενώ το Raspberry Pi με το Servo Driver HAT τοποθετήθηκε στο τέλος.

Κατά την διαδικασία της χειροκίνητου ελέγχου, ο οποίος εξετάζεται σε επόμενο κεφάλαιο, παρατηρήθηκε συνεχής αποτυχία του μοτέρ της δαγκάνας. Η κίνηση του άξονα εξόδου του σερβοκινητήρα εμποδιζόταν από το να κλείσει εντελώς κατά το γράπωμα ενός πιονιού λόγω της μη παραμορφώσιμης δομής του. Αυτό οδηγούσε σε υπερθέρμανση και τελικά στην καταστροφή του μοτέρ.

Για να αποφευχθεί αυτό, κρίθηκε αναγκαία η τροποποίηση της δαγκάνας. Στην περιοχή όπου το τμήμα της δαγκάνας έρχεται σε επαφή με το πιόνι, τοποθετήθηκε μία λεπτή επένδυση συνθετικού σπόγγου όπως φαίνεται στην ΧΧ. Έτσι, το μοτέρ απαιτεί πλέον σημαντικά λιγότερη ροπή για να παραμορφώσει τον σπόγγο, ο οποίος με την σειρά του ασκεί δύναμη κάθετης αντίδρασης και τριβής στο πιόνι, συγκρατώντας το. Τελικά διαπιστώσαμε πως μία τέτοια αλλαγή ήταν αρκετή ώστε να αποτρέψει την υπερθέρμανση του μοτέρ.



Εικόνα 2‑5 Επένδυση συνθετικού σπόγγου επάνω στην δαγκάνα.

### Κατασκευή Βάσης Κάμερας

Για κατασκευή της βάσης χρησιμοποιήθηκε ηλεκτροσυγκόλληση τόξου. Αρχικά κόψαμε δύο μεταλλικές ράβδους ίδιου ύψους (ΧΧ εκατοστών) και τις τοποθετήσαμε παράλληλα, σε κατακόρυφη θέση. Αυτές λειτούργησαν σαν «πόδια» της βάσης. Τα επάνω άκρα τους ενώθηκαν με μία οριζόντια μεταλλική ράβδο μήκους ΧΧ εκατοστών έτσι ώστε τελικά να προκύψει μία διάταξη με σχήμα «Π».

Για να ισορροπήσει το σύστημα, στο κάτω άκρο των κατακόρυφων ράβδων, συγκολλήσαμε ακόμα μία ράβδο μικρών διαστάσεων, τοποθετημένη κάθετα με το επίπεδο που ορίζουν τα υπόλοιπα στοιχεία. Μία τέτοια προσθήκη αντισταθμίζει τυχόν ροπές οι οποίες θα μπορούσαν να ανατρέψουν την βάση.

A black camera on a white surface

AI-generated content may be incorrect.

Εικόνα 2‑6 Η κάμερα στηρίζεται επάνω στην βάση «Π»

# Software

## Αρχιτεκτονική Προγράμματος και Βιβλιοθήκες

### Βιβλιοθήκες

Οι βιβλιοθήκες που χρησιμοποιήθηκαν στον κώδικα της εφαρμογής από εξωτερικές πηγές είναι οι εξής:

* opencv-python
* pynput
* ultralytics

To opencv-python είναι μία βιβλιοθήκη η οποία χρησιμοποιείται στην επεξεργασία εικόνων. Σε αυτή την εργασία αξιοποιήθηκε με σκοπό την οπτικοποίηση των αποτελεσμάτων του μοντέλου Yolo, προβάλλοντας τις περιοχές στις οποίες εντοπίζεται κάθε αντικείμενο με ένα ορθογώνιο πλαίσιο και αναγράφοντας το confidence score στην κορυφή του. Το πακέτο ultralytics από την άλλη είναι υπεύθυνο για την χρήση του YOLO, τόσο κατά την εκπαίδευση όσο και την εφαρμογή του. Τέλος, έγινε χρήση του pynput έτσι ώστε να ανιχνεύονται οι είσοδοι του πληκτρολογίου από τον αλγόριθμο.

### Αρχιτεκτονική Προγράμματος

Σε αυτή την υποενότητα θα προσπαθήσουμε να περιγράψουμε την γενική δομή του προγράμματος και το πως συνεργάζονται όλα τα αρχεία και κλάσεις του κώδικα μεταξύ τους σε υψηλό επίπεδο.

Διάγραμμα 3‑1 Αρχιτεκτονική προγράμματος

Το πρόγραμμα έχει στηριχτεί στον αντικειμενοστραφή προγραμματισμό και έτσι τα περισσότερα αρχεία περιλαμβάνουν μία κλάση. Όπως φαίνεται στο Διάγραμμα 3‑1, το λογισμικό χωρίζεται σε πέντε πακέτα (modules), τα οποία λειτουργούν σαν ξεχωριστές κατηγορίες οργανώνοντας τον κώδικα με βάση την λειτουργία του.

Το module «computer\_vision» περιέχει τον κώδικα που σχετίζεται με την επεξεργασία εικόνας. Είναι υπεύθυνο τόσο για την καταγραφή φωτογραφιών όσο και για την εξαγωγή δεδομένων από αυτές (π.χ. ανίχνευση αντικειμένων).

Το module «input», όπως υποδηλώνει το όνομά του, διαχειρίζεται τις εισόδους του χρήστη (π.χ. πληκτρολόγιο) και ορίζει ένα σύστημα διαχείρισης γεγονότων (event system) το οποίο θα αναλυθεί σε επόμενο κεφάλαιο.

Ο έλεγχος του ρομποτικού βραχίονα γίνεται μέσω του module «robot\_arm» το οποίο περιέχει τα απαραίτητα scripts για την αποστολή σημάτων PWM στα σερβο-μοτέρ μέσω του μικροελεγκτή PWM.

Στο module «tic\_tac\_toe» περιλαμβάνεται κώδικας ο οποίος σχετίζεται με το παιχνίδι της τρίλιζας. Ένα script-διαχειριστής (game\_manager) έχει πλήρη επίγνωση των μεταβλητών του παιχνιδιού και έτσι καθορίζει την πορεία του, ενώ τα υπόλοιπα scripts διαχειρίζονται τις κινήσεις των παικτών – δηλαδή του ανθρώπου και του ρομπότ.

Τέλος, εκτός όσων αναφέραμε, υπάρχουν δύο αρχεία εκτός module: το και . Το ελέγχει την ροή του προγράμματος και έτσι επιλέγει πότε το παιχνίδι ξεκινά και πότε τελειώνει. Το , χρησιμοποιείται για την μετατροπή δεδομένων από την μία μορφή σε κάποια άλλη όπως η μετατροπή της επιθυμητής γωνίας περιστροφής σε πλάτος παλμού PWM.

## Έλεγχος των Μοτέρ Servo

### Τροφοδοσία παλμού σε κανάλι με το PCA9685.py

Αφού πραγματοποιήθηκε η συναρμολόγηση του ρομποτικού βραχίονα, το επόμενο φυσικό βήμα είναι η ανάπτυξη ενός αλγόριθμου χειροκίνητου ελέγχου.

Ωστόσο, πριν φτάσουμε στον έλεγχο του βραχίονα, θα πρέπει να έχει αναπτυχθεί ένα πρόγραμμα για τον έλεγχο των μοτέρ servo. Για να γίνει αυτό θα εξετάσουμε τον έτοιμο κώδικα ελέγχου του ολοκληρωμένου κυκλώματος PCA9685 ο οποίος παρέχεται στο site του ρομποτικού βραχίονα [9]:

PCA9685.py

1. #!/usr/bin/python

2. import sys

3. import time

4. import math

5. if not 'win' in sys.platform:

6.     import smbus

7.

8. # ============================================================================

9. # Raspi PCA9685 16-Channel PWM Servo Driver

10. # ============================================================================

11.

12. class PCA9685:

13.

14.     # I2C register addresses

15.     \_\_SUBADR1            = 0x02

16.     \_\_SUBADR2            = 0x03

17.     \_\_SUBADR3            = 0x04

18.     \_\_MODE1              = 0x00

19.     \_\_PRESCALE           = 0xFE

20.     \_\_LED0\_ON\_L          = 0x06

21.     \_\_LED0\_ON\_H          = 0x07

22.     \_\_LED0\_OFF\_L         = 0x08

23.     \_\_LED0\_OFF\_H         = 0x09

24.     \_\_ALLLED\_ON\_L        = 0xFA

25.     \_\_ALLLED\_ON\_H        = 0xFB

26.     \_\_ALLLED\_OFF\_L       = 0xFC

27.     \_\_ALLLED\_OFF\_H       = 0xFD

28.

29.     def \_\_init\_\_(self, address=0x40, debug=False):

30.         # Initialize PCA9685 object

31.         self.bus = smbus.SMBus(1)

32.         self.address = address

33.         self.debug = debug

34.

35.         # Reset PCA9685 to initialize

36.         if (self.debug):

37.             print("Reseting PCA9685")

38.         self.write(self.\_\_MODE1, 0x00)

39.

40.     def write(self, reg, value):

41.         # Writes an 8-bit value to the specified register/address

42.         self.bus.write\_byte\_data(self.address, reg, value)

43.         if (self.debug):

44.             print("I2C: Write 0x%02X to register 0x%02X" % (value, reg))

45.

46.     def read(self, reg):

47.         # Read an unsigned byte from the I2C device

48.         result = self.bus.read\_byte\_data(self.address, reg)

49.         if (self.debug):

50.             print("I2C: Device 0x%02X returned 0x%02X from reg 0x%02X" % (self.address, result & 0xFF, reg))

51.         return result

52.

53.     def setPWMFreq(self, freq):

54.         # Sets the PWM frequency

55.         # Calculate prescale value based on desired frequency

56.         prescaleval = 25000000.0    # 25MHz

57.         prescaleval /= 4096.0       # 12-bit

58.         prescaleval /= float(freq)

59.         prescaleval -= 1.0

60.         if (self.debug):

61.             print("Setting PWM frequency to %d Hz" % freq)

62.             print("Estimated pre-scale: %d" % prescaleval)

63.

64.         # Set prescale for the PCA9685

65.         prescale = math.floor(prescaleval + 0.5)

66.         if (self.debug):

67.             print("Final pre-scale: %d" % prescale)

68.

69.         # Save old mode, sleep, set prescale, restore old mode

70.         oldmode = self.read(self.\_\_MODE1)

71.         newmode = (oldmode & 0x7F) | 0x10        # sleep

72.         self.write(self.\_\_MODE1, newmode)        # go to sleep

73.         self.write(self.\_\_PRESCALE, int(math.floor(prescale)))

74.         self.write(self.\_\_MODE1, oldmode)

75.         time.sleep(0.005)

76.         self.write(self.\_\_MODE1, oldmode | 0x80)

77.

78.     def setPWM(self, channel, on, off):

79.         # Sets a single PWM channel

80.         # Directly controls the speed of the servo motor

81.         # Set PWM values for the specified channel

82.         self.write(self.\_\_LED0\_ON\_L + 4 \* channel, on & 0xFF)

83.         self.write(self.\_\_LED0\_ON\_H + 4 \* channel, on >> 8)

84.         self.write(self.\_\_LED0\_OFF\_L + 4 \* channel, off & 0xFF)

85.         self.write(self.\_\_LED0\_OFF\_H + 4 \* channel, off >> 8)

86.         if (self.debug):

87.             print("channel: %d  LED\_ON: %d LED\_OFF: %d" % (channel, on, off))

88.

89.     def setServoPulse(self, channel, pulse):

90.         # Sets the Servo Pulse, assuming a PWM frequency of 50Hz

91.         # Determines the specific position or angle to which the servo motor should move

92.         # Convert pulse duration to PWM value

93.         pulse = pulse \* 4096 / 20000        # PWM frequency is 50Hz, the period is 20000us

94.         self.setPWM(channel, 0, int(pulse))

95.

96. if \_\_name\_\_=='\_\_main\_\_':

97.

98.     # Create PCA9685 object

99.     pwm = PCA9685(0x40, debug=False)

100.

101.     # Set PWM frequency to 50 Hz

102.     # Primarily influences how often the servo controller updates the position

103.     pwm.setPWMFreq(50)

104.

105.     # Continuously sweep the servo back and forth

106.     channel = 15

107.     while True:

108.         for i in range(500, 2500, 10):

109.             pwm.setServoPulse(channel, i)

110.             time.sleep(0.02)

111.

112.         for i in range(2500, 500, -10):

113.             pwm.setServoPulse(channel, i)

114.             time.sleep(0.02)

115.

116.

117. # PULSE TO DEGREES IN "setServoPulse()"

118. # 500  -------   0 °

119. # 1000 -------  45 °

120. # 1500 -------  90 °

121. # 2000 ------- 135 °

122. # 2500 ------- 180 °

123.

124.

Οι μέθοδοι setServoPulse και setPWMFreq είναι αυτές οι οποίες θα φανούν χρήσιμες στον έλεγχο των servo. Ο υπόλοιπος κώδικας έχει να κάνει με την επικοινωνία μεταξύ του Raspberry Pi και του κυκλώματος PCA9685 μέσω της βιβλιοθήκης smbus.

Η μέθοδος setPWMFreq, καθορίζει την συχνότητα PWM έτσι ώστε να είναι ίση την τιμή της παραμέτρου freq. Σε αυτή την εφαρμογή, η συχνότητα που επιλέγουμε είναι αυτή των , η οποία αντιστοιχεί σε συχνότητα ή και θα την θεωρούμε δεδομένη από εδώ και πέρα.

Η μέθοδος setServoPulse τροφοδοτεί το servo στο κανάλι channel με έναν παλμό PWM με πλάτος pulse . To servo driver HAT έχει ανάλυση 4096 (12-bit) και επομένως μπορεί να αναγνωρίζει τιμές από 0 έως 4095. Για την τροφοδοσία του servo, ο κώδικας μετατρέπει πρώτα το πλάτος του παλμού από μs σε επίπεδα PWM. Η αναλογία είναι:

Συνεπώς, εάν τροφοδοτήσουμε ένα servo με παλμό πλάτους pulse , το αντίστοχο επίπεδο PWM δίνεται από την σχέση:

Για παράδειγμα, αν επιθυμούμε να τροφοδοτήσουμε ένα servo με παλμό πλάτους 500μs, η αντίστοιχη τιμή PWM που θα σταλεί στο servo είναι:

Η διαδικασία της μετατροπής γίνεται στην γραμμή του παραπάνω κώδικα.

Τέλος, πρέπει να αναφερθεί, ότι ενώ ο μικροελεγκτής μπορεί να υποστηρίξει επίπεδα PWM από 0 έως 4095, τα servo που χρησιμοποιούνται σε αυτή την εφαρμογή λειτουργούν σε ένα πιο περιορισμένο εύρος παλμών. Συγκεκριμένα υποστηρίζουν παλμούς από έως , που αντιστοιχούν σε επίπεδα PWM από έως . Ένας παλμός περιστρέφει τον άξονα του μοτέρ στους , ενώ ένας παλμός το περιστρέφει στους . Στις γραμμές 117-122 του κώδικα φαίνονται οι αναλογίες μεταξύ μήκους παλμού pulse και της αντίστοιχης γωνίας περιστροφής που θα αποκτήσει το servo για δεδομένη συχνότητα PWM .

### Χειριστής Motor

Για τον εύκολο έλεγχο των servo σε υψηλό επίπεδο και έτσι ώστε να μην χρειάζεται να υπολογίζουμε, ορίζουμε μία νέα κλάση :

motor.py

1. from robot\_arm.PCA9685 import PCA9685

2. import conversions

3. import time

4.

5. class Motor:

6.     def \_\_init\_\_(self, channel, pca):

7.         self.channel = channel

8.         self.pwm = pca

9.         self.rotation = 90

10.         self.angular\_speed = 50

11.         self.set\_rotation(90)

Οι τέσσερις μεταβλητές που ορίζονται παραπάνω είναι:

* self.channel: Το κανάλι στο οποίο ανήκει το μοτέρ
* self.pwm: Ένα στιγμιότυπο της κλάσης PCA9685
* self.rotation: Η τελευταία περιστροφή στην οποία βρισκόταν το μοτέρ
* self.angular\_speed: Η γωνιακή ταχύτητα περιστροφής του μοτέρ

Για να περιστρέψουμε το μοτέρ σε μία γωνία target\_rotation ορίζουμε την μέθοδο:

motor.py

1. def set\_rotation(self, target\_rotation):

2.         if(target\_rotation >= 180): target\_rotation = 180

3.         if(target\_rotation <= 0): target\_rotation = 0

4.

5.         pulse = conversions.angle\_to\_pulse(target\_rotation)

6.         self.pwm.setServoPulse(self.channel, pulse)

7.         self.rotation = target\_rotation

H οποία ελέγχει εάν η επιθυμητή γωνία είναι μεταξύ των ορίων , την μετατρέπει στο αντίστοιχο πλάτος παλμού PWM. Η τιμή της μεταβλητής self.rotation ανανεώνεται αναλόγως.

Ο παλμός ο οποίος θα σταλεί στο μοτέρ μέσω του self.pwm υπολογίζεται μέσω του κώδικα όπως φαίνεται παραπάνω, στην γραμμή 5. Όλο το αρχείο conversions.py αποτελείται από 2 συναρτήσεις: μία για την αντιστοίχιση της επιθυμητής γωνίας περιστροφής σε πλάτος σήματος PWM, και μία για την αντιστοίχιση του πλάτους σε γωνία:

conversions.py

1. def angle\_to\_pulse(angle):

2.     if (angle <= 0): return 500

3.     elif (angle >= 180): return 2500

4.

5.     pulse\_per\_angle = (2500 - 500) / 180

6.     return (500 + pulse\_per\_angle \* angle)

7.

8. def pulse\_to\_angle(pulse):

9.     if(pulse <= 500): return 0

10.     elif (pulse >= 2500): return 180

11.

12.     angle\_per\_pulse = 180 / 2500

13.     return angle\_per\_pulse \* (pulse - 500)

14.

Τέλος, ορίζονται δύο ακόμα συναρτήσεις.

Η initialize\_rotation εκτελείται κατά την δημιουργία ενός αντικειμένου της κλάσης και το περιστρέφει σε γωνία .

H set\_rotation\_smooth μας επιτρέπει να εκτελούμε ομαλή περιστροφή του μοτέρ, εφαρμόζοντάς όλες τις ενδιάμεσες περιστροφές μεταξύ της τελικής και χρονικές σε διακριτά χρονικά διαστήματα:

motor.py

1. def initialize\_rotation(self, initial\_rotation):

2.         self.rotation = initial\_rotation

3.         pulse = conversions.angle\_to\_pulse(self.rotation)

4.         self.pwm.setServoPulse(self.channel, pulse) # degrees per second

5.

6.     def set\_rotation\_smooth(self, target\_rotation):

7.         if(target\_rotation >= 180): target\_rotation = 180

8.         if(target\_rotation <= 0): target\_rotation = 0

9.

10.         delta\_time = 0.02

11.         delta\_step = self.angular\_speed \* delta\_time

12.         if(target\_rotation < self.rotation): delta\_step  \*= -1

13.         number\_of\_steps = abs(int((target\_rotation - self.rotation) / delta\_step))

14.         print(f"Number of steps: {number\_of\_steps}, Target rotation: {target\_rotation}")

15.

16.         for i in range(number\_of\_steps):

17.             self.set\_rotation(self.rotation + delta\_step )

18.             time.sleep(delta\_time)

19.

20.         self.set\_rotation(target\_rotation)

## Έλεγχος του Ρομποτικού Βραχίονα

### Αντίστροφη Κινηματική Βραχίονα

Αντίστροφη κινηματική ονομάζεται η διαδικασία κατά την οποία υπολογίζεται η γωνία περιστροφής που θα πρέπει να έχει κάθε μοτέρ του βραχίονα έτσι ώστε το άκρο του (η δαγκάνα στην περίπτωσή μας) να βρίσκεται σε μία συγκεκριμένη θέση στο χώρο. Ονομάζεται αντίστροφη επειδή ο στόχος της είναι αντίστροφος με αυτόν της ευθείας κινηματικής, δηλαδή ο υπολογισμός του τελικού σημείο για δεδομένες περιστροφές των μοτέρ.

Ακόμα μία διαφορά μεταξύ των δύο είναι ότι για η αντίστροφη κινηματική μπορεί να έχει πολλαπλές ή και καμία λύση για κάποια θέση του άκρου, ενώ στην ευθεία κινηματική, οι διάφορες περιστροφές των μοτέρ θα καταλήγουν πάντα σε μία τελική θέση του άκρου.

Ας ξεκινήσουμε την αντίστροφη κινηματική για τον βραχίονα που χρησιμοποιείται στην παρούσα εφαρμογή. Αρχικά θα ορίσουμε τις γενικευμένες συντεταγμένες με τις οποίες θα ορίζεται η θέση του βραχίονα. Επίσης από εδώ και πέρα, για ευκολία, θα θεωρούμε «θέση του βραχίονα», την θέση του άκρου του, δηλαδή της δαγκάνας.

Παρακάτω παρουσιάζεται η μέθοδος με την οποία υπολογίζουμε την περιστροφή των αρθρώσεων (μοτέρ) των τριών μοτέρ θεωρώντας πως η επιθυμητή τελική θέση είναι αυτή με καρτεσιανές συντεταγμένες .

Το διάνυσμα θέσης του άκρου του βραχίονα μπορεί προσεγγιστεί ως ένα ευθύγραμμο τμήμα που ξεκινά από την αρχή των αξόνων , καταλήγει στην επιθυμητή θέση και έχει μέτρο . Ορίζουμε ως το μήκος της προβολής του θέσης στο επίπεδο **xz,** και , τη γωνία που σχηματίζει η προβολή αυτή με τον άξονα . Το διάνυσμα της θέσης σχηματίζει γωνία με το οριζόντιο επίπεδο . Θεωρούμε ότι ο άξονας έχει φορά από τη βάση προς τα εμπρός, ο άξονας από τη βάση προς τα δεξιά και ο άξονας από τη βάση προς τα επάνω. Οι σχέσεις που συνδέουν όλα τα μεγέθη μεταξύ τους φαίνονται στα παρακάτω σχέδια.

A math equations and formulas

AI-generated content may be incorrect.

Οι γενικευμένες συντεταγμένες με τις οποίες θα περιγράφουμε την θέση του βραχίονα θα είναι:

* Depth (Βάθος):
* Height (Ύψος):
* Rotation (Περιστροφή):

Οι γωνίες των κινητήρων διαφέρουν από τις γωνίες των αρθρώσεων λόγω της γεωμετρίας του ρομποτικού βραχίονα και του τρόπου τοποθέτησης των σερβοκινητήρων. Κάθε μοτέρ ελέγχει μία γωνία που σχετίζεται με τη συγκεκριμένη άρθρωση, αλλά λόγω της κατασκευής του βραχίονα, η γωνία του κινητήρα δεν είναι ακριβώς ίδια με την γωνία της άρθρωσης.

Ορίζουμε ως **base motor** το μοτέρ το οποίο ελέγχει την γωνία , η οποία συνδέεται με την περιστροφή από την σχέση:

Στη συνέχεια θα εξετάσουμε το πως μπορούμε ελέγξουμε το βάθος και το ύψος. Η γωνία θ δεν αντιστοιχεί σε κάποια από τις αρθρώσεις του βραχίονα αλλά είναι μία θεωρητική γωνία. Αυτή εξαρτάται από τις γωνίες και τις οποίες σχηματίζουν δύο σύνδεσμοι ίδιου μήκους του βραχίονα με επίπεδο , όπως φαίνεται στο σχήμα ΧΧ.

A math equations and formulas on a white background

AI-generated content may be incorrect.

Ορίζουμε ως **shoulder motor** το μοτέρ που ελέγχει την γωνία , η οποία καθορίζει την περιστροφή του βραχίονα γύρω από τον άξονα του ώμου. Η γωνία είναι συνδεδεμένη με την γωνία ​ της άρθρωσης του ώμου, και δίνεται από τη σχέση:

Ως **elbow motor** ορίζουμε το μοτέρ που ελέγχει την γωνία , η οποία καθορίζει την κίνηση της άρθρωσης του αγκώνα. Η γωνία σχετίζεται με την γωνία ​ της άρθρωσης του αγκώνα, και δίνεται από τη σχέση:

Καταλήγουμε στο εξής: Εάν θέλουμε να φέρουμε τον βραχίονα στην θέση , τότε θέτουμε τα μοτέρ στις εξής γωνίες:

**Base Motor:**

**Shoulder Motor:**

, όπου

**Elbow Motor:**

, όπου

Για δεδομένο επιθυμητό depth ( και height , οι γωνίες και υπολογίζονται από τις σχέσεις:

, όπου

### Η Κλάση RobotArm

Για τον χειρισμό του βραχίονα, υπεύθυνη είναι η κλάση RobotArm:

robot\_arm.py

1. from robot\_arm.motor import Motor

2. from robot\_arm.PCA9685 import PCA9685

3. import math

4.

5. class RobotArm:

6.     def \_\_init\_\_(self) -> None:

7.         pwm = PCA9685(0x40, debug=False)

8.         pwm.setPWMFreq(50)

9.

10.         self.base\_motor = Motor(0, pwm) # base motor

11.         self.shoulder\_motor = Motor(4, pwm) # shoulder arm motor

12.         self.elbow\_motor = Motor(8, pwm) # elbow arm motor

13.         self.claw\_motor = Motor(12, pwm) # claw motor

14.

15.         self.motors = {

16.             "base": self.base\_motor,

17.             "elbow": self.elbow\_motor,

18.             "shoulder": self.shoulder\_motor,

19.             "claw": self.claw\_motor

20.         }

21.

22.         self.max\_s\_motor\_rotation = 170

23.         self.max\_e\_motor\_rotation = 110

24.         self.min\_e\_motor\_rotation = 10

25.         self.min\_s\_motor\_rotation = 5

26.         self.min\_grip = 8

27.

28.         self.l\_s = 8

29.         self.rotation = 0

30.         self.depth = 8

31.         self.height = 8

32.         self.grip = 90

33.         self.reset()

Kώδικας 3‑1

Αρχικά, δημιουργούμε 4 στιγμιότυπα (instances) της κλάσης Motor και μεταβιβάζουμε στον κατασκευαστή της (constructor), ένα τυχαίο κανάλι από 0 έως 15 και ένα αντικείμενο pwm της κλάσης PCA9685 που έχουμε δημιουργήσει νωρίτερα, αφού έχουμε θέσει την συχνότητα στα 50

Λόγω της κατασκευής και της γεωμετρίας του βραχίονα, οι θέσεις στις οποίες βρίσκονται τα μοτέρ τους επιτρέπουν να περιστρέφονται γύρω από ένα συγκεκριμένο εύρος γωνιών. Εάν δεν επιβάλλουμε αυτή τη συνθήκη μέσω του κώδικα (γραμμές 22-25 του Kώδικας 3‑1) τότε ρισκάρουμε την σύγκρουση των μοτέρ με τον σκελετό του βραχίονα.

Ύστερα ορίζεται η μέθοδος για τον έλεγχο του ύψους του βραχίονα σύμφωνα με την κινηματική της προηγούμενης ενότητας:

robot\_arm.py

1. def set\_height(self, height):

2.         if(self.depth > 15 or self.depth < 4 or height > 15 or height < -5):

3.             return

4.

5.          # try-catch block to handle math domain error

6.         try:

7.             y = height

8.             l\_xz = self.depth

9.             l = math.sqrt(y\*\*2 + l\_xz\*\*2)

10.

11.             theta = math.atan(y/l\_xz)

12.             omega = math.acos((l/2)/self.l\_s)

13.

14.             theta\_1 = theta + omega

15.             theta\_2 = theta - omega

16.         except:

17.             print("It's not possible to reach that height")

18.             return

19.

20.         theta\_e = math.pi/2 + theta\_2

21.         theta\_s = math.pi - theta\_1

22.

23.         theta\_e\_deg = math.degrees(theta\_e)

24.         theta\_s\_deg = math.degrees(theta\_s)

25.

26.         # check if the angles are within the range of motor rotation

27.         if(theta\_e\_deg < self.min\_s\_motor\_rotation or theta\_e\_deg > self.max\_s\_motor\_rotation):

28.             print("Impossible to reach that position")

29.             return

30.         elif(theta\_s\_deg < self.min\_e\_motor\_rotation or theta\_s\_deg > self.max\_e\_motor\_rotation):

31.             print("Impossible to reach that position")

32.             return

33.

34.         self.elbow\_motor.set\_rotation\_smooth(theta\_e\_deg)

35.         self.shoulder\_motor.set\_rotation\_smooth(theta\_s\_deg)

36.

37.         # update height

38.         self.height = height

39.         print(f"Current depth: {self.depth}, height: {self.height}, base rotation: {self.rotation}, grip: {self.grip}")

40.

Εάν η μετακίνηση στο επιθυμητό ύψος ήταν επιτυχής, ανανεώνουμε την μεταβλητή self.height καταλλήλως.

Αντίστοιχα δημιουργούμε μεθόδους για τον έλεγχο των υπόλοιπων γενικευμένων μεταβλητών θέσης, δηλαδή του βάθους, και της περιστροφής:

robot\_arm.py

1. def set\_vertical\_rotation(self, angle):

2.         theta\_b = angle + 90

3.

4.         if not(theta\_b >= 0 and theta\_b <= 180):

5.             return

6.         else:

7.             self.base\_motor.set\_rotation\_smooth(theta\_b)

8.

9.         self.rotation = angle

10.

11.     def set\_depth(self, depth):

12.         if(depth > 15 or depth < 4 or self.height > 15 or self.height < -5):

13.             return

14.

15.         # try-catch block to handle math domain error

16.         try:

17.             y = self.height

18.             l\_xz = depth

19.             l = math.sqrt(y\*\*2 + l\_xz\*\*2)

20.

21.             theta = math.atan(y/l\_xz)

22.             omega = math.acos((l/2)/self.l\_s)

23.             theta\_1 = theta + omega

24.             theta\_2 = theta - omega

25.         except:

26.             print("It's not possible to reach that depth")

27.             return

28.

29.         theta\_s = math.pi - theta\_1

30.         theta\_e = math.pi/2 + theta\_2

31.

32.         theta\_s\_deg = math.degrees(theta\_s)

33.         theta\_e\_deg = math.degrees(theta\_e)

34.

35.

36.         # check if the angles are within the range of motor rotation

37.         if(theta\_s\_deg < self.min\_s\_motor\_rotation or theta\_s\_deg > self.max\_s\_motor\_rotation):

38.             print("Impossible to reach that position")

39.             return

40.         elif(theta\_e\_deg < self.min\_e\_motor\_rotation or theta\_e\_deg > self.max\_e\_motor\_rotation):

41.             print("Impossible to reach that position")

42.             return

43.

44.         self.elbow\_motor.set\_rotation\_smooth(theta\_s\_deg)

45.         self.shoulder\_motor.set\_rotation\_smooth(theta\_e\_deg)

46.

47.

48.         self.depth = depth

49.         print(f"Current depth: {self.depth}, height: {self.height}, base rotation: {self.rotation}, grip: {self.grip}")

50.

Επιπλέον, ορίζουμε κάποιες βοηθητικές μεθόδους για την εύκολη μεταβολή των συντεταγμένων κατά ένα βαθμό:

robot\_arm.py

1. def rotate\_vertically(self, delta\_angle):

2.             self.set\_vertical\_rotation(self.rotation + delta\_angle)

3.

4.     def move\_forwards(self, distance):

5.         self.set\_depth(self.depth + distance)

6.

7.     def move\_upwards(self, distance):

8.         self.set\_height(self.height + distance)

9.

10.     def change\_grip(self, delta\_grip):

11.         self.set\_grip(self.grip + delta\_grip)

### Η Κλάση ArmController

Για να μπορέσουμε να ελέγξουμε τον βραχίονα χειροκίνητα, θα πρέπει να αντιστοιχήσουμε την είσοδο του πληκτρολογίου σε μία από την εντολές κίνησης της κλάσης RobotArm. Η κλάση ArmController είναι υπεύθυνη για την λειτουργία αυτή:

1. from input.input\_manager import InputManager

2. import time

3. from robot\_arm.robot\_arm import RobotArm

4.

5. class ArmController:

6.     def \_\_init\_\_(self):

7.         self.move\_speed = 6  # centimeters per second

8.         self.grip\_speed = 40  # degrees per second

9.         self.rotation\_speed = 100 # degrees per second

10.         self.last\_time = time.time()

11.

12.         print("Welcome to the controller...")

13.         self.input\_manager = InputManager()

14.         self.robot = RobotArm()

15.         self.handle\_input()

16.

17.     def debug\_arm(self, robot):

18.         while True:

19.             pulse = input("Enter angle: ")

20.             pulse = int(pulse)

21.

22.     # Handle the input from the keyboard

23.     def handle\_input(self):

24.         try:

25.             while True:

26.                 current\_time = time.time()

27.                 delta\_time = current\_time - self.last\_time  # Time since last update

28.                 input = self.input\_manager.arrow\_input

29.                 input\_x = input[0]

30.                 input\_y = input[1]

31.

32.                 # Rotate the robot

33.                 if(input\_x != 0):

34.                     val = input\_x \* self.rotation\_speed \* delta\_time

35.                     self.robot.rotate\_vertically(val)

36.

37.                 # Move the robot up or down

38.                 if self.input\_manager.space\_pressed:

39.                     self.robot.move\_upwards(self.move\_speed \* delta\_time)

40.                 elif self.input\_manager.shift\_pressed:

41.                     self.robot.move\_upwards(- self.move\_speed \* delta\_time)

42.

43.                 # Move the robot forwards or backwards

44.                 if(input\_y != 0):

45.                     self.robot.move\_forwards(input\_y \* self.move\_speed \* delta\_time)

46.

47.                 if(self.input\_manager.z\_pressed):

48.                     self.robot.change\_grip(self.grip\_speed \* delta\_time)

49.                 elif(self.input\_manager.x\_pressed):

50.                     self.robot.change\_grip(-self.grip\_speed \* delta\_time)

51.

52.

53.                 time.sleep(0.005)  # Small delay to avoid high CPU usage

54.                 self.last\_time = current\_time  # Update the last\_time

55.

56.         except KeyboardInterrupt:

57.             print("\nController stopped.")

58.

Αρχικά ορίζουμε την ταχύτητα περιστροφής των αρθρώσεων, δημιουργούμε ένα αντικείμενο InputManager και ένα RobotArm, και αποθηκεύουμε την τελευταία στιγμή που ανανεώθηκε η θέση του βραχίονα, στην μεταβλητή self.last\_time. Η λειτουργία του InputManager περιγράφεται στην επόμενη ενότητα.

Η μέθοδος handle\_input είναι ουσιαστικά ένας βρόγχος ο οποίος ελέγχει συνεχώς τον χρόνο που περάσει από την τελευταία ανανέωση. Με βάση αυτό το χρονικό περιθώριο, την ταχύτητα περιστροφής, και την είσοδο του χρήστη από το πληκτρολόγιο περιστρέφει τις αρθρώσεις (μοτέρ) του βραχίονα αναλόγως. Όταν τελειώσει ο βρόγχος, η τιμή της μεταβλητής self.last\_time, ανανεώνεται ξανά.

Στο πληκτρολόγιο, το δεξί και αριστερό βελάκι ελέγχουν την περιστροφή (rotation) του βραχίονα, ενώ τα πλήκτρα x και z ελέγχουν την δαγκάνα. Το επάνω και κάτω βελάκι ελέγχουν το βάθος του (depth) και τέλος, τα πλήκτρα space και shift αυξάνουν ή μειώνουν το ύψος.

## Διαχείριση Γεγονότων και Είσοδος Πληκτρολογίου

### Γεγονότα

Η κλάση Event υλοποιεί ένα προγραμματιστικό μοτίβο γνωστό ως observer pattern (μοτίβο παρατηρητή). Αυτό το μοτίβο, επιτρέπει σε μία ή περισσότερες συναρτήσεις να εγγράφονται σε ένα συγκεκριμένο γεγονός. Όταν το γεγονός αυτό ενεργοποιηθεί, όλες οι εγγεγραμμένες συναρτήσεις θα εκτελεστούν αυτόματα μαζί του.

Αυτή η λειτουργία κάνει τον κώδικα πιο ευέλικτο και αποσυνδεδεμένο, καθώς το μέρος του κώδικα στο οποίο ενεργοποιείται το γεγονός δεν χρειάζεται να γνωρίζει ή να κάνει αναφορά στις συναρτήσεις που χρειάζεται να εκτελεστούν. Την ευθύνη για την εγγραφή των συναρτήσεων έχουν οι κλάσεις που τις ορίζουν.

Event.py

1. class Event:

2.     def \_\_init\_\_(self):

3.         self.\_subscribers = []

4.

5.     def subscribe(self, callback):

6.         if callback not in self.\_subscribers:

7.             self.\_subscribers.append(callback)

8.

9.     def unsubscribe(self, callback):

10.         if callback in self.\_subscribers:

11.             self.\_subscribers.remove(callback)

12.

13.     def invoke(self, \*args, \*\*kwargs):

14.         for callback in self.\_subscribers:

15.             callback(\*args, \*\*kwargs)

16.

17. # Example usage

18. def on\_event\_fired(message):

19.     print(f"Event received with message: {message}")

20.

21. event = Event()

22. event.subscribe(on\_event\_fired)

23. event.invoke("Hello, World!")

Ένα γεγονός είναι απλά ένα αντικείμενο της κλάσης Event, και ενεργοποιείται μέσω της μεθόδου invoke. Για να γίνει εγγραφή ή απεγγραφή μίας συνάρτησης από το γεγονός, χρησιμοποιούμε τις συναρτήσεις subscribe και unsubscribe αντίστοιχα.

### Είσοδος Πληκτρολογίου

Για την αναγνώριση της εισόδου του πληκτρολογίου, χρησιμοποιούμε την βιβλιοθήκη pynput ως εξής:

keyboard.py

1. from pynput import keyboard

2. listener = keyboard.Listener(on\_press=on\_press, on\_release=on\_release)

3. listener.start()

Καταρχάς, εισάγεται το απαραίτητο module από την βιβλιοθήκη. Ύστερα Δημιουργούμε έναν ακροατή ο οποίος θα παρακολουθεί τις εισόδους του πληκτρολογίου. Στον constructor του ακροατή, ορίζουμε τις συναρτήσεις που θα εκτελεστούν όταν πατηθεί (on\_press) ή αφεθεί (on\_release) ένα πλήκτρο.

Έτσι, φροντίζουμε για την δημιουργία και την ενεργοποίηση ένα γεγονός (event) για το πάτημα και την απελευθέρωση κάθε κουμπιού που σχετίζεται με τον βραχίονα:

keyboard.py

1. from input.event import Event

2.

3. on\_left\_arrow\_press = Event()

4. on\_right\_arrow\_press = Event()

5. on\_up\_arrow\_press = Event()

6. on\_down\_arrow\_press = Event()

7. on\_space\_press = Event()

8. on\_shift\_press = Event()

9. on\_z\_press = Event()

10. on\_x\_press = Event()

11.

12. on\_left\_arrow\_release = Event()

13. on\_right\_arrow\_release = Event()

14. on\_up\_arrow\_release = Event()

15. on\_down\_arrow\_release = Event()

16. on\_space\_release = Event()

17. on\_shift\_release = Event()

18. on\_z\_release = Event()

19. on\_x\_release = Event()

20.

21. def on\_press(key):

22.     try:

23.         if key == keyboard.Key.left:

24.             on\_left\_arrow\_press.invoke()

25.         elif key == keyboard.Key.right:

26.             on\_right\_arrow\_press.invoke()

27.         elif key == keyboard.Key.up:

28.             on\_up\_arrow\_press.invoke()

29.         elif key == keyboard.Key.down:

30.             on\_down\_arrow\_press.invoke()

31.         elif key == keyboard.Key.space:

32.             on\_space\_press.invoke()

33.         elif key == keyboard.Key.shift:

34.             on\_shift\_press.invoke()

35.         elif key.char == 'z':

36.             on\_z\_press.invoke()

37.         elif key.char == 'x':

38.             on\_x\_press.invoke()

39.     except:

40.         pass

41. def on\_release(key):

42.     try:

43.         if key == keyboard.Key.left:

44.             on\_left\_arrow\_release.invoke()

45.         elif key == keyboard.Key.right:

46.             on\_right\_arrow\_release.invoke()

47.         elif key == keyboard.Key.up:

48.             on\_up\_arrow\_release.invoke()

49.         elif key == keyboard.Key.down:

50.             on\_down\_arrow\_release.invoke()

51.         elif key == keyboard.Key.space:

52.             on\_space\_release.invoke()

53.         elif key == keyboard.Key.shift:

54.             on\_shift\_release.invoke()

55.         elif key.char == 'z':

56.             on\_z\_release.invoke()

57.         elif key.char == 'x':

58.             on\_x\_release.invoke()

59.     except:

60.         pass

### Διαχειριστής Εισόδων

Μία κλάση InputManager, διαχειρίζεται την είσοδο του πληκτρολογίου και παρέχει έναν πιο εύχρηστο και υψηλού επιπέδου τρόπου μετάδοσης της πληροφορίας στον υπόλοιπο κώδικα.

1. import input.keyboard as keyboard

2.

3. class InputManager:

4.     def \_\_init\_\_(self):

5.         self.arrow\_input = [0, 0]

6.         self.space\_pressed = False

7.         self.shift\_pressed = False

8.         self.z\_pressed = False

9.         self.x\_pressed = False

10.

11.         keyboard.on\_left\_arrow\_press.subscribe(self.on\_left\_arrow\_press)

12.         keyboard.on\_right\_arrow\_press.subscribe(self.on\_right\_arrow\_press)

13.         keyboard.on\_up\_arrow\_press.subscribe(self.on\_up\_arrow\_press)

14.         keyboard.on\_down\_arrow\_press.subscribe(self.on\_down\_arrow\_press)

15.         keyboard.on\_space\_press.subscribe(self.on\_space\_press)

16.         keyboard.on\_shift\_press.subscribe(self.on\_shift\_press)

17.         keyboard.on\_z\_press.subscribe(self.on\_z\_press)

18.         keyboard.on\_x\_press.subscribe(self.on\_x\_press)

19.

20.         keyboard.on\_left\_arrow\_release.subscribe(self.on\_left\_arrow\_release)

21.         keyboard.on\_right\_arrow\_release.subscribe(self.on\_right\_arrow\_release)

22.         keyboard.on\_up\_arrow\_release.subscribe(self.on\_up\_arrow\_release)

23.         keyboard.on\_down\_arrow\_release.subscribe(self.on\_down\_arrow\_release)

24.         keyboard.on\_space\_release.subscribe(self.on\_space\_release)

25.         keyboard.on\_shift\_release.subscribe(self.on\_shift\_release)

26.         keyboard.on\_z\_release.subscribe(self.on\_z\_release)

27.         keyboard.on\_x\_release.subscribe(self.on\_x\_release)

28.

29.     def on\_left\_arrow\_press(self):

30.         self.arrow\_input[0] = -1

31.

32.     def on\_right\_arrow\_press(self):

33.         self.arrow\_input[0] = 1

34.

35.     def on\_up\_arrow\_press(self):

36.         self.arrow\_input[1] = 1

37.

38.     def on\_down\_arrow\_press(self):

39.         self.arrow\_input[1] = -1

40.

41.     def on\_left\_arrow\_release(self):

42.         self.arrow\_input[0] = 0

43.

44.     def on\_right\_arrow\_release(self):

45.         self.arrow\_input[0] = 0

46.

47.     def on\_up\_arrow\_release(self):

48.         self.arrow\_input[1] = 0

49.

50.     def on\_down\_arrow\_release(self):

51.         self.arrow\_input[1] = 0

52.

53.     def on\_space\_press(self):

54.         self.space\_pressed = True

55.

56.     def on\_space\_release(self):

57.         self.space\_pressed = False

58.

59.     def on\_shift\_press(self):

60.         self.shift\_pressed = True

61.

62.     def on\_shift\_release(self):

63.         self.shift\_pressed = False

64.

65.     def on\_z\_press(self):

66.         self.z\_pressed = True

67.

68.     def on\_z\_release(self):

69.         self.z\_pressed = False

70.

71.     def on\_x\_press(self):

72.         self.x\_pressed = True

73.

74.     def on\_x\_release(self):

75.         self.x\_pressed = False

Η είσοδος από τα βελάκια του πληκτρολογίου αποτυπώνεται απλά σε μία λίστα 2 στοιχείων, ενώ εύκολα μπορεί να εντοπιστεί εάν ένα από τα κουμπιά Space,Shift, X, Z είναι πατημένα την δεδομένη στιγμή μέσω των μεταβλητών τύπου bool που έχουν οριστεί αρχικά. Οι μεταβλητές αυτές ανανεώνουν την τιμή τους κάθε φορά κάποιο γεγονός από το keyboard.py ενεργοποιείται.

## Εκπαίδευση Μοντέλου YOLO

### Συλλογή Φωτογραφιών

Το πρώτο βήμα για την εκπαίδευση του μοντέλου YOLO είναι η συλλογή φωτογραφικού υλικού. Τα αντικείμενα πρέπει να αναγνωρίσει το μοντέλο είναι: άδεια κελία, πιόνια Χ, και πιόνια Ο.

Εικόνα 3‑3 Τα αντικείμενα προς ανίχνευση

Για την ταχύτερη συλλογή των φωτογραφιών, αναπτύχθηκε ο παρακάτω αλγόριθμος στο αρχείο camera.py:

camera.py

1. def capture\_train\_photos(self):

2.         base\_folder = "data/ml\_photos"

3.         subfolder\_index = 1

4.         while os.path.exists(os.path.join(base\_folder, f"session\_{subfolder\_index}")):

5.             subfolder\_index += 1

6.         session\_folder = os.path.join(base\_folder, f"session\_{subfolder\_index}")

7.         os.makedirs(session\_folder)

8.

9.         photo\_index = 0

10.         while True:

11.             if input("Press 'q' to quit, any other key to capture photo: ") == 'q':

12.                 break

13.             else:

14.                 photo\_index += 1

15.                 self.capture\_photo(f"ml\_photo\_{photo\_index}", session\_folder, True)

16.         self.release()

Εικόνα 3‑4 Παραδείγματα φωτογραφικών δειγμάτων

Συνολικά, συγκεντρώθηκαν 214 φωτογραφίες. Όπως παρατηρήθηκε, τα πιόνια ήταν πιο δύσκολο να ανιχνευθούν από το μοντέλο, γι’ αυτό καταγράφηκαν περισσότερες φωτογραφίες για αυτά από όσες καταγράφηκαν για τα άδεια κελιά. Τέλος, δόθηκε έμφαση στην συλλογή εικόνων με ποικίλα επίπεδα και είδη φωτισμού όπως: φυσικός φωτισμός, φως λευκής λάμπας, και φως λάμπας με ζεστή, κιτρινωπή απόχρωση.

### Σήμανση Φωτογραφιών

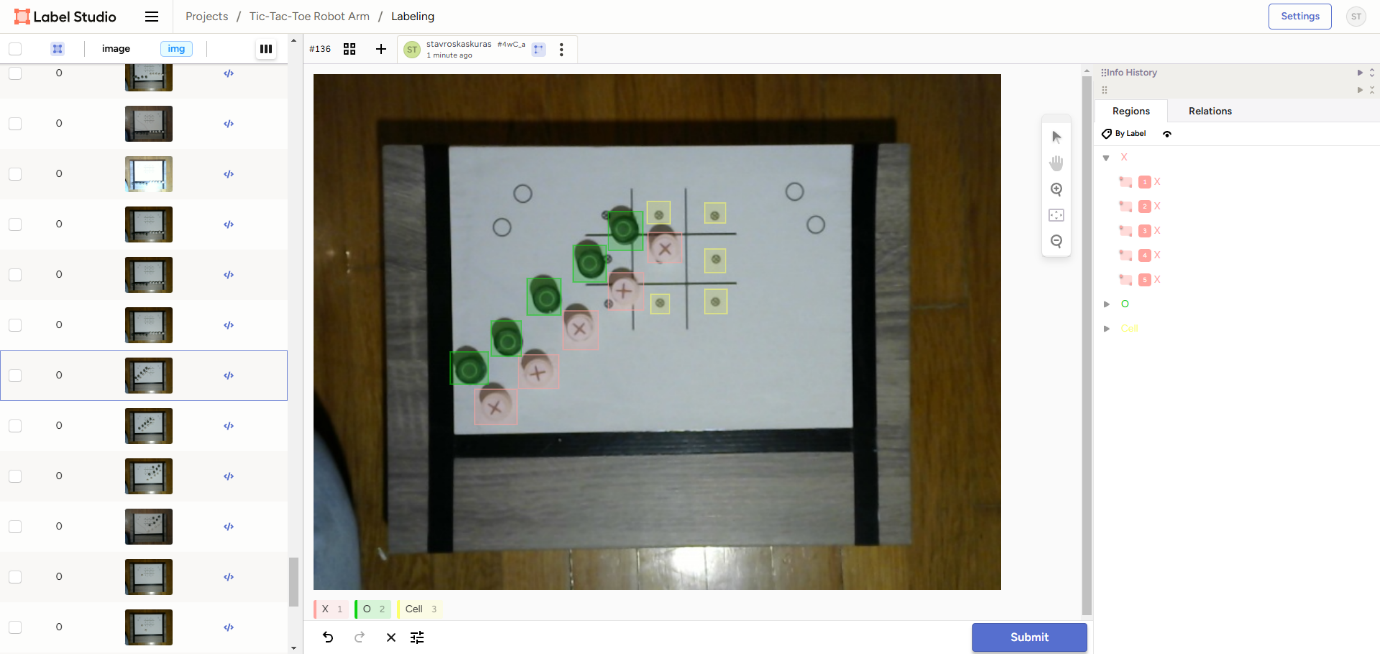
Για την σήμανση των δεδομένων, χρησιμοποιήθηκε το «Label Studio». Οδηγίες για την εγκατάσταση και χρήση του προγράμματος μπορούν να βρεθούν στην ιστοσελίδα: <https://labelstud.io/> [10].

Πρώτα ορίσαμε τις κλάσεις των αντικειμένων προς σήμανση. Στην περίπτωσή μας, οι:

* X
* O
* Cell

Είναι οι κλάσεις των πιονιών Χ, Ο, και των άδειων κελιών, αντίστοιχα. Αφού μεταφορτωθούν οι εικόνες στο πρόγραμμα, η σήμανσή τους μπορεί να ξεκινήσει.

Η σήμανση των εικόνων είναι μία απλή αλλά χρονοβόρα διαδικασία. Για κάθε μία φωτογραφία που καταγράφηκε, ορίζουμε τα χωρικά όρια μέσα στα οποία βρίσκεται κάθε αντικείμενο και την κλάση στην οποία ανήκει όπως φαίνεται στην Εικόνα 3‑5.



Εικόνα 3‑5 Σήμανση εικόνων στο πρόγραμμα label-studio

Αφού ολοκληρωθεί η διαδικασία για κάθε στοιχείο, εξάγουμε τα δεδομένα και τα αποθηκεύουμε στον υπολογιστή. Αυτά αποτελούνται από δύο φακέλους και δύο αρχεία:

* Φάκελος images
* Φάκελος labels
* classes.txt
* notes.json

Τα αρχεία classes.txt και notes.json εμπεριέχουν πληροφορίες σχετικά με το ποιες κλάσσεις ορίστηκαν κατά την εκπαίδευση, και αντιστοιχίζουν κάθε μία από αυτές με έναν μοναδικό αριθμό (id). Στον φάκελο images εμπεριέχονται όλες οι φωτογραφίες που σημάνθηκαν, ενώ στον φάκελο labels υπάρχουν αρχεία τύπου για κάθε σήμανση, τα οποία περιγράφουν τις συντεταγμένες των πλαισίων οριοθέτησης που δημιουργήθηκαν σε κάθε εικόνα.

### Εκπαίδευση Μοντέλου YOLO

Για την εκπαίδευση του μοντέλου, χρειάζεται να χωρίσουμε τα αρχεία των φακέλων images και labels σε δύο κατηγορίες: train και validate. Το μοντέλο χρησιμοποιεί τα αρχεία train για την εκμάθηση των χαρακτηριστικών κάθε αντικειμένου. Τα αρχεία validate χρησιμοποιούνται για την αξιολόγηση της απόδοσής του και την αποφυγή «υπερεκπαίδευσης» (overfitting)[[1]](#footnote-1). Συνήθως το 80%-90% χρησιμοποιείται για την εκμάθηση (train) και το υπόλοιπο 10%-20% χρησιμοποιείται για την αξιολόγηση (validate).

Στην παρούσα εφαρμογή, το 20% των εικόνων και τα αντίστοιχα δεδομένα σήμανσής τους μεταφέρθηκαν σε έναν νέο φάκελο με το όνομα val, ενώ τα υπόλοιπα στοιχεία μεταφερθήκαν σε έναν νέο φάκελο με το όνομα train. Έτσι, η δομή των αρχείων του project διαμορφώνεται ως εξής:

* Φάκελος train
  + Φάκελος images
  + Φάκελος labels
* Φάκελος val
  + Φάκελος images
  + Φάκελος labels

Στη συνέχεια, δημιουργούμε δύο ακόμα αρχεία: και . Το αρχείο χρησιμοποιείται από το μοντέλο για να εντοπιστεί την θέση των φακέλων train και val, ενώ ταυτόχρονα παρέχει το id και το όνομα των κλάσεων των αντικειμένων που θα πρέπει να εντοπιστούν.

data.yaml

1. path: .

2. train: C:/Users/Steve/Documents/Dev/Engineering Dev/Tic Tac Toe Robot Arm/ML Data/train/images  # training images

3. val: C:/Users/Steve/Documents/Dev/Engineering Dev/Tic Tac Toe Robot Arm/ML Data/val/images  # validation images

4. names:

5.   0: Cell

6.   1: O

7.   2: X

Το id που προσδίδεται σε κάθε μία από τις κλάσεις μεταξύ των γραμμών 5-7 του παραπάνω κώδικα πρέπει να ταιριάζει με την σειρά των κλάσεων, όπως αναφέρονται στο αρχείο .

Το αρχείο train.py ξεκινά την εκπαίδευση του μοντέλου ως εξής:

train.py

1. from ultralytics import YOLO

2. import os

3.

4. def train\_model():

5.     # Ensure the dataset path is correct

6.     dataset\_path = os.path.abspath(os.path.join(os.path.dirname(\_\_file\_\_), 'data.yaml'))

7.

8.     # Load the YOLOv8 model

9.     model = YOLO('yolov8.yaml')

10.

11.     # Train the model

12.     model.train(

13.         data=dataset\_path,  # path to the dataset configuration file

14.         epochs=50,  # number of epochs to train for

15.         imgsz=640,  # image size

16.         batch=16,  # batch size

17.         name='yolov8-tic-tac-toe',  # name of the training run

18.         device='cuda'  # use CUDA GPU instead of CPU

19.     )

20.

21. if \_\_name\_\_ == "\_\_main\_\_":

22.     train\_model()

23.

Όπως φαίνεται, το μοντέλο που χρησιμοποιήθηκε είναι το YOLOv8. Για την ταχύτερη εκπαίδευση του, χρησιμοποιήθηκε κάρτα γραφικών με πυρήνες CUDA. Μετά την ολοκλήρωση της εκπαίδευσης, δημιουργείται ένας φάκελος με το όνομα «runs». Εσωτερικά αυτού βρίσκεται ένας υποφάκελος «weights» ο οποίος περιέχει δύο αρχεία με τους συντελεστές βαρύτητας (weights) που υπολογίστηκαν από το νευρωνικό δίκτυο. Το πρώτο αρχείο εμπεριέχει τα βάρη που υπολογίστηκαν κατά την τελευταία εκπαίδευση (, ενώ το δεύτερο, τα βάρη που υπολογίστηκαν κατά την πιο ακριβή εκπαίδευση. Σε αυτή την εφαρμογή χρησιμοποιήθηκε το δεύτερο αρχείο.

## Καταγραφή και Επεξεργασία Εικόνας

### Καταγραφή Φωτογραφιών

Το αρχείο camera.py αποτελείται από 4 κύριες μεθόδους:

* capture\_photo
* release
* get\_feed\_photo
* get\_feed\_video

camera.py

1. import cv2

2. import os

3. import time

4.

5. class Camera:

6.     def \_\_init\_\_(self, camera\_index=0):

7.         self.camera = cv2.VideoCapture(camera\_index)

8.         if not self.camera.isOpened():

9.             raise Exception("Error: Could not open webcam.")

10.

11.     def capture\_photo(self, filename="webcam\_photo.jpg", save\_location="./photos", show\_preview=False):

12.         ret, frame = self.camera.read()

13.

14.         # Flush the buffer by reading a few frames

15.         for \_ in range(10):

16.             ret, frame = self.camera.read()

17.

18.         # Ensure the filename has a valid image extension

19.         if not filename.lower().endswith(('.png', '.jpg', '.jpeg', '.bmp', '.tiff')):

20.             filename += ".jpg"

21.

22.         # Create save location if it doesn't exist

23.         if not os.path.exists(save\_location):

24.             os.makedirs(save\_location)

25.

26.         # if the frame was captured successfully, save it

27.         if ret:

28.             full\_path = os.path.join(save\_location, filename)

29.             cv2.imwrite(full\_path, frame)

30.             print(f"Photo saved as {full\_path}")

31.

32.             if show\_preview:

33.                 # Display the captured photo

34.                 cv2.imshow("Captured Photo", frame)

35.                 cv2.waitKey(1)  # Display the image for 1 millisecond to ensure it shows up

36.                 time.sleep(1)  # Wait for 1 second

37.                 cv2.destroyAllWindows()

38.             return frame

39.         else:

40.             print("Error: Could not capture photo.")

41.             return None

42.

43.     def release(self):

44.         self.camera.release()

45.         cv2.destroyAllWindows()

46.

47.     def get\_feed\_photo(self):

48.         ret, frame = self.camera.read()

49.

50.         # Flush the buffer by reading a few frames

51.         for \_ in range(10):

52.             ret, frame = self.camera.read()

53.

54.         if not ret:

55.             print("Error: Could not read frame.")

56.             return None

57.         return frame

58.

59.     def get\_feed\_video(self):

60.         while True:

61.             frame = self.get\_feed\_photo()

62.             if frame is None:

63.                 break

64.             yield frame

65.         self.release()

66.

67.     def capture\_train\_photos(self):

68.         base\_folder = "data/ml\_photos"

69.         subfolder\_index = 1

70.         while os.path.exists(os.path.join(base\_folder, f"session\_{subfolder\_index}")):

71.             subfolder\_index += 1

72.         session\_folder = os.path.join(base\_folder, f"session\_{subfolder\_index}")

73.         os.makedirs(session\_folder)

74.

75.         photo\_index = 0

76.         while True:

77.             if input("Press 'q' to quit, any other key to capture photo: ") == 'q':

78.                 break

79.             else:

80.                 photo\_index += 1

81.                 self.capture\_photo(f"ml\_photo\_{photo\_index}", session\_folder, True)

82.         self.release()

83.

84. def main():

85.     camera = Camera()

86.     try:

87.         for frame in camera.get\_feed\_video():

88.             cv2.imshow("Video Feed", frame)

89.             if cv2.waitKey(1) & 0xFF == ord('q'):

90.                 break

91.     finally:

92.         camera.release()

93.         cv2.destroyAllWindows()

94.

95.

96. if \_\_name\_\_ == "\_\_main\_\_":

97.     main()

98.

Η μέθοδος capture\_photo καταγράφει και αποθηκεύει μία φωτογραφία μέσω της βιβλιοθήκης OpenCV. Μπορεί να εμφανίζει μία προεπισκόπηση της φωτογραφίας στην οθόνη για λίγα δευτερόλεπτα, εάν επιλεγεί από τον χρήστη. Τελικά, επιστρέφει το τελευταίο καρέ που καταγράφηκε.

Η μέθοδος release απλά αποδεσμεύει την κάμερα για να χρησιμοποιηθεί από κάποια άλλη συνάρτηση ή πρόγραμμα.

Οι μέθοδος get\_feed\_photo έχει την ίδια λειτουργία με την capture\_photo, με την διαφορά ότι δεν αποθηκεύει την φωτογραφία στον υπολογιστή. Αντ’ αυτού, επιστρέφει απλώς το τελευταίο καρέ που καταγράφηκε.

Για την συνεχή λήψη καρέ, χρησιμοποιείται η μέθοδος get\_feed\_video, η οποία μέσω ενός βρόγχου, επιστρέφει συνεχώς το τελευταίο καρέ έως ότου ο βρόγχος διακοπεί.

### Επεξεργασία Εικόνας – Ανίχνευση Αντικειμένων

Η ανίχνευση των αντικειμένων γίνεται μέσω του αρχείου image\_processor.py. Αρχικά ορίζουμε μία κλάση ImageProcessor, το μοντέλο το οποίο χρησιμοποιείται, και τον ελάχιστο βαθμό βεβαιότητας ο οποίος πρέπει να χαρακτηρίζει μία ανίχνευση για να θεωρείται αυτή έγκυρη.

Image\_processor.py

1. import cv2

2. from ultralytics import YOLO

3.

4. class ImageProcessor:

5.     def \_\_init\_\_(self):

6.         self.model = YOLO("last.pt")

7.         self.min\_confidence = 0.45

Στην συνέχεια ορίζουμε την μέθοδο detect\_objects η οποία χρησιμοποιείται για την ανίχνευση των αντικειμένων σε ΄ένα καρέ (frame):

Image\_processor.py

1. def detect\_objects(self, frame):

2.         """

3.         Detects objects in the frame and returns the labels and bounding boxes.

4.         """

5.         if frame is None:

6.             return []

7.

8.         results = self.model(frame)

9.

10.         objects = []

11.         for result in results:

12.             for box in result.boxes:

13.                 x1, y1, x2, y2 = map(int, box.xyxy[0])

14.                 label = result.names[int(box.cls)]

15.                 confidence = box.conf[0]

16.                 if confidence > self.min\_confidence:

17.                     object = (label, (x1, y1, x2, y2))

18.                     objects.append(object)

19.

20.         return objects

Για κάθε εικόνα που παρέχεται σαν παράμετρος της μεθόδου self.model στην γραμμή 8 του παραπάνω κώδικα, εξάγεται ένα αποτέλεσμα. Η μεταβλητή results εμπεριέχει τα αποτελέσματα ανίχνευσης όλων των εικόνων.

Ο αλγόριθμος στη συνέχεια ελέγχει κάθε αντικείμενο που ανιχνεύθηκε στην εικόνα μέσω ενός for loop, και εάν αυτό έχει αρκετά μεγάλο βαθμό βεβαιότητας, το προσθέτει σε μία λίστα objects η οποία αποτελείται από στοιχεία τύπου .

H θέση label παίρνει τιμές string “X”, “O” ή “Cell”. Οι συντεταγμένες εκφράζουν το ορθογώνιο πλαίσιο (bounding box) που περικλείει το αντικείμενο. Συγκεκριμένα, το ζεύγος ορίζει την θέση της πάνω αριστερής γωνίας του πλαισίου, ενώ το αυτή της κάτω δεξιάς. Τελικά, η λίστα objects επιστρέφεται από την μέθοδο.

Η επόμενη μέθοδος της κλάσης ImageProcessor που θα εξετάσουμε είναι η detect\_grid\_area. Στόχος αυτής της συνάρτησης είναι η αναγνώριση των ορίων μεταξύ των οποίων ανιχνεύεται το πλέγμα της τρίλιζας στην εικόνα.

Image\_processor.py

1. def detect\_grid\_area(self, cells):

2.         """

3.         Detects the grid area from the provided cells and returns the top left and bottom right coordinates.

4.         """

5.         if not cells:

6.             return None, None

7.

8.         cell\_positions = []

9.         total\_width = 0

10.         total\_height = 0

11.

12.         for cell in cells:

13.             x1, y1, x2, y2 = map(int, cell[1])

14.             # Add the center of the box to the list of cell positions

15.             center = ((x1 + x2) // 2, (y1 + y2) // 2)

16.             cell\_positions.append(center)

17.             # Calculate the width and height of the cell

18.             total\_width += (x2 - x1)

19.             total\_height += (y2 - y1)

20.

21.         # Calculate the average width and height of a cell

22.         cell\_width = total\_width // len(cells)

23.         cell\_height = total\_height // len(cells)

24.

25.         # Find the top left and bottom right coordinates of the grid area

26.         top\_left = min(cell\_positions, key=lambda x: x[0] + x[1])

27.         bottom\_right = max(cell\_positions, key=lambda x: x[0] + x[1])

28.

29.         # Offset the top left and bottom right coordinates

30.         top\_left = (top\_left[0] - cell\_width \* 2, top\_left[1] - cell\_height \* 2)

31.         bottom\_right = (bottom\_right[0] + cell\_width \* 2, bottom\_right[1] + cell\_height \* 2)

32.

33.         return top\_left, bottom\_right

Η συνάρτηση δέχεται ως παράμετρο μία λίστα με στοιχεία τύπου (label, ()) όπως τα αυτά τα οποία επιστρέφονται στην συνάρτηση detect\_objects. Τα στοιχεία αυτά πρέπει να ανήκουν στην κλάση Cell, δηλαδή να έχουν τη μεταβλητή label ίση με “Cell”. Για το φιλτράρισμα των αντικειμένων με βάση το label, ορίζεται μία νέα μέθοδος filter\_objects\_by\_label:

image\_processor.py

1. def filter\_objects\_by\_label(self, name, objects):

2.         """

3.         Filters the detected objects by the given name.

4.         """

5.         return [object for object in objects if object[0] == name]

Επιστρέφοντας την προσοχή μας στη μέθοδο detect\_grid\_area, παρατηρούμε πως αρχικά υπολογίζεται η συντεταγμένη του κέντρου κάθε άδειου κελιού. Οι συντεταγμένες αυτές συγκεντρώνονται σε μία λίστα, μέσω της οποίας βρίσκουμε την επάνω αριστερά και κάτω δεξιά συντεταγμένη του πλέγματος στην εικόνα. Ύστερα υπολογίζεται το μέσο πλάτος και ύψος των κελιών. Τελικά, εφαρμόζεται μία απόκλιση και στις δύο συντεταγμένες των άκρων και το αποτέλεσμα επιστρέφεται από την συνάρτηση.

Στην ίδια κλάση ορίζονται 6 ακόμα κλάσεις:

filter\_objects\_within\_grid: Δέχεται ως είσοδο μία λίστα αντικειμένων (label, () και την περιοχή του πλέγματος. Εξάγει μία λίστα με τα αντικείμενα τα οποία βρίσκονται εντός του πλέγματος.

image\_processor.py

1. def filter\_objects\_within\_grid(self, objects, grid\_area):

2. """

3. Returns only the objects that are within the grid area.

4. """

5. if not grid\_area or not objects:

6. return objects

7.

8. filtered\_objects = []

9. for object in objects:

10. object\_center = ((object[1][0] + object[1][2]) // 2, (object[1][1] + object[1][3]) // 2)

11. is\_within\_grid = grid\_area[0][0] < object\_center[0] < grid\_area[1][0] and grid\_area[0][1] < object\_center[1] < grid\_area[1][1]

12. if is\_within\_grid:

13. filtered\_objects.append(object)

14.

15. return filtered\_objects

16.

draw\_bounding\_box: Δέχεται ως είσοδο ένα καρέ και ένα αντικείμενο (label, (), ζωγραφίζει ένα πλαίσιο οριοθέτησης γύρω από το αντικείμενο και επισημάνει την κλάση του με κείμενο το οποίο εμφανίζεται στην κορυφή του.

image\_processor.py

1. def draw\_bounding\_box(self, frame, object):

2. label, (x1, y1, x2, y2) = object

3.

4. colors = {

5. 'X': (0, 0, 255),

6. 'O': (0, 255, 0),

7. 'Cell': (255, 255, 255),

8. }

9. color = colors.get(label, (255, 255, 255))

10. cv2.rectangle(frame, (x1, y1), (x2, y2), color, 2)

11. cv2.putText(frame, f"{label}", (x1, y1 - 10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, color, 2)

12. return frame

draw\_objects\_in\_frame: Δέχεται ως είσοδο ένα καρέ και καλεί την μέθοδο draw\_bounding\_box σε κάθε αντικείμενο που ανιχνεύεται. Επίσης επισημάνει με κείμενο στην πάνω δεξιά μεριά της εικόνας τον αριθμό αντικειμένων που εντοπίστηκαν ανά κλάση.

image\_processor.py

1. def draw\_objects\_in\_frame(self, frame, objects):

2. if not objects:

3. return frame

4.

5. def draw\_class\_counts(frame, class\_counts):

6. y\_offset = 20

7. for label, count in class\_counts.items():

8. text = f"{label}: {count}"

9. cv2.putText(frame, text, (frame.shape[1] - 150, y\_offset), cv2.FONT\_HERSHEY\_SIMPLEX, 0.6, (255, 255, 255), 2)

10. y\_offset += 30

11. return frame

12.

13. class\_counts = {}

14. for object in objects:

15. label = object[0]

16. class\_counts[label] = class\_counts.get(label, 0) + 1

17. frame = self.draw\_bounding\_box(frame, object)

18.

19. frame = draw\_class\_counts(frame, class\_counts)

20.

21. return frame

22.

draw\_grid: Δέχεται ως είσοδο ένα καρέ και τις συντεταγμένες των ορίων του πλέγματος στην εικόνα. Ζωγραφίζει ένα πλαίσιο στην περιοχή που ορίζουν τα όρια του πλέγματος και επιστρέφει το καρέ.

image\_processor.py

1. def draw\_grid(self, frame, grid\_area):

2. if grid\_area is None:

3. return frame

4.

5. top\_left, bottom\_right = grid\_area

6.

7. if top\_left is None or bottom\_right is None:

8. return frame

9.

10. overlay = frame.copy()

11. cv2.rectangle(overlay, top\_left, bottom\_right, (0, 255, 255), -1)

12. alpha = 0.05 # Transparency factor.

13. cv2.addWeighted(overlay, alpha, frame, 1 - alpha, 0, frame)

14. cv2.rectangle(frame, top\_left, bottom\_right, (0, 255, 255), 2)

15. return frame

draw\_grid\_with\_objects: Καλεί τις δύο προηγούμενες μεθόδους σε ένα καρέ ταυτόχρονα.

image\_processor.py

1. def draw\_grid\_and\_objects(self, frame, grid\_area, objects):

2. frame\_with\_grid = self.draw\_grid(frame, grid\_area)

3. frame\_with\_objects = self.draw\_objects\_in\_frame(frame\_with\_grid, objects)

4. return frame\_with\_objects

## Αλγόριθμος Τρίλιζας

### Πλέγμα

Το πιο βασικό στοιχείο ενός παιχνιδιού τρίλιζας είναι το πλέγμα, γι’ αυτό τον λόγο θα ξεκινήσουμε αυτή την ενότητα με την περιγραφή του αρχείου grid.py:

grid.py

1. from input.event import Event

2.

3. class Grid:

4.     def \_\_init\_\_(self):

5.         # Create a 3x3 grid filled with empty spaces

6.         self.on\_winner\_detected = Event()

7.         self.on\_turn\_end = Event()

8.         self.rows = [[' ' for \_ in range(3)] for \_ in range(3)]

9.

10.     def set\_rows(self, rows):

11.         self.rows = rows

12.

13.     def check\_for\_winner(self):

14.         # Check rows for a winner

15.         winner = None

16.

17.         for row in self.rows:

18.             if row[0] == row[1] == row[2] and row[0] != ' ':

19.                 winner = row[0]

20.

21.         # Check columns for a winner

22.         for col in range(3):

23.             if self.rows[0][col] == self.rows[1][col] == self.rows[2][col] and self.rows[0][col] != ' ':

24.                 winner = self.rows[0][col]

25.

26.         # Check diagonals for a winner

27.         if self.rows[0][0] == self.rows[1][1] == self.rows[2][2] and self.rows[0][0] != ' ':

28.             winner = self.rows[0][0]

29.         if self.rows[0][2] == self.rows[1][1] == self.rows[2][0] and self.rows[0][2] != ' ':

30.             winner = self.rows[0][2]

31.

32.         return winner

33.

34.     def is\_grid\_full(self):

35.         for row in self.rows:

36.             for cell in row:

37.                 if cell == ' ':

38.                     return False

39.         return True

Αρχικά ορίζεται μία κλάση Grid, μαζί με δύο αντικείμενα τύπου Event και μία δισδιάστατη λίστα self.rows. Τα γεγονότα ενεργοποιούνται όταν τελειώνει η σειρά του ενός παίκτη ή όταν ανιχνεύεται νικητής. Η λίστα rows αποτελείται από τρείς λίστες, με τρία στοιχεία η κάθε μία. Τα στοιχεία μπορούν να πάρουν τιμές:

* “ “ (κενό): άδειο κελί
* “X”: σύμβολο X
* “O”: σύμβολο O

Στον constructor της κλάσης αρχικοποιείται η μεταβλητή self.rows ώστε να αναπαριστά ένα πλέγμα με μόνο άδεια κελιά. Η συνάρτηση set\_rows απλά θέτει μία τιμή στην μεταβλητή self.rows και χρησιμοποιείται από άλλες κλάσεις όπως θα δούμε. Τέλος, η μέθοδος check\_for\_winner ελέγχει εάν υπάρχει νικητής, εξετάζοντας εάν εμφανίζεται το ίδιο σύμβολο τρείς φορές σε μία σειρά, στήλη ή διαγώνια γραμμή του πλέγματος.

### Ψηφιοποίηση Δεδομένων Παιχνιδιού

Το σύστημα πρέπει να είναι σε θέση να καταλαβαίνει από μία φωτογραφία, το στάδιο στο οποίο βρίσκεται το πλέγμα της τρίλιζας κάθε φορά, έτσι ώστε να τροφοδοτήσει αυτά τα δεδομένα στον παίκτη-ρομπότ και αυτός να αντιδράσει κατάλληλα. Για την ψηφιοποίηση των δεδομένων του πλέγματος, δημιουργήθηκε το αρχείο grid\_digitizer.py και η κλάση GridDigitizer:

grid\_digitizer.py

1. class GridDigitizer:

2.     def \_\_init\_\_(self):

3.         self.image\_processor = ImageProcessor()

4.         self.camera = Camera()

5.         self.grid\_area = None

6.         self.grid = Grid()

7.

Στην κλάση αυτή γίνεται χρήση του επεξεργαστή εικόνας (ImageProcessor), της κάμερας (Camera) και του πλέγματος, ενώ ορίζεται και μία μεταβλητή grid\_area, της οποίας την τιμή μπορούμε να βρούμε μέσω του ImageProcessor, όπως είδαμε στην ενότητα 3.6. Συγκεκριμένα, η τιμή της μεταβλητής self.grid\_area ελέγχεται από την συνάρτηση capture\_grid\_area ως εξής:

grid\_digitizer.py

1.     def capture\_grid\_area(self):

2.         # Capture frame from camera

3.         frame = self.camera.get\_feed\_photo()

4.         if frame is None:

5.             print("Failed to capture image from camera.")

6.             return None

7.

8.         # Detect objects in the frame

9.         objects = self.image\_processor.detect\_objects(frame)

10.

11.         # Detect grid area

12.         cells = self.image\_processor.filter\_objects\_by\_label("Cell", objects)

13.         grid\_area = self.image\_processor.detect\_grid\_area(cells)

14.

15.         self.grid\_area = grid\_area

16.

17.         grid\_detections = self.image\_processor.filter\_objects\_within\_grid(objects, self.grid\_area)

18.         self.display\_detections(frame, grid\_detections)

19.         print("Grid area captured successfully.")

20.

Η πιο βασική και χρήσιμη μέθοδος αυτής της κλάσης είναι η μέθοδος capture\_grid\_state:

grid\_digitizer.py

1.     def capture\_grid\_area(self):

2.         # Capture frame from camera

3.         frame = self.camera.get\_feed\_photo()

4.         if frame is None:

5.             print("Failed to capture image from camera.")

6.             return None

7.

8.         # Detect objects in the frame

9.         objects = self.image\_processor.detect\_objects(frame)

10.

11.         # Detect grid area

12.         cells = self.image\_processor.filter\_objects\_by\_label("Cell", objects)

13.         grid\_area = self.image\_processor.detect\_grid\_area(cells)

14.

15.         self.grid\_area = grid\_area

16.

17.         grid\_detections = self.image\_processor.filter\_objects\_within\_grid(objects, self.grid\_area)

18.         self.display\_detections(frame, grid\_detections)

19.         print("Grid area captured successfully.")

20.

21.     def capture\_grid\_state(self):

22.         # returns all objects that are within the grid area

23.         def capture\_objects\_in\_grid\_area():

24.             if(self.grid\_area is None):

25.                 print("Grid area is not initialized.")

26.                 return None

27.

28.             # Capture frame from camera

29.             frame = self.camera.get\_feed\_photo()

30.

31.             if frame is None:

32.                 print("Failed to capture image from camera.")

33.                 return

34.

35.             # Detect objects in the frame

36.             objects = self.image\_processor.detect\_objects(frame)

37.

38.             # Filter objects within the grid area

39.             grid\_objects = self.image\_processor.filter\_objects\_within\_grid(objects, self.grid\_area)

40.             self.display\_detections(frame, grid\_objects)

41.

42.             return grid\_objects

43.

44.         grid\_objects = capture\_objects\_in\_grid\_area()

45.         if grid\_objects is None:

46.             return None

47.         else:

48.             grid\_width = self.grid\_area[1][0] - self.grid\_area[0][0]

49.             grid\_height = self.grid\_area[1][1] - self.grid\_area[0][1]

50.             cell\_width = grid\_width // 3

51.             cell\_height = grid\_height // 3

52.

53.             grid\_rows = [[' ' for \_ in range(3)] for \_ in range(3)]

54.             for obj in grid\_objects:

55.                 if(obj[0] == "X" or obj[0] == "O"):

56.                     x1, y1, x2, y2 = obj[1]

57.                     center = ((x1 + x2) // 2, (y1 + y2) // 2)

58.                     obj\_row = (center[1] - self.grid\_area[0][1]) // cell\_height

59.                     obj\_col = (center[0] - self.grid\_area[0][0]) // cell\_width

60.                     grid\_rows[obj\_row][obj\_col] = obj[0]

61.

62.             self.grid.set\_rows(grid\_rows)

63.

64.         return self.grid

65.

Σε υψηλό επίπεδο, η συνάρτηση αρχικά καταγράφει μία φωτογραφία της τρίλιζας την δεδομένη στιγμή, την επεξεργάζεται και εντοπίζει την περιοχή στην οποία βρίσκεται το πλέγμα στην εικόνα καθώς και τα αντικείμενα (πιόνια, άδεια κελιά) που υπάρχουν εντός αυτής. Με βάση τις συντεταγμένες της επάνω αριστερής και κάτω δεξιάς γωνίας της περιοχής του πλέγματος, υπολογίζεται το πλάτος και το ύψος του. Στη συνέχεια, το πλάτος και το ύψος χωρίζονται σε τρία ίσα τμήματα, δημιουργώντας ένα πλέγμα εννέα κελιών. Εάν ένα ανιχνευμένο πιόνι “X” ή “O” βρίσκεται εντός των ορίων κάποιου κελιού, η αντίστοιχη τιμή καταχωρείται στην αρμόδια θέση του πίνακα grid\_rows. Το αντικείμενο self.grid ενημερώνεται καταλλήλως, και επιστρέφεται από την συνάρτηση.

Στην ίδια κλάση έχουν προστεθεί δύο ακόμα μέθοδοι για την οπτική απεικόνιση του σταδίου της τρίλιζας:

display\_detections: Εμφανίζει μία προεπισκόπηση των αντικειμένων που εντοπίστηκαν εντός του πλέγματος:

grid\_digitizer.py

1.     def display\_detections(self, frame, grid\_objects):

2.         frame = frame.copy()

3.

4.         # Draw the grid and objects on the processed frame

5.         processed\_frame = self.image\_processor.draw\_grid\_and\_objects(frame, self.grid\_area, grid\_objects)

6.

7.         cv2.imshow("Grid State", processed\_frame)

8.         cv2.waitKey(2000)  # Display for 2000 milliseconds (2 seconds)

9.         cv2.destroyAllWindows()

10.

display\_grid\_state: Εκτυπώνει το ανιχνευμένο στάδιο της τρίλιζας σε μορφή κειμένου:

grid\_digitizer.py

1.     def display\_grid\_state(self):

2.         for row in self.grid.rows:

3.             print(' | '.join(row))

4.             print('-' \* 10)

### Παίκτες

Το παιχνίδι αποτελείται από δύο παίκτες, τον χρήστη (άνθρωπο) και τον ρομποτικό βραχίονα, οι οποίοι παίζουν ως αντίπαλοι. Από εδώ και πέρα θα αναφερόμαστε στον χρήστη ως human-player και στον ρομποτικό βραχίονα ως bot-player. Και τα δύο είδη παίκτη μοιράζονται κάποια κοινά χαρακτηριστικά τα οποία φαίνονται στην κλάση Player, του αρχείου player.py:

player.py

1. from input.event import Event

2.

3. class Player:

4.     def \_\_init\_\_(self, id, game\_manager):

5.         self.id = id

6.         self.game\_manager = game\_manager

7.         self.on\_move\_completed = Event()

8.         self.symbol = None

9.

10.         if(self.id == "human"):

11.             self.symbol = self.game\_manager.human\_player\_symbol

12.         elif(self.id == "bot"):

13.             self.symbol = self.game\_manager.bot\_player\_symbol

14.

15.     def make\_move(self):

16.         raise NotImplementedError("Subclasses should implement this method.")

17.

Όπως φαίνεται κάθε παίκτης θα πρέπει να έχει μία μοναδική ταυτότητα id, πρόσβαση στον διαχειριστή του παιχνιδιού (self.game\_manager), ένα σύμβολο (“X” ή “O”) το οποίο θα χρησιμοποιεί στο παιχνίδι, και μία συνάρτηση make\_move η οποία θα εκτελείται όταν έρχεται η σειρά του να παίξει. Επίσης ορίζεται ένα Event, on\_move\_completed, το οποίο θα ενεργοποιείται όταν ο παίκτης έχει ολοκληρώσει την κίνησή του.

Ο έλεγχος του human-player γίνεται μέσω του αρχείου human\_player.py όπου ορίζεται η κλάση HumanPlayer:

1. from tic\_tac\_toe.player import Player

2.

3. class HumanPlayer(Player):

4.     def \_\_init\_\_(self,id, game\_manager):

5.         super().\_\_init\_\_(id, game\_manager)

6.

7.     def make\_move(self):

8.         input("Press Enter when you have made your move...")

9.         self.on\_move\_completed.invoke()

10.

H κλάση HumanPlayer κληρονομεί (inherits) από την κλάση Player και ορίζει την λειτουργία της μεθόδου make\_move η οποία εκτελείται όταν έρχεται η σειρά του ανθρώπου να παίξει. Όταν έρχεται η σειρά του χρήστη, ένα κείμενο "Press Enter when you have made your move..." θα εκτυπωθεί στην οθόνη του υπολογιστή. Ο χρήστης τότε θα πρέπει να κάνει την κίνησή του επάνω στην τρίλιζα, τοποθετώντας ένα πιόνι του κατάλληλου συμβόλου επάνω σε ένα άδειο κελί. Πατώντας Enter στο πληκτρολόγιο ενεργοποιείται το Event on\_move\_completed.

Για τον έλεγχο του bot-player ορίζουμε μία νέα κλάση ΑΙ, η οποία εφαρμόζει απλή τεχνητή νοημοσύνη για να αποφασίσει την επόμενη κίνηση του ρομπότ.

AI.py

1. class AI(Player):

2.     def \_\_init\_\_(self, id, game\_manager):

3.         super().\_\_init\_\_(id, game\_manager)

4.

5.         self.arm = RobotArm()

6.         self.grab\_index = 0

7.         self.grab\_coordinates = [[8.9, 0.4, -41.5], [12.6, 0.4, -42.2], [9.1, 0.2, 55.3], [12.3, 0.2, 52]]

8.         self.grid\_cell\_coordinates = [

9.             # Low right-center-left

10.             [[6.5, 1, -16], [4.6, 1, 6], [6.5, 1, 26]],

11.             # Middle right-center-left

12.             [[9.5, 1, -13], [9, 1, 4], [9.5, 1, 21]],

13.             # Upper right-center-left

14.             [[13, 1, -10], [13, 1, 2], [13, 1, 17]]

15.         ]

16.

17.         self.reset\_arm\_position()

18.

Η μεταβλητή self.grab\_coordinates είναι μία λίστα που περιέχει τις γενικευμένες συντεταγμένες στις οποίες θα πρέπει να μετακινηθεί ο βραχίονας έτσι ώστε να γραπώσει ένα πιόνι του συμβόλου που έχει οριστεί για τον bot-player. Αντίστοιχα, η μεταβλητή self.grid\_cell\_coordinates είναι ένας πίνακας ο οποίος αποτελείται από τις συντεταγμένες στις οποίες εντοπίζεται το κάθε κελί της τρίλιζας στο χώρο. Θυμίζουμε πως η τρίλιζα μένει πάντα σταθερή στον χώρο σε σχέση με τον βραχίονα για αυτό και όλες τιμές που προα ναφέρθηκαν είναι στατικές και δεν μεταβάλλονται. Ο ακέραιος self.grab\_index υποδεικνύει τον αριθμό των πιονιών που έχουν ήδη χρησιμοποιηθεί από τον bot-player.

Οι τρείς μέθοδοι που φαίνονται στον παρακάτω κώδικα είναι υπεύθυνες για το γράπωμα και την τοποθέτηση πιονιών στην τρίλιζα:

AI.py

1. def place\_object\_to\_grid(self, row, col):

2.         self.grab\_piece()

3.         time.sleep(0.5)

4.         pos = self.grid\_cell\_coordinates[row][col]

5.         self.arm.set\_position(pos[0], pos[1] + 2, pos[2])

6.         time.sleep(0.5)

7.         self.arm.set\_position(pos[0], pos[1] + 0.3, pos[2])

8.         time.sleep(0.5)

9.         self.release\_piece()

10.         time.sleep(0.5)

11.         self.reset\_arm\_position()

12.         time.sleep(0.5)

13.

14. def reset\_arm\_position(self):

15.         self.arm.set\_position(8, 8, 90)

16.

17.     def grab\_piece(self):

18.         grab\_position = self.grab\_coordinates[self.grab\_index]

19.

20.         # Open the grip, move to the grab position with a height offset, close the grip and lift the piece

21.         self.arm.set\_grip(20)

22.         time.sleep(1)

23.         self.arm.set\_position(grab\_position[0] - 2, grab\_position[1] + 2, grab\_position[2])

24.         time.sleep(1)

25.         self.arm.set\_position(grab\_position[0] - 2, grab\_position[1], grab\_position[2])

26.         time.sleep(1)

27.         self.arm.set\_position(grab\_position[0], grab\_position[1], grab\_position[2])

28.         time.sleep(1)

29.         self.arm.set\_grip(5)

30.         time.sleep(1.5)

31.         self.arm.set\_position(6, 8, 0)

32.

33.         self.grab\_index = (self.grab\_index + 1) % len(self.grab\_coordinates)

34.

35.     def release\_piece(self):

36.         self.arm.set\_grip(25)

37.

Η μέθοδος grab\_piece γραπώνει τα πιόνια από την βάση, και η place\_object\_to\_grid τα τοποθετεί στην γραμμή row και στήλη col του πλέγματος. Ακόμα, μία μέθοδος reset\_arm\_position επαναφέρει τον βραχίονα στην προεπιλεγμένη θέση του, ενώ μία άλλη, release\_piece, ανοίγει την δαγκάνα του για να απελευθερώσει το πιόνι που κρατά.

Οι δύο τελευταίες μέθοδοι που ανήκουν στην κλάση AI είναι οι calculate\_best\_move και make\_move. Από την ονομασία τους είναι φανερό πως η πρώτη διαβάζει το στάδιο στο οποίο βρίσκεται η τρίλιζα και υπολογίζει την καλύτερη δυνατή κίνηση που μπορεί να γίνει από τον bot-player, ενώ η δεύτερη την εκτελεί:

1. def make\_move(self):

2.         (row, col) = self.calculate\_best\_move(self.game\_manager.grid, self.symbol)

3.         self.place\_object\_to\_grid(row, col)

4.         self.on\_move\_completed.invoke()

5.

6.     def calculate\_best\_move(self, grid, symbol):

7.         opponent\_symbol = 'O' if symbol == 'X' else 'X'

8.

9.         # Check if AI can win in the next move

10.         for row in range(3):

11.             for col in range(3):

12.                 if grid.elements[row][col] == ' ':

13.                     elements = [r[:] for r in grid.elements]

14.                     elements[row][col] = symbol

15.                     new\_grid = Grid()

16.                     new\_grid.set\_rows(elements)

17.                     if new\_grid.check\_for\_winner():

18.                         return row, col

19.

20.         # Check if opponent can win in the next move and block them

21.         for row in range(3):

22.             for col in range(3):

23.                 if grid.elements[row][col] == ' ':

24.                     elements = [r[:] for r in grid.elements]

25.                     elements[row][col] = opponent\_symbol

26.                     new\_grid = Grid()

27.                     new\_grid.set\_rows(elements)

28.                     if new\_grid.check\_for\_winner():

29.                         return row, col

30.

31.         # Place symbol in a random available cell

32.         available\_moves = [(row, col) for row in range(3) for col in range(3) if grid.elements[row][col] == ' ']

33.         return random.choice(available\_moves)

34.

1. Προκύπτει όταν το μοντέλο αδυνατεί να αναγνωρίσει αντικείμενα τα οποία διαφέρουν αρκετά από αυτά που χρησιμοποιήθηκαν για την εκπαίδευσή του. [↑](#footnote-ref-1)