Πανεπιστήμιο Πειραιώς – Τμήμα Πληροφορικής

Πρόγραμμα Μεταπτυχιακών Σπουδών

«Ψηφιακός Πολιτισμός, Έξυπνες Πόλεις, IoT και Προηγμένες Ψηφιακές Τεχνολογίες»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	Μελέτη και Υλοποίηση Αυτόνομου Οχήματος Autonomous Vehicle Implementation
Ονοματεπώνυμο Φοιτητή	Κωνσταντίνος Χατζής
Πατρώνυμο	Ιωάννης
Αριθμός Μητρώου	ΨΠΟΛ/19067
Επιβλέπων	Εμμανουήλ Σκόνδρας

Ημερομηνία Παράδοσης, Νοέμβριος 2023

Τριμελής Εξεταστική Επιτροπή

Δρ. Εμμανουήλ Σκόνδρας Διδάσκων ΠΜΣ	Δημήτριος Δ. Βέργαδος Καθηγητής	Τσίγκας Επαμεινώνδας ΕΔΙΠ

Περίληψη

Σκοπός αυτής της μεταπτυχιακής διατριβής είναι η κατασκευή ενός αυτόνομου οχήματος μικρής κλίμακας. Η διατριβή καλύπτει όλες τις πτυχές της δημιουργίας ενός τέτοιου οχήματος για την κατασκευή και την ενσωμάτωση της πλατφόρμας υλικού για τον έλεγχο του οχήματος μέσω λογισμικού. Επίσης καλύπτει πλήρως τη δημιουργία, την εκπαίδευση και την εφαρμογή ενός μοντέλου βαθιάς μάθησης με τη χρήση συνελικτικών νευρωνικών δικτύων το οποίο λαμβάνει ως είσοδο υλικό από την κάμερα του οχήματος και ελέγχει την κίνηση και τη συμπεριφορά του οχήματος. Τέλος καταλήγει σε κάποια συμπεράσματα σχετικά με το όχημα αλλά και τη μεταφορά της συγκεκριμένης τεχνολογίας σε πραγματικά οχήματα.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Αυτόνομα οχήματα, Νευρωνικά δίκτυα, Μηχανική μάθηση

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: τεχνητή νοημοσύνη, βαθιά μάθηση, αυτοκινούμενο όχημα, υπολογιστική όραση

Abstract

The goal of this thesis is the creation of an autonomous RC scale car model. This thesis covers all the aspects of the creation of an autonomous vehicle for the construction and the integration of the hardware platform to control the vehicle via software. It also covers the generation, the training and the application of a deep learning model, based on the technology of convolutional neural networks, which receives input from the camera and controls the movement and the behavior of the vehicle. Finally, it comes to conclusions regarding the vehicle but also the transfer of the specific technology to real vehicles.

SUBJECT AREA: Autonomous vehicles, Neural networks, Machine learning

KEYWORDS: artificial intelligence, deep learning, self-driving vehicle, computer vision

Περιεχόμενα

1. Εισαγωγή

Το θέμα αυτής της διπλωματικής εργασίας είναι η κατασκευή ενός αυτόνομου οχήματος και πιο συγκεκριμένα αυτό-οδηγούμενο όχημα RC. Στόχος της διατριβής είναι η ανάπτυξη ενός μοντέλου για αυτόνομη οδήγηση σε πίστα, επιδεικνύοντας ταυτόχρονα την ικανότητα εκτέλεσης συμπεριφορών όπως η αποφυγή εμποδίων. Η διατριβή αφορά όλη τη διαδικασία κατασκευής του οχήματος καθώς και την ανάπτυξη του μοντέλου οδήγησης.

Το κύριο κίνητρο πίσω από το επιλεγόμενο θέμα είναι η γρήγορη ανάπτυξη των εφαρμογών τεχνητής νοημοσύνης (AI) σε πολλούς τομείς και η σημασία των αυτόνομων οχημάτων για την ανθρωπότητα. Τα αυτόνομα οχήματα προβλέπεται ότι θα βοηθήσουν τον άνθρωπο και σε άλλους τομείς, όπως η εξερεύνηση του διαστήματος και των πλανητών. Η άνοδος της τεχνητής παράλληλα με τις μεθόδους της βαθιάς μάθησης (DL) καταστούν δυνατή τη δημιουργία ενός τέτοιου οχήματος χωρίς τη χρήση ακριβών εργαστηρίων και χρόνιας έρευνας.

Επί του παρόντος, πολλές ιδιωτικές εταιρείες και ακαδημαϊκές ομάδες εργάζονται πάνω στον τομέα των αυτόνομων οχημάτων και την ενσωμάτωσή τους στους υφιστάμενους κανονισμούς και νόμους. Μερικές από τις κορυφαίες εταιρείες είναι η Google Waymo, η οποία διαθέτει ήδη επιβατικά και φορτηγά χωρίς οδηγό και η Tesla που έχει το πιο ευρέως χρησιμοποιούμενο αυτόνομο σύστημα οχημάτων στον κόσμο.

Εκτός από τα τεχνολογικά εμπόδια που δεν έχουν ακόμα ξεπεραστεί, ένα μεγάλο εμπόδιο που αποκλείει τα αυτόνομα οχήματα από τους δρόμους είναι ηθικής και νομοθετικής φύσης. Πιο συγκεκριμένα, ήδη έχουν συμβεί τα πρώτα θανατηφόρα ατυχήματα με αυτόνομα οχήματα, γεγονός που οδήγησε σε έρευνες και συζητήσεις σχετικά με τις υποχρεώσεις ενός αυτόνομου οχήματος ώστε να περιηγείται σε πραγματικούς δρόμους συνυπάρχοντας με οχήματα που οδηγούνται από ανθρώπους, εστιάζοντας ειδικά σε καταστάσεις στις οποίες μια θανατηφόρα σύγκρουση μπορεί να είναι αναπόφευκτη όπου το όχημα πρέπει να αποφασίσει ποια πορεία δράσης πρέπει να ακολουθήσει, από αυτοθυσία που θα σήμαινε θυσία των επιβατών που μεταφέρει το όχημα μέχρι τραυματισμό άλλων συμμετεχόντων στην κυκλοφορία. Άλλα ζητήματα αφορούν το κόστος και την επεκτασιμότητα της αυτόνομης οδήγησης και την τυποποίηση της ασφάλειας σε τέτοια οχήματα.

Παρά τις δυσκολίες που υπάρχουν, τα πλεονεκτήματα των αυτόνομων οχημάτων για τον άνθρωπο και το περιβάλλον είναι πολλά. Ορισμένα από αυτά είναι:

1. Ασφάλεια: Η πλειονότητα των ατυχημάτων που συμβαίνουν στους δρόμους προκαλούνται από ανθρώπινο λάθος. Τα αυτόνομα οχήματα μπορούν να μειώσουν τον αριθμό των ατυχημάτων, καθώς έχουν τη δυνατότητα να αντιλαμβάνονται το περιβάλλον τους και να λαμβάνουν ασφαλείς αποφάσεις κατά την οδήγηση.
2. Συμφόρηση κίνησης: Τα αυτόνομα οχήματα μπορούν να βελτιώσουν τη ροή της κίνησης και να μειώσουν τη συμφόρηση στους δρόμους. Επικοινωνούν μεταξύ τους και συντονίζουν τις κινήσεις τους, αποφεύγοντας παραβιάσεις στους κόμβους κυκλοφορίας και τη συμφόρηση.
3. Εξοικονόμηση ενέργειας: Τα αυτόνομα οχήματα μπορούν να οδηγούν με πιο οικονομικό τρόπο, βελτιστοποιώντας την κατανάλωση καυσίμων και μειώνοντας τις εκπομπές ρύπων.
4. Βελτιωμένη χρήση χώρου: Η τεχνολογία των αυτόνομων οχημάτων μπορεί να σχεδιαστεί έτσι ώστε να μπορούν να κινούνται πιο αποτελεσματικά και να εκμεταλλεύονται πλήρως τον χώρο του οδοστρώματος.
5. Προσβασιμότητα: Τα αυτόνομα οχήματα μπορούν να προσφέρουν νέες δυνατότητες μετακίνησης σε άτομα με περιορισμένη κινητικότητα ή αναπηρία.
6. Αύξηση παραγωγικότητας: Η δυνατότητα για τους ανθρώπους να δουλεύουν ή να εξομοιώνουν κατά την διάρκεια των μετακινήσεών τους, χωρίς να αφιερώνουν χρόνο στην οδήγηση, μπορεί να αυξήσει την παραγωγικότητα.

Είναι προφανές, με όλα όσα αναφέρθηκαν παραπάνω αλλά και με την εκρηκτική άνοδο της τεχνητής νοημοσύνης ότι ο τομέας των αυτόνομων οχημάτων βρίσκεται στην αρχή του και

ότι θα έχει σημαντικό αντίκτυπο στην κοινωνία, τόσο από οικονομικής όσο και από ηθικής άποψης. Η προσβασιμότητα μιας τέτοιας τεχνολογίας θα πρέπει να γίνει ευρύτερα διαθέσιμη σε ερευνητές και φοιτητές, προκειμένου το πεδίο να συνεχίσει να προοδεύει μέσα από συζητήσεις και έρευνες και να μην μείνει στάσιμο, κάτι που είναι ένας από τους λόγους πίσω από το θέμα αυτής της διατριβής.

Ακολούθως, η εργασία περιλαμβάνει τα εξής κεφάλαια:

- Κεφάλαιο 2. Κατασκευή του οχήματος: Το κεφάλαιο 2 περιλαμβάνει πληροφορίες σχετικά με τα μηχανικά, τα ηλεκτρικά και τα ηλεκτρονικά μέρη του οχήματος. Επίσης περιγράφεται η συναρμολόγηση και η συνδεσμολογία του καθώς και πληροφορίες για το δίκτυο
- Κεφάλαιο 3. Λογισμικό οχήματος: Στο κεφάλαιο 3 περιλαμβάνονται πληροφορίες σχετικά με το λειτουργικό σύστημα του οχήματος και του δικτύου, καθώς και της εφαρμογής για τη λειτουργία του
- Κεφάλαιο 4. Επικοινωνία με το δίκτυο: Στο κεφάλαιο 4 περιγράφεται ο τρόπος επικοινωνίας του οχήματος με το δίκτυο και τον κεντρικό υπολογιστή μέσω του πρωτοκόλλου MQTT
- Κεφάλαιο 5. Απεικόνιση στο δίκτυο: Το κεφάλαιο 5 περιγράφει την απεικόνιση του οχήματος στο δίκτυο μέσω μιας web εφαρμογής γραμμένη σε Angular
- Κεφάλαιο 6. Τεχνητή νοημοσύνη & deep learning: Στο κεφάλαιο 6 παρουσιάζονται τα βασικά δομικά στοιχεία της τεχνητής νοημοσύνης και των νευρωνικών δικτύων
- Κεφάλαιο 7. Εκπαίδευση και δημιουργία μοντέλου αυτόνομης οδήγησης: Το κεφάλαιο 7 περιγράφει τη διαδικασία συλλογής δεδομένων και την εκπαίδευση ενός μοντέλου αυτόνομης οδήγησης με χρήση της γλώσσας προγραμματισμού Python
- Κεφάλαιο 8. Δοκιμή μοντέλου αυτόνομης οδήγησης και συμπεράσματα: Στο κεφάλαιο 8 περιγράφεται η εφαρμογή του μοντέλου αυτόνομης οδήγησης στο όχημα και τα συμπεράσματα που προκύπτουν από τη μεταπτυχιακή διατριβή

2. Κατασκευή του οχήματος

Σε αυτό το κεφάλαιο περιγράφονται τα εξαρτήματα που χρησιμοποιήθηκαν για την κατασκευή της πλατφόρμας υλικού, όπου θα τρέχει όλο το λογισμικό του οχήματος. Παράλληλα θα εξεταστούν εναλλακτικές λύσεις και θα δοθούν εξηγήσεις για κάθε επιλεγμένο κομμάτι υλικού.

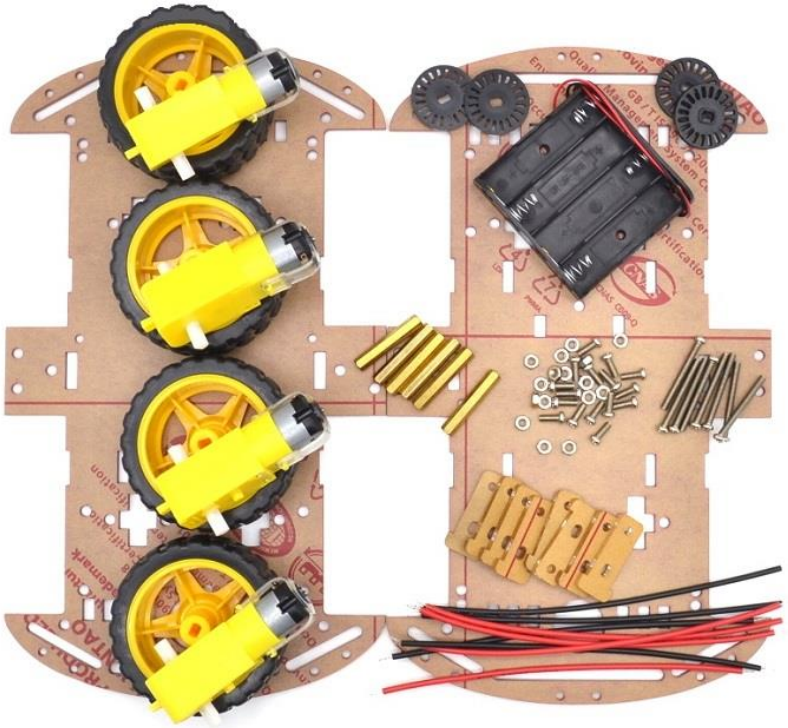
2.1 Μηχανικά μέρη οχήματος

Τα μηχανικά μέρη του οχήματος αποτελούνται από το αμάξωμα (σώμα οχήματος), το στήριγμα της κάμερας, τα στηρίγματα των κινητήρων και μία θήκη για το Jetson Nano

2.1.1 Αμάξωμα

Το αμάξωμα που χρησιμοποιήθηκε για την κατασκευή του οχήματος είναι το Robot Smart Car 4WD – Chassis 26cm [1]. Το Robot Smart Car 4WD είναι μία πολλαπλών χρήσεων πλατφόρμα ρομπότ και έχει από μόνη της μεγάλη ευελιξία. Διαθέτει τέσσερα μοτέρ με encoder και τροχούς 65mm. Οι πλάκες πλαισίου και τα στηρίγματα είναι από διάφανο ακρυλικό (plexiglass) 3 mm και έχουν οπές για την προσάρτηση πλακέτας Arduino, αισθητήρες, ελεγκτές και πρόσθετες μονάδες.

Για τη συναρμολόγησή του συνδέθηκαν οι δύο πλάκες πλαισίου με τα σιδερένια διαχωριστικά με τη χρήση των κατάλληλων βιδών. Τα περιεχόμενα του Robot Smart Car 4WD φαίνονται στην παρακάτω εικόνα.

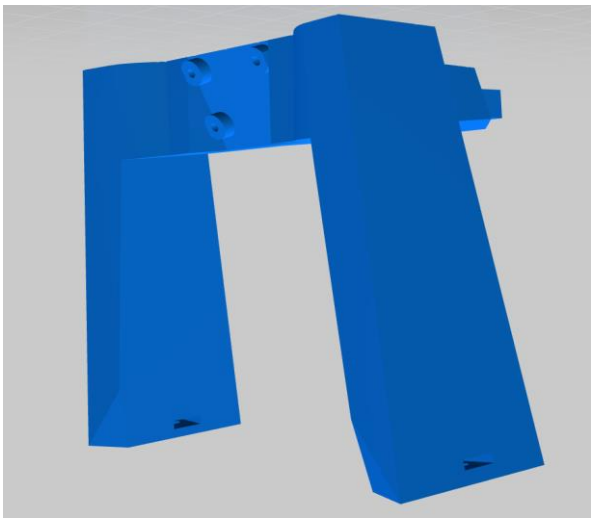


Εικόνα 1: Μέρη αμαξώματος [1]

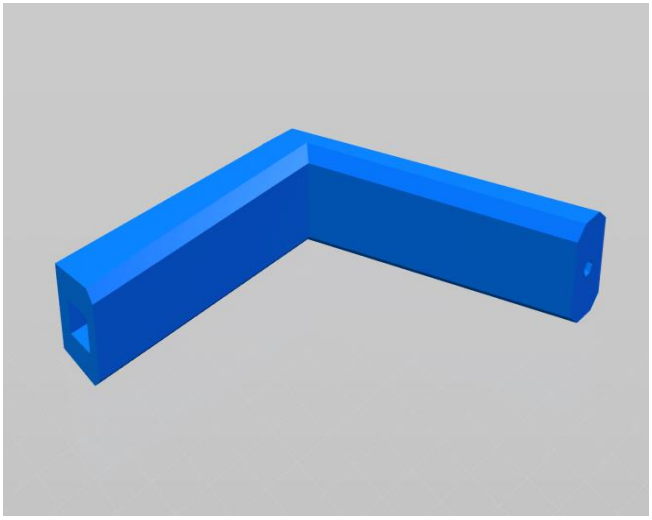
2.1.2 Στήριγμα κάμερας

Για να εξασφαλιστεί καλή στήριξη και σταθερότητα στην κάμερα χρησιμοποιήθηκε ένα στήριγμα. Το στήριγμα αυτό εκτυπώθηκε σε 3D εκτυπωτή και το σχέδιο που χρησιμοποιήθηκε είναι το Donkey Car - parts for small-footprint build plates από την ιστοσελίδα thingiverse.com [2]. Τα αρχεία που εκτυπώθηκαν έχουν προστεθεί στον φάκελο της εργασίας. Το στήριγμα αποτελείται από 2 κομμάτια τα οποία ενώθηκαν.

Ο 3D εκτυπωτής που χρησιμοποιήθηκε είναι ο Cr-20 Pro της Creality [3] και το υλικό κατασκευής του στηρίγματος είναι PLA [4]. Όλες οι 3D εκτυπώσεις πραγματοποιήθηκαν με τον ίδιο εκτυπωτή.



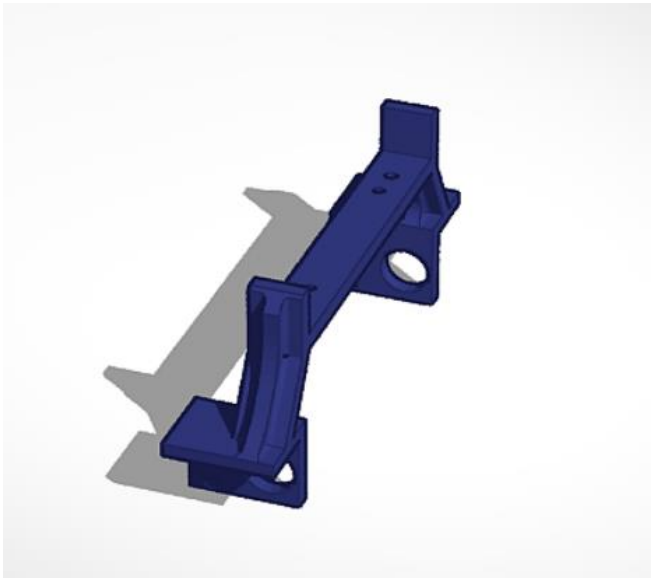
Εικόνα 2: Στήριγμα κάμερας - 1ο κομμάτι [2]



Εικόνα 3: Στήριγμα κάμερας - 2ο κομμάτι [2]

2.1.3 Στήριγμα κινητήρων

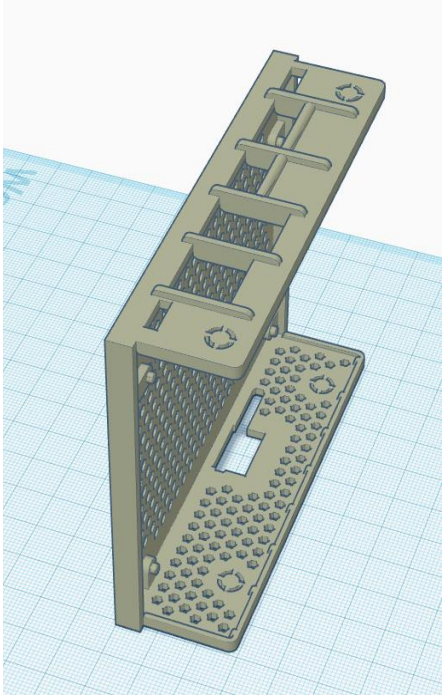
Τα στηρίγματα των κινητήρων του Robot Smart Car 4WD αντικαταστάθηκαν με νέα τα οποία σχεδιάστηκαν στην πλατφόρμα tinkercad [5] και εκτυπώθηκαν στον 3D εκτυπωτή ώστε να εξασφαλιστεί σταθερότητα στην κίνηση του οχήματος.



Εικόνα 4: Στήριγμα κινητήρων [5]

2.1.4 Θήκη Jetson Nano

Η θήκη του Jetson Nano έχει σκοπό να κρατάει σταθερή τη θέση του Jetson Nano κατά την κίνηση του οχήματος και για το λόγο αυτό έχει κολληθεί πάνω στο αμάξωμα. Τα σχέδια για τη θήκη βρέθηκαν στο thingiverse.com [6]. Εκτυπώθηκε μόνο η βάση της θήκης ώστε να μην εμποδίζεται η σύνδεση των GPIO με τα υπόλοιπα ηλεκτρονικά στοιχεία.



Εικόνα 5: Θήκη Jetson Nano [6]

2.2 Ηλεκτρικά μέρη οχήματος

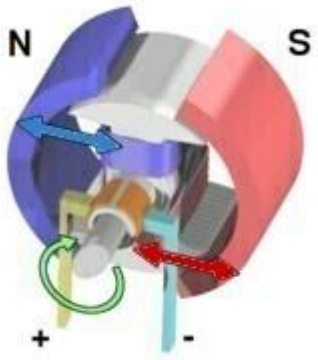
Σε αυτή την ενότητα περιγράφονται τα ηλεκτρικά μέρη του οχήματος. Αυτά είναι οι ηλεκτρικοί κινητήρες και η τροφοδοσία τους.

2.2.1 Ηλεκτρικοί κινητήρες

Οι κινητήρες του οχήματος είναι μέρος της συσκευασίας του Robot Smart Car 4WD – Chassis 26cm [1]. Πρόκειται για τέσσερις (4) TT κινητήρες σταθερής τάσης (DC) κίτρινου χρώματος με σχέση μετάδοσης 1:48.

Οι Κινητήρες DC είναι συνεχείς ενεργοποιητές που μετατρέπουν την ηλεκτρική ενέργεια σε μηχανική ενέργεια. Χρησιμοποιούνται συνήθως για την παραγωγή συνεχούς κίνηση και της οποίας η ταχύτητα περιστροφής μπορεί εύκολα να ελεγχθεί, καθιστώντας τους ιδανικούς για χρήση σε διάφορες εφαρμογές.

Ένας ηλεκτροκινητήρας συνεχούς ρεύματος αποτελείται από δύο μέρη, τον στάτη που είναι το σταθερό τμήμα και τον δρομέα που είναι το περιστρεφόμενο τμήμα. Έχουν σχεδόν γραμμικά χαρακτηριστικά και η ταχύτητα περιστροφής τους καθορίζεται από την εφαρμοζόμενη τάση συνεχούς ρεύματος και η ροπή εξόδου τους καθορίζεται από το ρεύμα που ρέει μέσω των περιελίξεων του κινητήρα.



Εικόνα 6: Ηλεκτρικός κινητήρας

Οι κινητήρες έχουν μονό άξονα και η τάση λειτουργίας τους είναι 3 – 6Volts. Η ταχύτητά τους είναι 125 σ.α.λ. και η ροπή 0,8kg cm. Είναι κατασκευασμένοι από πλαστικό και οι διαστάσεις τους είναι 2,5 ίντσες μήκος, 0,85 ίντσες πλάτος και 0,7 ίντσες πάχος.

2.2.2 Τροφοδοσία ηλεκτρικών κινητήρων

Η τροφοδοσία των ηλεκτρικών κινητήρων πραγματοποιείται με τρεις (3) μπαταρίες τύπου 18650 συνδεδεμένες παράλληλα. Οι μπαταρίες που επιλέχθηκαν είναι οι Panasonic NCR18650PF High drain TABS 18650 [7]. Οι μπαταρίες φορτίζονται ώστε να αποδώσουν συνολική τάση περίπου 12V.

2.3 Ηλεκτρονικά μέρη οχήματος

Παράλληλα τα μηχανικά και τα ηλεκτρικά μέρη του οχήματος πολύ σημαντικό ρόλο κατέχουν τα ηλεκτρονικά μέρη του. Πιο συγκεκριμένα τα ηλεκτρονικά μέρη και ο κύριος υπολογιστής είναι απαραίτητα για τον για τον έλεγχο του αυτοκινήτου μέσω λογισμικού και για την ανάπτυξη και εκτέλεση μοντέλων μηχανικής και βαθιάς μάθησης. Οι ακόλουθες ενότητες περιγράφουν τα ηλεκτρονικά μέρη που χρησιμοποιήθηκαν στο όχημα και τη σύνδεσή τους.

2.3.1 Κύριος υπολογιστής

Το πιο σημαντικό μέρος της πλατφόρμας υλικού είναι ο κύριος υπολογιστής όπου τρέχει το λειτουργικό σύστημα, το οποίο με τη σειρά του θα εκτελεί όλο το λογισμικό που απαιτείται για το μοντέλο Deep Learning καθώς και λογισμικό χαμηλότερου επιπέδου που επιτρέπει έλεγχο του οχήματος.

Αρχικά πραγματοποιήθηκε δοκιμή με το Raspeberry Pi 3 Model B σε συνδυασμό με το Intel Movidius Neural Compute Stick.

Το Raspberry Pi 3 Model B είναι ένας υπολογιστής σε μέγεθος πιστωτικής κάρτας. Διαθέτει επεξεργαστή Quad Core Processor 1.2GHz 64-Bit, 1GB RAM, 4 θύρες USB 2.0 για σύνδεση με πληκτρολόγιο, ποντίκι και άλλα περιφερειακά, θύρα Ethernet, WiFi, Bluetooth 4.1, έξοδο HDMI, έξοδο ήχου mini jack και microUSB υποδοχή για να την τροφοδοσία του. Για τη λειτουργία του απαιτείται τροφοδοσία 5V 2.5A και κάρτα microSD όπου θα εγκατασταθεί το λειτουργικό σύστημα.

Λόγω έλειψης GPU (Graphics Processing Unit) συνδυάστηκε με το Intel Movidius Neural Compute Stick. Το Intel Movidius αποτελεί μία VPU (Video Process Unit) της Intel η οποία χρησιμοποιείται για ανάπτυξη και εφαρμογή μοντέλων συνελικτικών νευρωνικών δικτύων (cnns) σε χαμηλή ισχύ που απαιτούν συμπεράσματα σε πραγματικό χρόνο.

Εικόνα 7: Raspberry Pi

Εικόνα 8: Intel Movidius Neural Compute Stick

Λόγω της χαμηλής υπολογιστικής ισχύος και κατ'επέκταση της μικρής απόδοσης επεξεργασίας των σύνθετων μοντέλων Deep Learning που θα χρησιμοποιηθούν σε αυτή την εργασία, οι συγκεκριμένες επιλογές εγκαταλήφθηκαν.

Επόμενη επιλογή αποτέλεσε το Jetson Nano 2 GB της εταιρείας NVIDIA. Το Jetson Nano είναι ένας μικρός, ισχυρός υπολογιστής που σας επιτρέπει να εκτελείτε πολλαπλά νευρωνικά δίκτυα παράλληλα για εφαρμογές όπως ταξινόμηση εικόνας, ανίχνευση αντικειμένων, τμηματοποίηση και επεξεργασία ομιλίας.

Αποτελείται από μία 128-core dedicated GPU (768 GFLOPS), 2GB PDDR4 RAM και έναν επεξεργαστή Quad-core ARM CPU (1.43 GHz). Επίσης διαθέτει:

- Θύρα Ethernet
- 1x USB 3.0 Type A,2x USB 2.0 Type A, USB 2.0 Micro-B
- HDMI έξοδο
- 40-pin header (GPIO, I2C, I2S, SPI, UART), 12-pin header (Power and related signals, UART), 4-pin Fan header
- 1x MIPI CSI-2 connector για την κάμερα

Εικόνα 9: Jetson Nano [14]

Το Jetson Nano ήταν η τελική επιλογή για την δημιουργία του αυτόνομου οχήματος.

2.3.2 Τροφοδοσία κύριου υπολογιστή

Η τροφοδοσία του κύριου υπολογιστή πραγματοποιήθηκε με τη χρήση ενός power bank. Πιο συγκεκριμένα με το Q10000 της Intenso [8] που υποστηρίζει Quick Charge 3.0. Το συγκεκριμένο power bank έχει χωρητικότητα 10000 mAh, 2 θύρες USB και ζυγίζει 260 gr.

2.3.3 Κάμερα

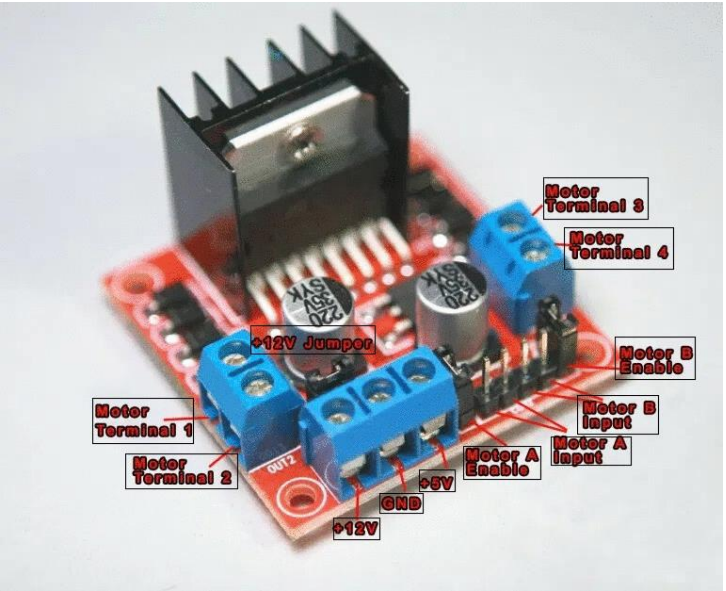
Οι αισθητήρες όρασης (κάμερες) είναι το κλειδί για αυτόνομα / ευφυή μηχανήματα και το Jetson υποστηρίζει πολλαπλές διεπαφές κάμερας, συμπεριλαμβανομένων USB, Ethernet και MIPI CSI. Για τις κάμερες που συνδέονται μέσω της διεπαφής MIPI CSI είναι απαραίτητο να έχουν αισθητήρα IMX219.

Η IMX219 Camera της Waveshare® [9] επιλέχθηκε για την υλοποίηση του αυτόνομου οχήματος. Τα χαρακτηριστικά της είναι τα εξής:

- 8 Megapixels
- Αισθητήρας IMX219
- Ανάλυση 3280 x 2464
- Γωνία θέασης 160 μοίρες

2.3.4 Stepper Motor Driver Controller Board L298N Dual H Bridge

Ο L298N είναι ένας οδηγός για κινητήρες συνεχούς ρεύματος και ακολουθεί τη διαμόρφωση της γέφυρας **H**, η οποία είναι χρήσιμη κατά τον έλεγχο της κατεύθυνσης περιστροφής.



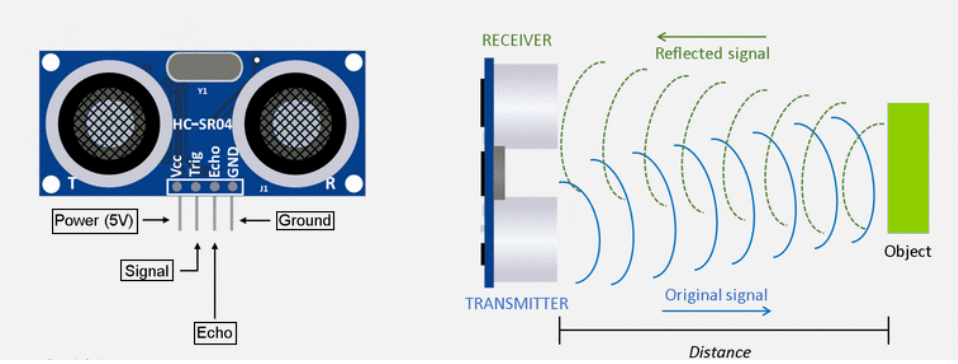
Εικόνα 10: L298N Dual H Bridge

Διαθέτει τέσσερις εισόδους (Motor Input) που αντιστοιχούν στους τέσσερις διακόπτες στο διάγραμμα γεφύρωσης. Για την περιστροφή των συνδεδεμένων κινητήτων είναι απαραίτητη η εφαρμογή σημάτων στις εισόδους. Επίσης ο πίνακας ελέγχου διαθέτει τερματικά +12V και +5V. Ο ακροδέκτης +12V μπορεί να δέχεται συνεχή τάση από 5V έως 35V. Αν η τάση εισόδου είναι μεγαλύτερη από 12V τότε πρέπει να αφαιρεθεί το βραχυκύκλωμα +12V Jumper. Το ρεύμα εξόδου ανά κινητήρα είναι 2A.

Ο οδηγός κινητήρα L298N έχει τη δυνατοτητα να ελέγχει και την ταχύτητα των συνδεδεμένων κινητήρων. Για το συγκεκριμένο σκοπό είναι απαραίτητη η τροφοδοσία των ακίδων ενεργοποίησης (Motor Enable) με σήματα PWM – Pulse Width Modulation. Η PWM αποτελεί μία τεχνική διαμόρφωσης η οποία παράγει παλμούς μεταβλητού πλάτους ώστε να αναπαρασταθεί το πλάτος ενός κύματος ή σήματος. Ως αποτέλεσμα η ταχύτητα του κινητήρα μεταβάλλεται ανάλογα με το πλάτος των παλμών. Επομένως είναι αναγκαίο να ενεργοποιηθούν τα PWM pins του Jetson, διαδικασία η οποία θα παρουσιαστεί σε επόμενο κεφάλαιο.

2.3.5 Αισθητήρας Υπερήχων

Ο αισθητήρας υπερήχων HC-SR04 χρησιμοποιείται για τον υπολογισμό της απόστασης του οχήματος από αντικείμενα που βρίσκονται μπροστά του και γι’ αυτό έχει τοποθετηθεί στο μπροστινό μέρος του αυτοκινήτου. Η απόσταση που μπορεί να υπολογίσει είναι από 2εκ. έως 400εκ. με ακρίβεια ενός εκατοστού. Η τάση εισόδου του αισθητήρα είναι 5V DC ενώ το ρεύμα λειτουργίας φτάνει τα 15mA και συνδέεται στις ψηφιακές εισόδους του Jetson.



Εικόνα 11: HC-SR04

Ο HC-SR04 λειτουργεί στέλνοντας ένα ηχητικό κύμα υψηλής συχνότητας και υπολογίζοντας το χρόνο για την ανάκλαση του σήματος πίσω. Τα 2 ανοίγματα που έχει στο μπροστινό μέρος χρησιμοποιούνται ως πομπός και δέκτης των κυμάτων αντίστοιχα. Βασική αρχή λειτουργίας του είναι ότι ο ήχος ταξιδεύει με περίπου 341 m/s στον αέρα. Χρησιμοποιώντας αυτή την αρχή μαζί με το χρόνο που απαιτείται για την αποστολή και τη λήψη του ηχητικού κύματος, ο αισθητήρας καθορίζει την απόσταση την απόσταση από το αντικείμενο χρησιμοποιώντας τον τύπο:

Απόσταση = (Χρόνος Αποστολής ηχητικού κύματος x ταχύτητα του ήχου) / 2

2.3.6 LED λυχνίες

Οι λυχνίες LED ή οι δίοδοι εκπομπής φωτός είναι μικρά ηλεκτρονικά εξαρτήματα στα οποία όταν παρέχεται ηλεκτρική τάση εκπέμπουν φως σε διάφορα χρώματα. Εάν το ρεύμα που διέρχεται από τη δίοδο είναι πολύ υψηλό, ενδέχεται να καταστρέψει το LED. Για την προστασία του LED και τον περιορισμό του ρεύμαος είναι σύνηθες να χρησημοποιείται μια αντίσταση συνδεδεμένη σε σειρά.

2.3.7 Ασύρματη Σύνδεση

Η ασύρματη σύνδεση του οχήματος με το δίκτυο πραγματοποιείται με τη χρήση ενός 802.11ac wireless adapter που συνδέεται σε μία USB θύρα του Jetson Nano.

2.3.8 Χώρος αποθήκευσης

Το Jetson Nano απαιτεί τη χρήση κάρτας Micro SD ως το κύριο εξάρτημα αποθήκευσης από το οποίο εκτελεί το λειτουργικό του σύστημα (OS) και όλο το άλλο λογισμικό.

Για τον σκοπό αυτής της διατριβής θα χρησιμοποιηθεί μία κάρτα μνήμης Samsung EVO Plus 32 GB U1 class [10] με ταχύτητα ανάγνωσης μέχρι 95 MB/s με διεπαφή UHS-1 και ταχύτητα εγγραφής μέχρι 20 MB/s.

2.3.9 Διάφορα μικροηλεκτρονικά και καλώδια

Για την ένωση όλων των παραπάνω εξαρτημάτων χρησιμοποιήθηκαν:

- Διάφορα καλώδια (Jumper Wires) με αρχιτεκτονικές θηλυκό σε θηλυκό, αρσενικό σε αρσενικό και αρσενικό σε θηλυκο
- Μία πλακέτα κατασκευών (Breadboard)
- Μία θήκη για τις μπαταρίες όπου συνδέονται σε σειρά

- Κολλητική ταινία
- Ένας διακόπτης ρεύματος
- Αντιστάσεις διαφόρων μεγεθών
- Βίδες και παξιμάδια

2.4 Συναρμολόγηση οχήματος

Το τελευταίο βήμα για την κατασκευή του οχήματος είναι η συναρμολόγησή του με χρήση των εξαρτημάτων που περιγράφηκαν παραπάνω. Οι υποενότητες που ακολουθούν θα περιγράψουν τη ροή της συναρμολόγησης.

2.4.1 Συναρμολόγηση αμαξώματος

Για τη συναρμολόγηση του οχήματος αρχικά κατασκευάστηκε ο κύριος κορμός του αμαξώματος ο οποίος περιλαμβάνει 2 επίπεδα.

Στο πρώτο επίπεδο, ανάμεσα στις 2 πλάκες του πλαισίου τοποθετήθηκαν:

- Οι μπαταρίες για την τροφοδοσία των ηλεκτρικών κινητήρων
- Ο οδηγός L298N για την τους κινητήρες
- Ο αισθητήρας υπερήχων, με τη χρήση μιας βάσης

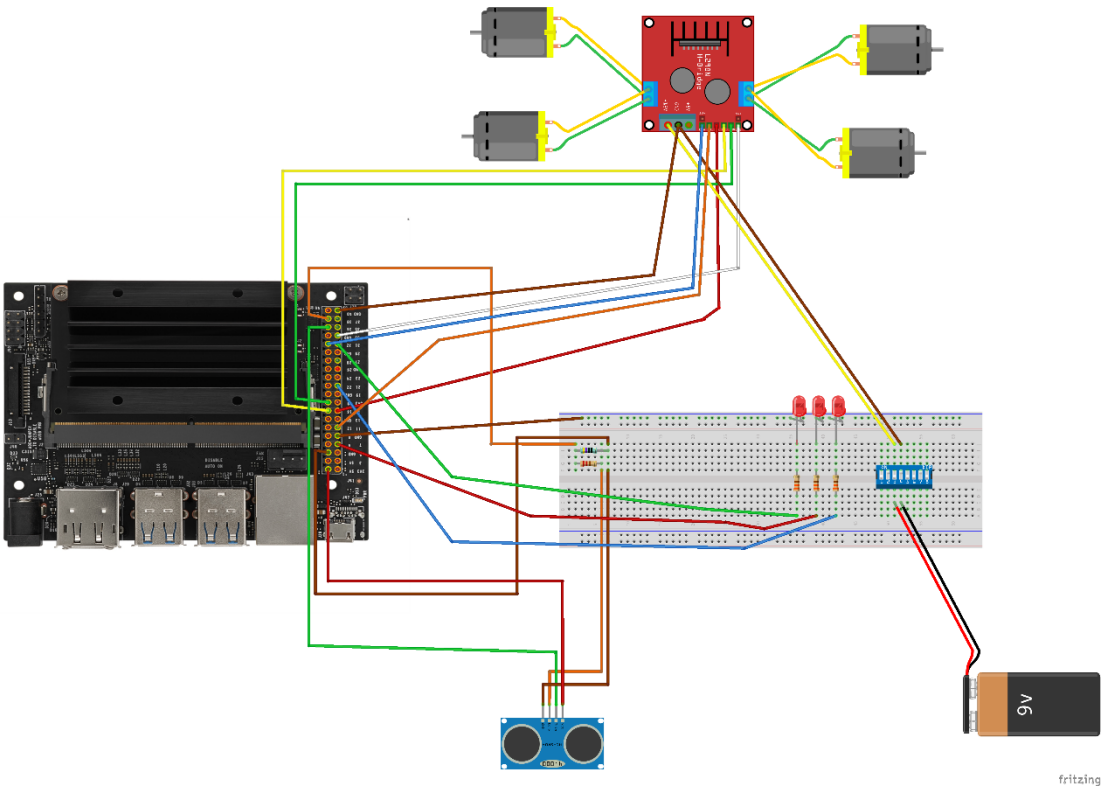
Στο δεύτερο επίπεδο, πάνω στο πλαίσιο του αμαξώματος τοποθετήθηκαν:

- Το στήριγμα της κάμερας μαζί με την κάμερα
- Η τροφοδοσία του κεντρικού υπολογιστή
- Ο κεντρικός υπολογιστής μαζί με τη βάση του
- Η πλακέτα κατασκευών μαζί με τις LED λυχνίες

Τέλος, κάτω από το όχημα συνδέθηκαν τα στηρίγματα των κινητήρων μαζί με τους κινητήρες και τα ελαστικά.

2.4.2 Συνδεσμολογία ηλεκτρονικών και ηλεκτρικών εξαρτημάτων

Η συνδεσμολογία των ηλεκτρικών και ηλεκτρονικών εξαρτημάτων του οχήματος παρουσιάζεται στην παρακάτω εικόνα.



Εικόνα 12: Συνδεσμολογία οχήματος

Πιο συγκεκριμένα:

- Οι 3 LED λυχνίες συνδέθηκαν στα pins 7,21 και 31 μέσω αντιστάσεων 330Ω και έπειτα στη γείωση
- Ο αισθητήρας υπερήχων συνδέθηκε ως εξής
  - Το pin τροφοδοσίας του αισθητήρα σε πιν 5V του Jetson
  - Το pin GND του αισθητήρα σε γείωση μέσω αντίστασης 470Ω
  - Το pin echo του αισθητήρα μέσω αντίστασης 330Ω στο pin 38
  - Το pin Trig του αισθητήρα στο pin 36 του Jetson
- Ο L298N συνδέθηκε με την τροφοδοσία του μέσω διακόπτη και
  - Το pin ENA του ελεγκτή συνδέθηκε στο pin 32 του Jetson
  - Το pin IN1 του ελεγκτή συνδέθηκε στο pin 11 του Jetson
  - Το pin IN2 του ελεγκτή συνδέθηκε στο pin 15 του Jetson
  - Το pin IN3 του ελεγκτή συνδέθηκε στο pin 16 του Jetson
  - Το pin IN4 του ελεγκτή συνδέθηκε στο pin 18 του Jetson
  - Το pin ENB του ελεγκτή συνδέθηκε στο pin 33 του Jetson
- Οι κινητήρες συνδέθηκαν στις εξόδους του L298N

2.5 Fog Node Hardware

Ως Hardware για το Fog χρησιμοποιήθηκε ένας φορητός ηλεκτρονικός υπολογιστής HP ZBook 15 G2 με τα εξής χαρακτηριστικά:

- Επεξεργαστή Intel® Core™ i7-4700MQ CPU @ 2.40GHz × 8
- Μνήμη 32GB
- Κάρτα γραφικών Quadro K1100M/PCIe/SSE2 / Quadro K1100M/PCIe/SSE2
- Σκληρό δίσκο 1TB

Περισσότερες πληροφορίες σχετικές με το μοντέλο βρίσκονται στη σελίδα της HP [11]

2.6 Network Hardware

Η σύνδεση του οχήματος με το Fog Node πραγματοποιείται μέσω WiFi οπότε σαν harware για το δίκτυο μπορεί να χρησιμοποιηθεί οπουδήποτε ασύρματο WiFi modem router είτε συσκευή κινητού τηλεφώνου χρησιμοποιώντας την ως mobile hotspot.

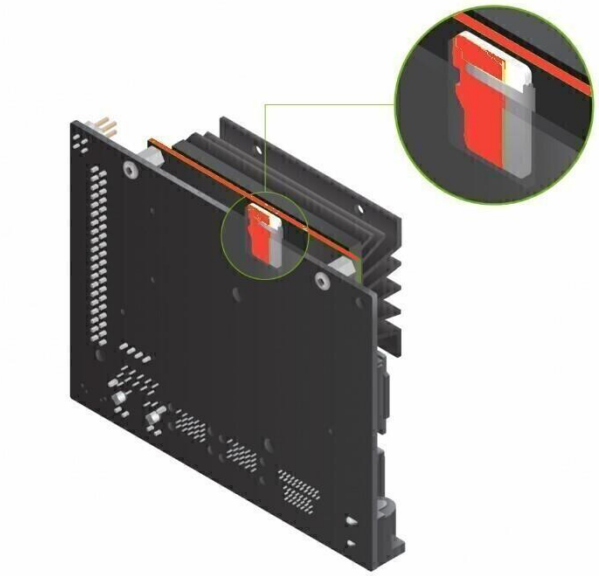
3. Λογισμικό οχήματος

Πέρα από τα μηχανικά και ηλεκτρικά μέρη του οχήματος, απαραίτητο για την κίνησή του είναι και το λογισμικό του. Στο κεφάλαιο που ακολουθεί περιγράφεται το λειτουργικό σύστημα του κεντρικού υπολογιστή του οχήματος (Jetson Nano) και η εφαρμογή σε γλώσσα προγραμματισμού Python η οποία χρησιμοποιείται για την κίνηση και τον έλεγχο των αισθητήρων του οχήματος.

3.1 Λειτουργικό σύστημα κεντρικού υπολογιστή οχήματος

Ο κεντρικός υπολογιστής (Jetson Nano) έχει εγκατεστημένη την έκδοση 4.4.1 του Jetpack [12]. Το Jetpack είναι ένα κιτ ανάπτυξης λογισμικού της NVIDIA που απευθύνεται στις πλακέτες Jetson. Περιλαμβάνει το Linux for Tegra(L4T) ως λειτουργικό σύστημα, το οποίο βασίζεται στα Ubuntu 18.04, καθώς και CUDA-X βιβλιοθήκες και διεπαφές για βαθιά μάθηση, υπολογιστική όραση και πολυμέσα.

Η εγκατάσταση του λειτουργικού συστήματος πραγματοποιήθηκε στην Micro SD κάρτα μνήμης με τη χρήση του προγράμματος balenaEtcher [13] και σύμφωνα με τις οδηγίες στην ιστοσελίδα του Jetson Nano [14]. Η Micro SD κάρτα μετά την ολοκλήρωση της εγκατάστασης τοποθετήθηκε στο Jetson στη θέση που φαίνεται παρακάτω.



Εικόνα 13: Θέση SD κάρτας [14]

Μετά την εγκατάσταση του λογισμικού και την είσοδο στο σύστημα είναι απαραίτητη η διαμόρφωση των pins του Jetson ώστε να ενεργοποιηθούν τα pins που υποστηρίζουν PWM. Αυτά τα pins (32, 33) χρησιμοποιούνται για τον έλεγχο της ταχύτητας των κινητήρων, όπως αναφέρθηκε στο

προηγούμενο κεφάλαιο. Επομένως ακολουθώντας τις οδηγίες που δίνονται στα επίσημα έγγραφα της NVIDIA [15] σχετικά με τη διαμόρφωση των pins, πραγματοποιήθηκε η ενεργοποίησή τους.

3.2 Εφαρμογή λειτουργίας του οχήματος (Python)

Μετά το πέρας της εγκατάστασης του λειτουργικού συστήματος στο Jetson σειρά έχει η δημιουργία μιας εφαρμογής σε γλώσσα Python με σκοπό τον έλεγχο των κινητήρων και των αισθητήρων που είναι συνδεδεμένοι σε αυτό.

Πιο συγκεκριμένα, δημιουργήθηκε ένας φάκελος με όνομα Autonomous_car όπου βρίσκονται τα αρχεία της εφαρμογής. Έπειτα η εφαρμογή χωρίστηκε σε πακέτα ώστε να είναι πιο ξεκάθαρο σε ποια λειτουργία αναφέρεται η κάθε κλάση της Python. Η δομή των πακέτων της εφαρμογής παρουσιάζεται παρακάτω:

Autonomous_car

- autopilot
  - following_road
- camera
- car
- data
- directory_manager
- distance_sensor
- led
- mqtt_connection
- utils

Περισσότερες πληροφορίες εμπεριέχονται στην ιστοσελίδα Github [16]. Τα πακέτα autopilot και mqtt_connection θα αναλυθούν σε επόμενα κεφάλαια.

Για το όχημα έχουν οριστεί 3 διαφορετικοί τρόποι λειτουργίας:

1. η εγγραφή δεδομένων κίνησης του οχήματος, όπου το όχημα οδηγείται χειροκίνητα από τον χειριστή και ταυτόχρονα καταγράφει πληροφορίες και εικόνες
2. η εγγραφή δεδομένων στάσης του οχήματος, όπου καταγράφονται εικόνες και πληροφορίες με τις οποίες το όχημα σταματάει να κινείται
3. ο αυτόματος πιλότος, όπου το όχημα κινείται εντός των ορίων του δρόμου χωρίς να το “οδηγεί” κάποιος χειριστής

3.2.1 Το πακέτο utils

Το πακέτο utils περιλαμβάνει βοηθητικές κλάσεις Python, οι οποίες χρησιμοποιούνται από τις υπόλοιπες για την ολοκλήρωση των μεθόδων τους.

- **constants.py**: Η κλάση constants.py χρησιμοποιείται για την αποθήκευση κάποιων σταθερών τιμών της εφαρμογής όπως τα pins του Jetson που είναι συνδεδεμένος κάθε αισθητήρας, στοιχεία για τον MQTT broker, ονόματα των MQTT topics κ.ά.
- **keyboard_reader.py**: Σκοπός της κλάσης keyboard_reader.py είναι να μπορεί η εφαρμογή να διαβάσει τα πλήκτρα που πατιούνται στο πληκτρολόγιο. Χρησιμεύει κυρίως για την οδήγηση του οχήματος.
- **preprocessor.py**: Η κλάση preprocessor.py χρησιμεύει για την μεταμόρφωση των εικόνων που τραβάει η κάμερα. Χρησιμοποιείται στη λειτουργία αυτόματου πιλότου ώστε να προετοιμάζει τις εικόνες πριν προωθηθούν στο μοντέλο τεχνητής νοημοσύνης. Περισσότερες πληροφορίες για τις μεταμορφώσεις της εικόνας θα παρουσιαστούν σε επόμενο κεφάλαιο.

3.2.2 Το πακέτο data

Το πακέτο data εμπεριέχει το αρχείο car_data.py, όπου υπάρχουν δύο κλάσεις, η CarData και η CsvFile. Με την κλάση CarData δημιουργείται ένας πίνακας δεδομένων (pandas dataframe [17]) ο οποίος περιέχει τις κολώνες:

- Move: Η κίνηση του οχήματος
- Created_on: Η ώρα και η ημερομηνία καταχώρησης της εγγραφής στον πίνακα
- Speed: Η ταχύτητα του οχήματος (0-100%)
- Distance: Η μέτρηση του αισθητήρα απόστασης του οχήματος

Με την κλάση CsvFile δημιουργείται ένα αρχείο .csv στο τέλος της καταγραφής, όπου περιέχει όλα τα δεδομένα που έχουν καταγραφεί στον παραπάνω πίνακα.

3.2.3 Το πακέτο dir_manager

Στο πακέτο dir_manager περιέχεται η κλάση DirectoryManager. Η συγκεκριμένη κλάση είναι υπεύθυνη για τη δημιουργία φακέλου σε συγκεκριμένο μονοπάτι και με συγκεκριμένο όνομα. Το όνομα του φακέλου αποτελείται από το όνομα της πίστας που καταγράφεται σε συνδυασμό με την ημερομηνία και την ώρα της καταγραφής και έχει τη μορφή **όνομα πίστας/ημερομηνία-ώρα**. Μέσα στον φάκελο αποθηκεύονται οι εικόνες κατά την καταγραφή δεδομένων καθώς και το .csv αρχείο με τη βοήθεια της κλάσης CsvFile που παρουσιάστηκε παραπάνω.

3.2.4 Το πακέτο led

Το πακέτο led περιέχει το αρχείο single_led.py με την κλάση SingleLED. Η κλάση χρησιμοποιείται για τον χειρισμό των τριών λυχνιών LED πάνω στο όχημα. Δέχεται σαν είσοδο τα pins του Jetson με τα οποία συνδέονται οι λυχνίες και προσφέρει μεθόδους για να ανάβουν και να σβήνουν. Κατά την εκκίνηση του οχήματος ανάβουν και οι 3 λυχνίες σταδιακά και έπειτα σβήνουν.

```
def turnOn(self, led):
    self.gpio.output(led, True)

def turnOff(self, led):
    self.gpio.output(led, False)

def onStartCarLEDs(self):
    self.turnOn(const.green_LED_pin)
    time.sleep(1)
    self.turnOn(const.red_LED_pin)
    time.sleep(1)
    self.turnOn(const.blue_LED_pin)
    time.sleep(1)
    self.turnOff(const.green_LED_pin)
    self.turnOff(const.red_LED_pin)
    self.turnOff(const.blue_LED_pin)
```

Εικόνα 14: Μέθοδοι πακέτου led

3.2.5 Το πακέτο distance_sensor

Το πακέτο distance_sensor περιέχει το ομώνυμα αρχείο Python με την κλάση DistanceSensor. Η DistanceSensor κάνει extend την κλάση Thread της Python με σκοπό να εκτελείται σε νέο διαφορετικό thread από αυτό του κύριου προγράμματος.

Αρχικά στον constructor ορίζονται τα GPIO pins του Jetson, ένα που θα στέλνει σήμα στον αισθητήρα και ονομάζεται trigger pin (output) και ένα που θα λαμβάνει σήμα και ονομάζεται echo pin (input). Στη συνέχεια, όταν ξεκινάει το thread, καλούνται μέθοδοι με τις οποίες ο πομπός στέλνει το ηχητικό κύμα και ο δέκτης το λαμβάνει καταγράφοντας τους χρόνους αρχής και τέλους με σκοπό τον υπολογισμό της απόστασης, όπως αναφέρθηκε στο προηγούμενο κεφάλαιο. Πριν το τέλος κάθε επανάληψης γίνεται publish στο αντίστοιχο MQTT topic.

3.2.6 Το πακέτο car

Το πακέτο car έχει σαν περιεχόμενο το αρχείο car.py και την αντίστοιχη κλάση Car. Η κλάση αυτή χρησιμοποιείται για τον έλεγχο της κίνησης και της ταχύτητας του οχήματος. Αρχικά στον constructor ορίζονται τα pins που αντιστοιχούν από το Jetson στον οδηγό κινητήρα L298N και η αρχική ταχύτητα. Στη συνέχεια ακολουθεί μία μέθοδος η go

```
def go(self, backward_right, forward_left, backward_left, forward_right, direction, move):
    self.direction = direction
    self.move = move
    self.gpio.output(const.backward_right_pin, backward_right)
    self.gpio.output(const.forward_left_pin, forward_left)
    self.gpio.output(const.backward_left_pin, backward_left)
    self.gpio.output(const.forward_right_pin, forward_right)
```

Εικόνα 15: Μέθοδος κίνησης

που χρησιμοποιείται για να οριστεί η κίνηση του οχήματος και οι πιθανές τιμές της είναι backward, backward_left, backward_right, forward, foraward_left, forward_right και stop. Ανάλογα με την κίνηση ορίζεται αντίστοιχα η λειτουργία των κινητήρων.

Έπειτα υπάρχει η μέθοδος set_speed

```
def set_speed(self, speed):
    self.speed = speed
    self.pw1.ChangeDutyCycle(self.speed)
    self.pw2.ChangeDutyCycle(self.speed)
```

Εικόνα 16: Μέθοδος ορισμού ταχύτητας

με την οποία ορίζεται η ταχύτητα του οχήματος μέσω του ορισμού της τιμής του duty cycle των PWM pins. Το εύρος των τιμών της ταχύτητας είναι από 0 έως 100.

Οι υπόλοιπες μέθοδοι που υπάρχουν στην κλάση καλούν τις παραπάνω μεθόδους με συγκεκριμένα ορίσματα.

3.2.7 Το πακέτο camera

Στο πακέτο camera υπάρχει το αρχείο car_camera.py με την κλάση CarCamera. Η CarCamera είναι η πιο περίπλοκη κλάση των πακέτων. Κάνει extend και αυτή την κλάση Thread της Python ώστε να εκτελείται σε διαφορετικό thread ανεξάρτητα από το κυρίως πρόγραμμα. Στον constructor της αρχικοποιούνται διάφορες μεταβλητές , όπως η διεύθυνση όπου θα εμφανίζεται το βίντεο της κάμερας, ο φάκελος που θα

αποθηκεύονται οι εικόνες, ένας μετρητής για τις εικόνες κ.ά και ορίζεται ως παράμετρος ο τρόπος λειτουργίας της εφαρμογής ώστε να γίνουν οι κατάλληλες ενέργειες για καθεμία.

- Για τη λειτουργία της εγγραφής δεδομένων της κίνησης του οχήματος αρχικά καλείται από την CarData η δημιουργία του πίνακα δεδομένων. Στη συνέχεια βρίσκεται μία επαναληπτική δομή while όπου "τραβάει" συνεχόμενες εικόνες από την κάμερα και ανάλογα με τις μεταβλητές που έχουν οριστεί τις δημοσιεύει σε βίντεο και τις αποθηκεύει.

```
def setup_camera_recording(self):
    carData = CarData()
    self.df = carData.create_df()

    while True:
        image = self.camera.Capture()
        if self.video_output:
            self.output.Render(image)

        if self.rec:
            self.save_photo(image=image, carData=carData)
            self.counter += 1
```

Εικόνα 17: Μέθοδος έναρξης εγγραφής μέσω κάμερας

Όταν ξεκινήσει η καταγραφή της κίνησης του οχήματος ξεκινάει η αποθήκευση των φωτογραφιών με τη μέθοδο save_photo. Η μέθοδος αυτή προσθέτει μία εγγραφή στον πίνακα δεδομένων του CarData για κάθε φωτογραφία και στη συνέχεια την αποθηκεύει σε συγκεκριμένο φάκελο που έχει οριστεί αφού πρώτα επεξεργαστεί το μέγεθός της και ορίζοντάς το σε 224 x 224 pixels. Στη συνέχεια αυξάνεται ο μετρητής κατά 1.

```
def save_photo(self, image, carData):
    self.df = carData.addToDataFrame(self.df,
                                     move=self.car_move,
                                     speed=self.car_speed,
                                     distance=self.distance)
    # generate image name
    name = self.camera_save_files_path + self.created_dir + generate_image_name(self.counter, self.car_move,
                                                                                 self.car_speed)
    # convert CudaImage to np array
    image = jetson.utils.cudaToNumpy(image).astype(np.uint8)
    # read PIL image from array
    image = PIL.Image.fromarray(image)
    # resize image
    image = transforms.functional.resize(image, [224, 224])
    # save image
    image.save(name)
```

Εικόνα 18: Μέθοδος αποθήκευσης εικόνων

Το όνομα της κάθε φωτογραφίας είναι ένας συνδυασμός της τιμής του μετρητή, της κίνησης του οχήματος και της ταχύτητάς του. Οι εικόνες είναι αρχεία τύπου jpg.

```python
def generate_image_name(counter, car_move, car_speed):
    name = '/img_' + str(counter) + '_' + car_move + '_' + str(car_speed) + '.jpg'
    return name
```

Εικόνα 19: Μέθοδος ονόματος κάθε εικόνας

Κατά την ολοκλήρωση της εγγραφής από τον πίνακα δεδομένων παράγεται ένα .csv αρχείο μέσω της κλάσης CsvFile το οποίο αποθηκεύεται στον φάκελο με τις εικόνες και επαναφέρονται οι αρχικές τιμές των μεταβλητών.

```python
def stop_recording(self):
    self.rec = False
    #  Create and export csv from dataframe
    csv = CsvFile(csv_name=self.start_rec + '.csv')
    csv.export_file(self.df, self.camera_save_files_path + self.created_dir + '/')
    self.df = pd.DataFrame()
    self.counter = 0
    self.created_dir = ''
    self.start_rec = ''
```

Εικόνα 20: Μέθοδος παύσης εγγραφής

- Για τη λειτουργία εγγραφής δεδομένων στάσης του οχήματος η διαδικασία που ακολουθείται είναι η ίδια με εκείνη της εγγραφής δεδομένων κίνησης. Οι διαφορές είναι ότι πρέπει να οριστεί σαν παράμετρος στον constructor της κλάσης η λειτουργία και ότι σε αυτή τη λειτουργία δεν αποθηκεύεται σειρά φωτογραφιών αλλά μόνο μία φωτογραφία. Για τη λήψη κάθε εικόνας χρησιμοποιείται η μέθοδος take_pic η οποία μεταβάλει την τιμή της boolean μεταβλητής take_photo από false σε true.

```python
def setup_photo_camera(self):
    carData = CarData()
    self.df = carData.create_df()

    while True:
        # read the camera image
        image = self.camera.Capture()
        if self.take_photo:
            self.save_photo(image=image, carData=carData)
            self.take_photo = False
            self.counter += 1

def take_pic(self, created_dir, start_rec):
    self.created_dir = created_dir
    self.start_rec = start_rec
    self.take_photo = True
```

Εικόνα 21: Μέθοδος λήψης εικόνων στάσης

- Για τη λειτουργία του αυτόματου πιλότου είναι αναγκαίο να οριστεί η τιμή της παραμέτρου του constructor της κλάσης autopilot σε true. Η κλάση CarCamera χρησιμοποιείται από το κύριο πρόγραμμα στη συγκεκριμένη λειτουργία για να "διαβάζει" εικόνες από την κάμερα πράγμα που επιτυγχάνεται μέσω της μεθόδου read.

```
def read(self, showMoves=False, move=None, output_percent=None):
    image = self.camera.Capture()
    if showMoves:
        self.font.OverlayText(image, 50, 20, move + ' ' + "{:.2f}%".format(output_percent), 10, 10, self.font.White,
                              self.font.Gray40)
    if self.video_output:
        self.output.Render(image)
    return image
```

Εικόνα 22: Μέθοδος λήψης βίντεο για το autopilot

3.2.8 Τρόποι λειτουργίας εφαρμογής

Στον κεντρικό φάκελο της εφαρμογής υπάρχουν τρία αρχεία όπου το κάθε ένα συνδέεται με μία ξεχωριστή λειτουργία του οχήματος.

1. Η record_dataset.py που σχετίζεται με τη λειτουργία εγγραφής δεδομένων για την κίνηση του οχήματος
2. Η record_stop_images.py που σχετίζεται με τη λειτουργία εγγραφής δεδομένων όπου το όχημα σταματάει να κινείται
3. Η autopilot.py που σχετίζεται με τη λειτουργία της αυτόματου πιλότου όπου το όχημα κινείται μόνο του

3.3 Λειτουργικό σύστημα Fog Node

Το Fog Node χρησιμοποιεί σαν λειτουργικό σύστημα το **Ubuntu 22.04 LTS.** Το Ubuntu είναι ένα πλήρες λειτουργικό σύστημα (όπως τα Windows και το MacOS) βασισμένο όμως στο Linux. Διατίθεται ελεύθερα και υποστηρίζεται τόσο από μια πολύ μεγάλη κοινότητα χρηστών όσο και από επαγγελματίες.

4. Επικοινωνία με το δίκτυο

Για την επικοινωνία του αυτόνομου ηλεκτρικού οχήματος με το δίκτυο, ώστε να παρουσιάζονται διάφορες πληροφορίες που αφορούν το όχημα, σε σχεδόν "ζωντανό" χρόνο χρησιμοποιείται ένα από τα πιο γνωστά M2M πρωτόκολλα επικοινωνίας, το MQTT [18] . Μέσω του συγκεκριμένου πρωτοκόλλου μπορεί να πραγματοποιηθεί και απομακρυσμένος χειρισμός του οχήματος σε συγκεκριμένες περιπτώσεις.

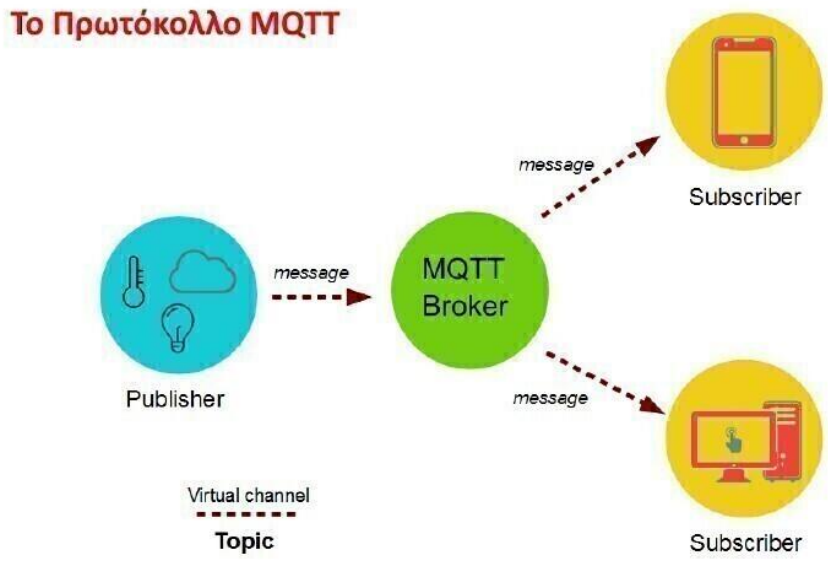
4.1 Το Πρωτόκολλο MQTT

Το Message Queuing Telemetry Transport (MQTT) είναι ένα πρωτόκολλο ανταλλαγής μηνυμάτων, πάνω από το πρωτόκολλο TCP/IP, το οποίο βασίζεται στην αρχιτεκτονική εκδότης/συνδρομητής (publish/subscribe). Το συγκεκριμένο πρωτόκολλο είναι ελαφρύ γι' αυτό χρησιμοποιείται σε συσκευές με περιορισμένους υπολογιστικούς πόρους.

Βασικά στοιχεία του MQTT είναι ο broker, ο publisher και ο subscriber.

- Ο broker αποτελεί τον ενδιάμεσο διαχειριστή και διακομιστή (server) του συστήματος των μηνυμάτων, ο οποίος φιλτράρει τα μηνύματα που δέχεται και τα προωθεί στους αντίστοιχους κόμβους. Γνωστοί brokers που υπάρχουν είναι ο Mosquitto , ο HiveMQ ο VernerMQ και άλλοι

- Ο publisher είναι ο κόμβος ο οποίος στέλνει μηνύματα στον broker με σκοπό την παράδοσή τους στο σωστό κόμβο subscriber
- Ο subscriber είναι ο κόμβος που συνδέεται στον broker για να λαμβάνει τα μηνύματα



Εικόνα 23: Λειτουργία MQTT

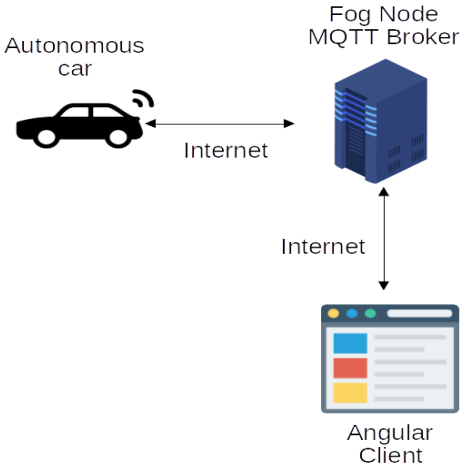
Για τη λειτουργία του MQTT χρησιμοποιούνται τα θέματα (topics) που είναι θεματικές ενότητες στις οποίες ένας ή περισσότεροι publishers στέλνουν μηνύματα . Στη συνέχεια οι subscribers εγγράφονται στα θέματα από τα οποία επιθυμούν να λαμβάνουν μηνύματα. Η επικοινωνία μέσω των θεμάτων πραγματοποιείται ασύγχρονα. Το κάθε θέμα είναι μία UTF-8 συμβολοσειρά (string) και χρησιμοποιείται από τον broker για την αποστολή των μηνυμάτων στους κατάλληλους subscribers. Κάθε θέμα περιέχει πολλαπλά θεματικά επίπεδα τα οποία διαχωρίζονται με "/".

Ακόμη ένα σημαντικό χαρακτηριστικό του MQTT είναι η ποιότητα των υπηρεσιών (Quality of Service – QoS) καθώς προσφέρει μεγάλη αξιοπιστία παράδοσης ενός μηνύματος ανάλογα με το κόστος της. Υπάρχουν 3 επίπεδα QoS:

- QoS-0 : Το μήνυμα αποστέλλεται μόνο μία φορά χωρίς να αναμένεται απάντηση από τον παραλήπτη και δε αποθηκεύονται για να αναμεταδοθούν
- QoS-1 : Το μήνυμα αποστέλλεται τουλάχιστον μία φορά επιτυχώς. Τα συγκεκριμένα μηνύματα αποθηκεύονται από τον αποστολέα μέχρι να λάβει επιβεβαίωση από τον παραλήπτη
- QoS-2 : Το μήνυμα θα σταλεί επιτυχημένα ακριβώς μία φορά. Αποτελεί την ασφαλέστερη επιλογή μετάδοσης μηνυμάτων αλλά και την πιο χρονοβόρα

4.2 Αρχιτεκτονικό διάγραμμα

Η σύνδεση του αυτόνομου ηλεκτρικού οχήματος μέσω MQTT με ένα Fog Node παρουσιάζεται στο παρακάτω διάγραμμα.



Εικόνα 24: Αρχιτεκτονική συστήματος

4.3 MQTT broker

Για την υλοποίηση του MQTT broker χρησιμοποιήθηκε το image eclise-mosquitto [19] του docker container το οποίο τρέχει στο Fog Node μέσω της τεχνολογίας Docker [20].

Το Docker(Ντόκερ) είναι μια πλατφόρμα λογισμικού ανοιχτού κώδικα που υλοποιεί εικονικοποίηση (Virtualization) σε επίπεδο λειτουργικού συστήματος. Ουσιαστικά το Docker προσφέρει αυτοματοποιημένες διαδικασίες για την ανάπτυξη εφαρμογών σε απομονωμένες Περιοχές Χρήστη (User Spaces) που ονομάζονται Software Containers. Το λογισμικό χρησιμοποιεί τεχνολογίες του πυρήνα του Linux όπως τα cgroups και οι χώροι ονομάτων πυρήνα (kernel namespaces), για να επιτρέπει σε ανεξάρτητα software containers να εκτελούνται στο ίδιο λειτουργικό σύστημα. Έτσι αποφεύγεται η χρήση επιπλέον υπολογιστικών πόρων που θα απαιτούσε μια εικονική μηχανή (virtual machine).

Για την αποστολή και λήψη μηνυμάτων μέσω MQTT από μία εφαρμογή ιστού (web application), η οποία τρέχει σε ένα πρόγραμμα περιήγησης (web browser) είναι απαραίτητη η ενεργοποίηση των WebSockets. Το WebSocket [21] είναι ένα πρωτόκολλο επικοινωνίας υπολογιστών που παρέχει full-dublex κανάλια επικοινωνίας μέσω μιας TPC/IP σύνδεσης.

Για την ενεργοποίηση του WebSocket πρωτοκόλλου χρησιμοποιήθηκε κατά την εκκίνηση του eclise-mosquitto image το παρακάτω αρχείο διαμόρφωσης (mosquitto.conf)

```
pid_file /var/run/mosquitto.pid
persistence true
persistence_location /var/lib/mosquitto/
log_dest file /var/log/mosquitto/mosquitto.log
include_dir /etc/mosquitto/conf.d
port 1883
protocol mqtt
listener 9001
protocol websockets
```

4.4 Σύνδεση αυτόνομου οχήματος με MQTT

Για τη σύνδεση του αυτόνομου οχήματος με τον MQTT broker χρησιμοποιήθηκε η βιβλιοθήκη paho-mqtt σε γλώσσα python.

Η υλοποίηση της σύνδεσης πραγματοποιείται στο πακέτο mqtt_connection μέσα στο αρχείο mqtt_car.py [22]. (Μέρος του κώδικα στηρίζεται στις ασκήσεις του κυρίου Βέργαδου στο πλαίσιο του μαθήματος "Λογισμικό για Έξυπνες Πόλεις, Λογισμικό Διαδικτύου των Πραγμάτων (IoT) και Εφαρμογές"). Στο αρχείο αυτό υπάρχει η κλάση MqttCar με την οποία πραγματοποιείται η σύνδεση του οχήματος στον MQTT broker καθώς και η διαδικασία για αποστολή και λήψη δεδομένων από τα topics.

Αρχικά στον constructor της κλάσης αρχικοποιούνται κάποιες μεταβλητές και πραγματοποιείται η σύνδεση με τον broker μέσω της μεθόδου _establish_mqtt_connection(). Μέσα σε αυτή τη μέθοδο δημιουργείται ένα object της κλάσης mqtt.Client() της βιβλιοθήκης paho-mqtt. Σε αυτό το object ορίζονται μέθοδοι που θα τρέξουν μετά τη σύνδεση, την αποστολή δεδομένων και τη λήψη δεδομένων.

```python
def _on_connect(self, client, userdata, flags, rc):
    print('Connected with result code ' + str(rc))
    client.subscribe(const.autopilot_topic)
    client.subscribe(const.car_speed_topic)

def _on_message(self, client, userdata, msg):
    if msg.topic == const.autopilot_topic:
        self.autopilot_status = str(msg.payload.decode('utf-8'))
        print(self.autopilot_status)
    elif msg.topic == const.car_speed_topic:
        self.speed_value = int(msg.payload.decode('utf-8'))
        print(self.speed_value)

def _establish_mqtt_connection(self):
    print('Trying to connect to MQTT server...')
    self.client = mqtt.Client()
    self.client.on_connect = self._on_connect
    self.client.on_message = self._on_message
    self.client.on_subscribe = self.on_subscribe
    self.client.connect(const.mqtt_hostname, const.mqtt_port)

def publish(self, msg, topic):
    self.client.publish(topic, msg, retain=False)
```

Εικόνα 25: Σύνδεση στον MQTT broker

Μετά την ολοκλήρωση της σύνδεσης τρέχει η μέθοδος _on_connect όπου πραγματοποιείται και συνδρομή (subscription) σε δύο topics. Κατά τη λήψη κάποιου μηνύματος από αυτά τα topics τρέχει η μέθοδος _on_message με την οποία ορίζονται μεταβλητές της κλάσεις με τις τιμές που εήφθησαν. Τέλος για την αποστολή μηνύματος από το όχημα στο δίκτυο καλείται η μέθοδος publish με παραμέτρους το topic και το ίδιο το μήνυμα που θα δημοσιευτεί.

4.5 MQTT topics

Τα MQTT topics που χρησιμοποιήθηκαν για αποστολή (publish) δεδομένων από το όχημα στον angular client είναι τα παρακάτω:

1. car/move: Αποστολή της κίνησης του οχήματος. Πιθανές τιμές: forward, backward, stop, forward_left, forward_right, backward_left, backward_right
2. car/speed/value: Αποστολή της ταχύτητας του οχήματος. Πιθανές τιμές: 0-100

3. car/mode: Αποστολή του τρόπου λειτουργίας του οχήματος. Πιθανές τιμές: 1 (καταγραφή δεδομένων), 2 (καταγραφή δεδομένων για στάση), 3 (αυτόματος πιλότος)
4. distance/value: Αποστολή της μέτρησης του αισθητήρα HC-SR04

Τα MQTT topics που χρησιμοποιήθηκαν για λήψη (subscribe) δεδομένων από τον angular client στο όχημα είναι τα παρακάτω:

1. car/speed/value: Ορισμός της ταχύτητας του οχήματος από τον browser. Πιθανές τιμές: 0-100
2. autopilot/status: Ορισμός της λειτουργίας του αυτόματου πιλότου. Πιθανές τιμές: ON, OFF
3. obj_detection/status: Ορισμός της λειτουργίας εντοπισμού αντικειμένων. Πιθανές τιμές: ON, OFF

Όλα τα μηνύματα που στέλνονται στα MQTT topics έχουν QoS-0.

Κεφάλαιο 5 - Απεικόνιση στο δίκτυο

Στο κεφάλαιο αυτό παρουσιάζεται ο τρόπος απεικόνισης του ηλεκτρικού οχήματος στο δίκτυο μέσω μιας web εφαρμογής και οι απομακρυσμένες λειτουργίες του.

5.1 Δημιουργία και λειτουργία Web εφαρμογής

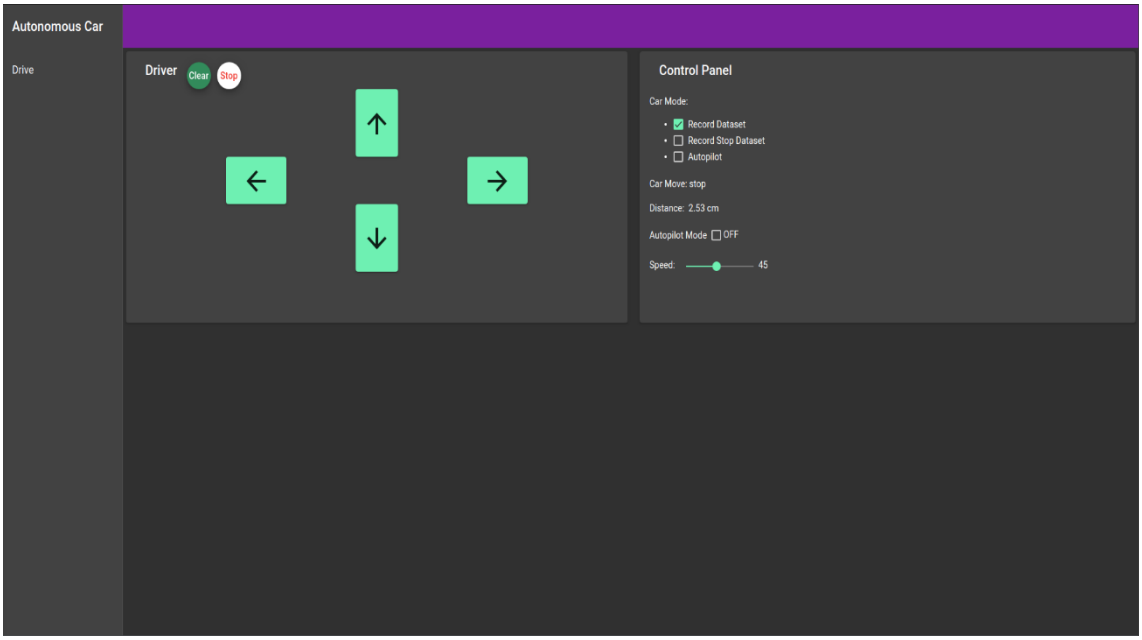
Για τη δημιουργία της web εφαρμογής χρησιμοποιήθηκε το framework της Angular [23]. Η Angular είναι μια πλατφόρμα ανάπτυξης για τη δημιουργία εξελιγμένων single-page εφαρμογών η οποία στηρίζεται στη γλώσσα προγραμματισμού Typescript [24] και HTML [25] και έχει αναπτυχθεί από την Google.

Βασικά στοιχεία της Angular αποτελούν τα components και κάθε εφαρμογή που αναπτύσσεται στηρίζεται σε μία δομή η οποία ξεκινάει από ένα βασικό component(root) και περιέχει πολλά εμφωλευμένα components (children).

Η εφαρμογή που αναπτύχθηκε για την απεικόνιση του αυτόνομου οχήματος και τις απομακρυσμένες λειτουργίες αποτελείται από 3 components:

- Το app component που αποτελεί το βασικό component της εφαρμογής και περιέχει όλα τα υπόλοιπα
- Το main-nav component το οποίο περιέχει την επικεφαλίδα και την μπάρα πλοήγησης (navigation sidebar) της εφαρμογής
- Το drive component το οποίο περιέχεται στο main-nav και απεικονίζει την κατάσταση και τις λειτουργίες του οχήματος

Η εμφάνιση της εφαρμογής παρουσιάζεται στην εικόνα που ακολουθεί.



Εικόνα 26: Ιστοσελίδα οχήματος

Κατά την εκκίνηση της εφαρμογής πραγματοποιείται σύνδεση της εφαρμογής στον MQTT broker μέσω του WebSockets πρωτοκόλλου, όπως περιγράφεται στο προηγούμενο κεφάλαιο, μέσω της βιβλιοθήκης ngx-mqtt [26] που υποστηρίζει η Angular. Απαραίτητος είναι ο ορισμός κάποιων μεταβλητών που αφορούν τον MQTT broker ώστε να "εντοπιστεί" από την εφαρμογή.

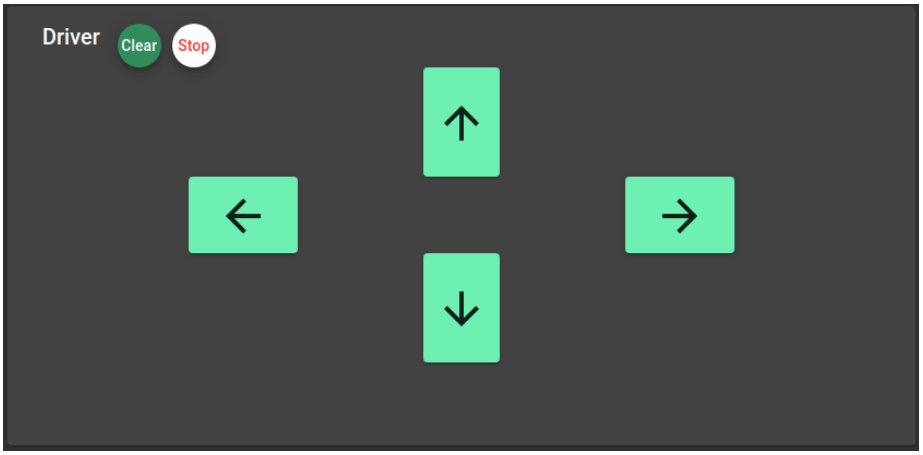
```
export const MQTT_SERVICE_OPTIONS: IMqttServiceOptions = {
 hostname: 'localhost',
  protocol: 'ws',
 port: 9001,
 path: '/mqtt'
};
```

Και αφορούν το hostname και το port του broker, το πρωτόκολλο σύνδεσης καθώς και το path όπου θα βρίσκονται τα topics μέσω των οποίων θα πραγματοποιηθεί η επικοινωνία.

Μετά την επιτυχημένη σύνδεση στον MQTT broker η εφαρμογή θα κάνει εγγραφή στα αντίστοιχα topics ώστε να δέχεται πληροφορίες τις οποίες στέλνει το όχημα με σκοπό να ενημερώσει την εφαρμογή σε πραγματικό χρόνο.

Το κύριο μέρος της εφαρμογής αποτελείται από 2 πάνελ:

1. Το πάνελ του οδηγού (Driver) που σχετίζεται με την οδήγηση του οχήματος. Πιο συγκεκριμένα περιέχει βελάκια τα οποία αλλάζουν χρώμα ανάλογα με την κατεύθυνση του οχήματος καθώς και 2 κουμπιά, "Stop" και "Clear" που χρησιμοποιούνται για την απεικόνιση όταν το όχημα σταματάει και για τον καθαρισμό των δεδομένων της εφαρμογής αντίστοιχα. Οι πληροφορίες αυτές λαμβάνονται μέσω του topic "car/move"



Εικόνα 27: Πάνελ οδηγού

2. Τον πίνακα ελέγχου (Control Panel), ο οποίος απεικονίζει διάφορες πληροφορίες για το όχημα και ενεργοποιεί ορισμένες λειτουργίες του. Πιο συγκεκριμένα περιέχει:

Εικόνα 28: Πίνακας ελέγχου

- Ένδειξη για τον τρόπο λειτουργίας του οχήματος (Car Mode), η οποία μπορεί να πάρει τις τιμές “Record Dataset”, “Record Stop Dataset” & “Autopilot” μέσω του topic “car/mode”
- Ένδειξη για την κίνηση του οχήματος (Car Move), μέσω topic “car/move”
- Ένδειξη για την τιμή του αισθητήρα απόστασης (Distance) σε cm μέσω του topic “distance/value”
- Πλαίσιο ελέγχου για τη λειτουργία αυτόματου πιλότου (Autopilot Mode). Το πλαίσιο παρέχει τη δυνατότητα ενεργοποίησης του αυτόματου πιλότου από το χρήστη της εφαρμογής μέσω της αποστολής δεδομένων στο topic “autopilot/status”
- Ολισθητή όπου απεικονίζεται η ταχύτητα του οχήματος (0 – 100%). Ο ολισθητής παρέχει τη συνατότητα στο χρήστη να ελέγχει την ταχύτητα του οχήματος μέσω της αποστολής δεδομένων στο topic “car/speed/value”

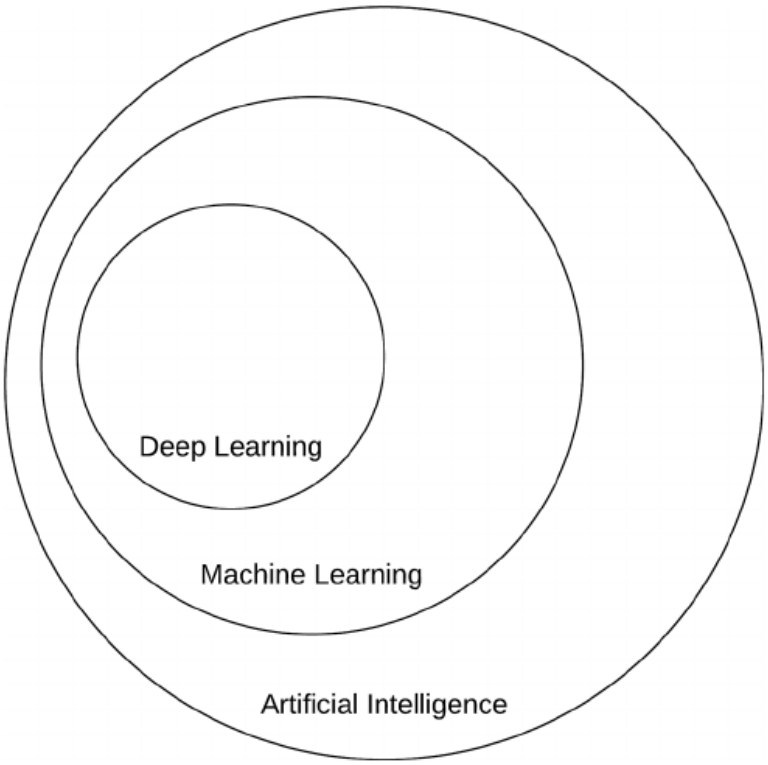
6. Τεχνητή Νοημοσύνη & Deep Learning

Ένα από τα φλέγοντα θέματα της εποχής μας και ένας από τους πιο υποσχόμενους τομείς της πληροφορικής αποτελεί η Τεχνητή Νοημοσύνη. Καθημερινά η τεχνητή νοημοσύνη εισέρχεται σε όλο και περισσότερους τομείς δίνοντας λύσεις σε προβλήματα που απασχολούν τον άνθρωπο για πολλά χρόνια.

Στο κεφάλαιο αυτό θα παρουσιαστούν εν συντομία μερικά από τα βασικά δομικά στοιχεία της τεχνητής νοημοσύνης και των νευρωνικών δικτύων, όπως οι συνελίξεις και τα πυκνά στρώματα, καθώς και οι συναρτήσεις ενεργοποίησής τους και οι τρόποι τακτοποίησής τους.

6.1 Τεχνητή Νοημοσύνη

Ως τεχνητή νοημοσύνη(Artificial Intelligence) ορίζεται η ικανότητα μιας μηχανής να αναπαράγει αυτόνομα ανώτερες γνωστικές λειτουργίες του ανθρώπου. Ορισμένες από αυτές τις λειτουργίες είναι η μάθηση, η προσαρμοστικότητα, η εξαγωγή συμπερασμάτων, η κατανόηση του περιβάλλοντος κτλ. Αποτελεί σημείο τομής πολλών επιστημών με στόχο τη σύνθεση ευφυούς συμπεριφοράς με στοιχεία συλλογιστικής, μάθησης και προσαρμογής στο περιβάλλον, η οποία εφαρμόζεται σε μηχανές ή υπολογιστές ειδικής κατασκευής.



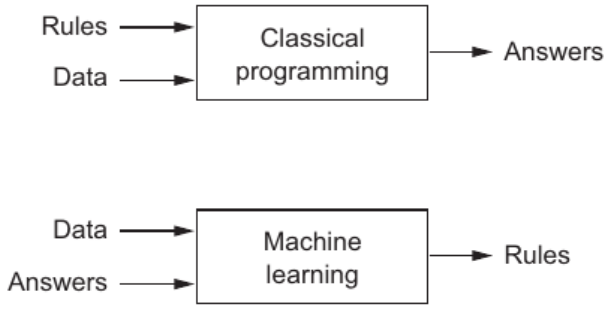
Εικόνα 29: Τομείς τεχνητής νοημοσύνης *[27]*

6.2 Μηχανική Μάθηση

Μία υποενότητα της τεχνητής νοημοσύνης, όπως φαίνεται και στην παραπάνω εικόνα, αποτελεί η μηχανική μάθηση (Machine Learning). Η Μηχανική Μάθηση ορίζεται ως το φαινόμενο κατά το οποίο ένα σύστημα βελτιώνει την απόδοσή του κατά την εκτέλεση μιας συγκεκριμένης εργασίας, χωρίς να

υπάρχει ανάγκη να προγραμματιστεί εκ νέου. Αποτελεί δηλαδή την επιστήμη να προγραμματίζεις υπολογιστές ώστε εκείνοι να έχουν τη δυνατότητα να "μάθουν" από διάφορα δεδομένα.

Στον κλασσικό προγραμματισμό ο άνθρωπος εισάγει κανόνες (έναν αλγόριθμο) και δεδομένα, τα οποία προωθούνται μέσω των κανόνων και προκύπτουν απαντήσεις. Στης περίπτωση της μηχανικής μάθησης, ο άνθρωπος εισάγει δεδομένα καθώς και τις αναμενόμενες απαντήσεις που προκύπτουν από τα δεδομένα και σαν αποτέλεσμα δημιουργούνται οι κανόνες. Αυτοί οι κανόνες μπορούν να εφαρμοστούν σε νέα δεδομένα για να παραχθούν απαντήσεις.



Εικόνα 30: Διαφορά μηχανικής μάθησης - κλασσικού προγραμματισμού

Επομένως ένα σύστημα μηχανικής μάθησης έχει εκπαιδευτεί και δεν έχει προγραμματιστεί. Για την εκπαίδευση του συστήματος δίνονται πολλά παραδείγματα σχετικά με μία εργασία και εκείνο βρίσκει μια στατιστική δομή πάνω σε αυτά τα παραδείγματα που του επιτρέπει να δημιουργήσει κανόνες για την αυτοματοποίηση της εργασίας αυτής.

6.3 Νευρωνικά Δίκτυα - Βαθιά Μάθηση

Η βαθιά μάθηση (Deep Learning) είναι ένα υποσύνολο της μηχανικής μάθησης όπως φαίνεται και στην παραπάνω εικόνα και αποτελεί μία οικογένεια αλγορίθμων νευρωνικών δικτύων. Στη βαθιά μάθηση αρκετά «επίπεδα» απλών μονάδων επεξεργασίας συνδέονται σε ένα δίκτυο, το ένα πίσω από το άλλο, με αποτέλεσμα η είσοδος στο σύστημα να διέρχεται διαδοχικά μέσα από κάθε ένα από αυτά.

Αυτό το βάθος επιτρέπει στο δίκτυο να μαθαίνει πιο σύνθετες δομές, χωρίς να απαιτούνται μεγάλες, μη ρεαλιστικές, ποσότητες δεδομένων.

Τα νευρωνικά δίκτυα αποτελούν μία κατηγορία αλγορίθμων μηχανικής μάθησης τα οποία "μαθαίνουν" από τα δεδομένα και ειδικεύονται στην αναγνώριση μοτίβου. Ως νευρωνικό δίκτυο ορίζεται ένα δίκτυο από απλούς υπολογιστικούς κόμβους (νευρώνες) οι οποίοι είναι συνδεδεμένοι μεταξύ τους. Πηγή έμπνευσης των νευρωνικών δικτύων αποτελεί η επεξεργασία των οπτικών πληροφοριών στον εγκέφαλο, δηλ. των πληροφοριών που εισέρχονται μέσω των ματιών και συλλαμβάνονται από τον αμφιβληστροειδή χιτώνα, προχωρούν μέσα στο οπτικό νεύρο και φτάνουν στον εγκέφαλο.

Βασικό δομικό στοιχείο ενός νευρωνικού δικτύου αποτελούν οι νευρώνες. Ρόλος κάθε νευρώνα είναι η εκτέλεση ενός υπολογισμού και η παραγωγή μιας εξόδου χρησιμοποιώντας ως είσοδο δεδομένα που προέρχονται από το περιβάλλον ή από άλλους νευρώνες.

Τα νευρωνικά δίκτυα αποτελούνται από πολλά συνδεδεμένα επίπεδα νευρώνων, γνωστά και ως στρώματα. Οι νευρώνες σε ένα στρώμα συνδέονται με τους νευρώνες του προηγούμενου και του επόμενου στρώματος μέσω "βαρών" ή "συνδέσμων". Κάθε στρώμα μετατρέπει την είσοδο που λαμβάνει από το προηγούμενο στρώμα σε έξοδο, η οποία περνά στο επόμενο στρώμα.

6.4 Συνάρτηση Ενεργοποίησης

Η συνάρτηση ενεργοποίησης (ή αλλιώς συνάρτηση ενεργοποίησης νευρώνων) είναι μια συνάρτηση που εφαρμόζεται σε κάθε νευρώνα του δικτύου, και καθορίζει την έξοδο του νευρώνα, δηλαδή την τιμή που προκύπτει μετά την εφαρμογή του συνδυασμού των εισόδων του νευρώνα.

Οι συναρτήσεις ενεργοποίησης είναι σημαντικές για τη λειτουργία των νευρωνικών δικτύων καθώς επιτρέπουν την εισαγωγή μη γραμμικοτήτων στο μοντέλο, προσδίδοντας την ικανότητα για πιο σύνθετη αναπαράσταση των δεδομένων. Οι συναρτήσεις ενεργοποίησης επιτρέπουν επίσης την προσθήκη μη γραμμικών στρωμάτων στο δίκτυο, καθορίζοντας το πεδίο των δυνατών συναρτήσεων που μπορούν να προσεγγίσουν. Ορισμένες από τις κυριότερες συναρτήσεις ενεργοποίησης που χρησιμοποιούνται σε νευρωνικά δίκτυα είναι οι εξής:

- Συνάρτηση Σιγμοειδούς Ενεργοποίησης (Sigmoid Activation Function): Είναι μια συνάρτηση που περιορίζεται στο διάστημα (0,1) και είναι κυρίως χρήσιμη για προβλήματα δυαδικής ταξινόμησης και προβλήματα που αφορούν πιθανότητες.
  Μαθηματική μορφή: σ(x) = 1 / (1 + exp(-x))
- Συνάρτηση Ενεργοποίησης ReLU (Rectified Linear Unit Activation Function): Είναι μια μη γραμμική συνάρτηση που είναι απλή και αποτελεσματική, ενεργοποιώντας τον νευρώνα μόνο αν η είσοδος του είναι θετική, αλλιώς τον απενεργοποιεί.
  Μαθηματική μορφή: f(x) = max(0, x)
- Συνάρτηση ενεργοποίησης tangent (ή tangent hyperbolic): Είναι μια μη γραμμική συνάρτηση που χρησιμοποιείται σε νευρωνικά δίκτυα. Παρέχει μια συμμετρική και μη γραμμική απόκριση, με το εύρος τιμών να είναι από -1 έως 1. Χρησιμοποιείται συχνά σε δίκτυα που απαιτούν κλιμάκωση των τιμών στο εύρος (-1, 1), όπως σε προβλήματα που αφορούν τον έλεγχο των καταστάσεων του δικτύου (state control) ή την εξαγωγή πληροφορίας (feature extraction) από την είσοδο του δικτύου.
  Μαθηματική μορφή: f(x) = (exp(x) - exp(-x)) / (exp(x) + exp(-x))

6.5 Συνάρτηση κόστους

Η συνάρτηση κόστους ή "loss function" στα νευρωνικά δίκτυα είναι μια μαθηματική συνάρτηση που χρησιμοποιείται για την αξιολόγηση της απόδοσης του μοντέλου κατά την διάρκεια της εκπαίδευσης του. Σκοπός της είναι να μετρήσει τον βαθμό της διαφοράς μεταξύ των προβλέψεων του μοντέλου και των ετικετών των δεδομένων εκπαίδευσης.

Η επιλογή της κατάλληλης συνάρτησης κόστους εξαρτάται από τον τύπο του προβλήματος που επιχειρείται να λυθεί και την αρχιτεκτονική του νευρωνικού δικτύου. Συνήθως, η συνάρτηση κόστους είναι μια μαθηματική έκφραση που συνδυάζει τις προβλέψεις του μοντέλου με τις τιμές των ετικετών, προκειμένου να αξιολογηθεί η ποιότητα των προβλέψεων.

Σκοπός της εκπαίδευσης του νευρωνικού δικτύου είναι να βρει την ελάχιστη τιμή της συνάρτησης κόστους, δηλαδή να βελτιστοποιήσει την απόδοση του μοντέλου. Η διαδικασία αυτή περιλαμβάνει την προσαρμογή των βαρών του νευρωνικού δικτύου.

Ορισμένες από τις πιο γνωστές συναρτήσεις κόστους ακολουθούν παρακάτω.

6.5.1 Συνάρτηση Μέσου Τετραγωνικού Σφάλματος

Η συνάρτηση κόστους μέσου τετραγωνικού σφάλματος (Mean Squared Error - MSE) είναι μια από τις πιο κοινές συναρτήσεις κόστους που χρησιμοποιούνται στα νευρωνικά δίκτυα για προβλήματα regression. Χρησιμοποιείται για να υπολογίσει το μέσο τετραγωνικό σφάλμα μεταξύ των προβλέψεων του μοντέλου και των πραγματικών τιμών των δεδομένων εκπαίδευσης.

Η MSE ορίζεται ως εξής: MSE(y, t) = 1/N * Σ((y - t)^2)

όπου y είναι οι προβλέψεις του μοντέλου, t είναι οι πραγματικές τιμές των δεδομένων εκπαίδευσης, και N είναι ο συνολικός αριθμός των προβλέψεων.

Η MSE μετρά την τετραγωνική απόκλιση μεταξύ των προβλέψεων του μοντέλου και των πραγματικών τιμών. Έχει το πλεονέκτημα ότι είναι ευαίσθητη στις μεγάλες αποκλίσεις, καθώς το τετράγωνο ενισχύει τις μεγάλες τιμές σφάλματος. Αυτό μπορεί να καταστήσει το MSE κατάλληλο για προβλήματα όπου θέλουμε να επιδιώξουμε μικρή απόκλιση από τις πραγματικές τιμές, όπως για παράδειγμα στην πρόβλεψη αριθμητικών τιμών

6.5.2 Συνάρτηση Διασταυρούμενης Εντροπίας

Η συνάρτηση κόστους διασταυρούμενης εντροπίας (Cross-Entropy) χρησιμοποιείται σε προβλήματα ταξινόμησης στα νευρωνικά δίκτυα. Συγκεκριμένα, χρησιμοποιείται στην περίπτωση προβλημάτων πολυκατηγορικής ταξινόμησης, δηλαδή όταν έχουμε περισσότερες από δύο κατηγορίες για την ταξινόμηση των δειγμάτων. Η συνάρτηση διασταυρούμενης εντροπίας εκφράζει τον βαθμό "απόκλισης" μεταξύ της πραγματικής κατηγορίας ενός δείγματος και των προβλέψεων του νευρωνικού δικτύου για αυτήν την κατηγορία. Υψηλότερη τιμή της συνάρτησης διασταυρούμενης εντροπίας αντιστοιχεί σε μεγαλύτερη απόκλιση ανάμεσα στις προβλέψεις του μοντέλου και της πραγματικής κατηγορίας του δείγματος, ενώ χαμηλότερη τιμή αντιστοιχεί σε μικρότερη απόκλιση.

Η διασταυρούμενη εντροπία ορίζεται ως εξής για έναν δυαδικό ταξινομητή (binary classifier): Cross Entropy(y, t) = - t * log(y) - (1 - t) * log(1 - y)

όπου y είναι οι προβλέψεις του μοντέλου, t είναι οι πραγματικές τιμές των δεδομένων εκπαίδευσης (π.χ. 0 ή 1), και log είναι ο φυσικός λογάριθμος.

6.6 Μέθοδοι βελτιστοποίησης

Οι μέθοδοι βελτιστοποίησης (optimization methods) είναι αλγόριθμοι ή τεχνικές που χρησιμοποιούνται στην εκπαίδευση των νευρωνικών δικτύων για να βρουν τις βέλτιστες τιμές των παραμέτρων του μοντέλου, έτσι ώστε να ελαχιστοποιηθεί η συνάρτηση κόστους που χρησιμοποιείται για την αξιολόγηση του μοντέλου.

Οι μέθοδοι βελτιστοποίησης επιτρέπουν τον αυτόματο υπολογισμό των βαρών του νευρωνικού δικτύου, προσαρμόζοντας τα βάρη κατά τη διάρκεια της εκπαίδευσης, έτσι ώστε να μειώνεται η τιμή της συνάρτησης κόστους.

Ορισμένες από τις βασικές μεθόδους βελτιστοποίησης ακολουθούν παρακάτω.

6.6.1 Αλγόριθμος Στοχαστικής Κατάβασης

Ο αλγόριθμος της στοχαστικής κατάβασης (stochastic gradient descent ή SGD) είναι ένας από τους πιο διαδεδομένους αλγορίθμους βελτιστοποίησης που χρησιμοποιούνται στη μηχανική μάθηση και την εκπαίδευση των νευρωνικών δικτύων.

Η ιδέα της στοχαστικής κατάβασης είναι να ενημερώνει τα βάρη ενός μοντέλου κατά μικρά βήματα, βασιζόμενος σε ένα τυχαίο δείγμα από τα δεδομένα εκπαίδευσης αντί να υπολογίζει το κόστος και να ενημερώνει τα βάρη για κάθε δείγμα των δεδομένων, όπως συμβαίνει στους παραδοσιακούς αλγορίθμους βελτιστοποίησης. Η τυχαιότητα εισάγεται στον αλγόριθμο με την επιλογή ενός τυχαίου δείγματος από τα δεδομένα εκπαίδευσης σε κάθε επανάληψη του αλγορίθμου.

Ο αλγόριθμος SGD έχει το πλεονέκτημα της απλότητας και της αποδοτικότητας στον χειρισμό μεγάλων συνόλων δεδομένων, καθώς απαιτεί λιγότερους υπολογιστικούς πόρους σε σχέση με άλλους αλγορίθμους.

6.6.2 Προσαρμοστική Εκτίμηση Ροπών

Η μέθοδος βελτιστοποίησης προσαρμοστικής εκτίμησης ροπών (Adaptive Moment Estimation - ADAM) είναι μια αποτελεσματική μέθοδος βελτιστοποίησης που χρησιμοποιείται στην εκπαίδευση νευρωνικών δικτύων. Είναι μια συνδυαστική μέθοδος που συνδυάζει στοιχεία από άλλους αλγορίθμους βελτιστοποίησης, όπως ο αλγόριθμος Momentum και ο αλγόριθμος RMSprop.

Ο αλγόριθμος ADAM χρησιμοποιεί δύο κύριες έννοιες για τον υπολογισμό των ενημερώσεων των παραμέτρων του μοντέλου κατά τη διάρκεια της εκπαίδευσης: τη μέση των προηγούμενων τετραγώνων των γραμμικών βαθμών των παραμέτρων (mean squared gradient) και τη μέση των προηγούμενων τετραγώνων των αρθρωτών βαθμών των παραμέτρων (mean squared momentum).

Συγκεκριμένα, η μέθοδος ADAM χρησιμοποιεί τις παρακάτω ανανεώσεις για τον υπολογισμό των παραμέτρων του μοντέλου:

1. Κίνητρο πρώτης τάξης (Momentum): Αναπαριστά την εκθετική κίνηση της μέσης των προηγούμενων τιμών των γραμμικών βαθμών των παραμέτρων. Χρησιμοποιείται για να καθορίσει την κατεύθυνση της ενημέρωσης των παραμέτρων και βοηθάει τον αλγόριθμο να ξεφύγει από τοπικά ελάχιστα.
2. Κίνητρο δεύτερης τάξης (RMSprop): Αναπαριστά την εκθετική κίνηση της μέσης των προηγούμενων τιμών των αρθρωτών βαθμών των παραμέτρων. Χρησιμοποιείται για να καθορίσει την έλλειψη ή την υπερβολή της ενημέρωσης των παραμέτρων.

6.7 Συνελικτικά Νευρωνικά Δίκτυα

Τα συνελικτικά νευρωνικά δίκτυα (Convolutional Neural Networks ή CNNs) είναι ένα είδος νευρωνικών δικτύων που είναι ειδικά σχεδιασμένα για την επεξεργασία εικόνων, αλλά επίσης ευρέως χρησιμοποιούνται σε άλλα πεδία, όπως η επεξεργασία σήματος, τον ήχο, τα βίντεο, και την αναγνώριση αντικειμένων.

Οι βασικές ιδέες που βρίσκονται πίσω από τα CNNs είναι οι εξής:

1. **Συνέλιξη:** Η βασική λειτουργία των CNNs είναι η συνέλιξη. Κατά τη συνέλιξη, ένα προκαθορισμένο φίλτρο (ή πυρήνας) ολισθαίνει πάνω από την εικόνα, υπολογίζοντας το εσωτερικό γινόμενο των τιμών του φίλτρου και της περιοχής της εικόνας. Αυτό έχει ως αποτέλεσμα τη δημιουργία ενός νέου χάρτη χαρακτηριστικών που αναδεικνύει τον τοπικό προσδιορισμό των χαρακτηριστικών.
2. **Συνάψεις (Strides):** Οι συνάψεις καθορίζουν πόσο "βήμα" πραγματοποιεί το φίλτρο κατά τη συνέλιξη. Χρησιμοποιώντας μεγαλύτερες συνάψεις, μπορεί να επιτευχθεί μείωση της διάστασης του εξόδου, ενώ μικρότερες συνάψεις επιτρέπουν μεγαλύτερη αναλυτική πληροφορία.
3. **Φασματική ακτίνωση (Padding):** Η προσθήκη μηδενικών περιγράφεται ως φασματική ακτίνωση. Αυτό μπορεί να βοηθήσει στη διατήρηση των διαστάσεων της εισόδου μετά τη συνέλιξη, προσφέροντας έτσι περισσότερη πληροφορία στο δίκτυο.
4. **Υποδειγματοληψία (Pooling):** Οι λειτουργίες υποδειγματοληψίας, όπως η μέγιστη υποδειγματοληψία, βοηθούν στη μείωση της διάστασης των χαρακτηριστικών, ενώ διατηρούν τα βασικά χαρακτηριστικά. Αυτό συμβάλλει στη μείωση του αριθμού των παραμέτρων και της υπολογιστικής πολυπλοκότητας.
5. **Πλήρως συνδεδεμένα επίπεδα:** Τα τελευταία στάδια ενός CNN μπορεί να περιλαμβάνουν πλήρως συνδεδεμένα επίπεδα, όπως σε κανονικά νευρωνικά δίκτυα. Αυτά τα επίπεδα λειτουργούν ως ταξινομητές για τις χαρακτηριστικές αναπαραστάσεις που έχουν εξάγει τα προηγούμενα επίπεδα.

Η διαδικασία εκπαίδευσης των CNNs περιλαμβάνει τη χρήση μεγάλων συνόλων δεδομένων με ετικέτες, όπως το ImageNet, για να ρυθμίσει τις παραμέτρους του δικτύου έτσι ώστε να αναγνωρίζει αποτελεσματικά τα πρότυπα και τις κατηγορίες που θέλουμε.

Τα συνελικτικά νευρωνικά δίκτυα έχουν επανασχεδιαστεί και εξελιχθεί με πολλές παραλλαγές για να ανταποκριθούν σε διάφορες απαιτήσεις εργασιών μηχανικής όρασης.

6.8 Προεκπαιδευμένα Συνελικτικά Νευρωνικά Δίκτυα

Τα προεκπαιδευμένα συνελικτικά νευρωνικά δίκτυα αναφέρονται σε μοντέλα CNN που έχουν εκπαιδευτεί σε μεγάλα σύνολα δεδομένων, όπως το ImageNet, προτού παραμετροποιηθούν για συγκεκριμένες εργασίες. Τα προεκπαιδευμένα μοντέλα αυτά έχουν την ικανότητα να εξάγουν πολύπλοκες και βαθιές αναπαραστάσεις από εικόνες, καθιστώντας τα κατάλληλα για πολλές εργασίες όρασης υπολογιστών.

Τα προεκπαιδευμένα μοντέλα μπορούν να χρησιμοποιηθούν σε διάφορες προκλήσεις, όπως η ελλιπής διαθεσιμότητα δεδομένων, η ανάγκη για υψηλή απόδοση ή η εξοικονόμηση χρόνου εκπαίδευσης. Είναι ευρέως διαθέσιμα και μπορούν να χρησιμοποιηθούν μέσω βιβλιοθηκών όπως TensorFlow, PyTorch και άλλες. Οι δύο πιο δημοφιλείς αρχιτεκτονικές προεκπαιδευμένων μοντέλων είναι το VGG και το ResNet. Υπάρχουν επίσης άλλες αρχιτεκτονικές όπως το Inception, το MobileNet, το EfficientNet και πολλές άλλες.

Παραδείγματα κάποιων δημοφιλών προεκπαιδευμένων μοντέλων περιλαμβάνουν:

1. **VGG (Visual Geometry Group):** Ένα βαθύ μοντέλο με πολλά συνελικτικά επίπεδα, γνωστό για την απλή του αρχιτεκτονική. Παρέχει καλή απόδοση, αλλά μπορεί να είναι πιο βαρύ σε όρους παραμέτρων.
2. **ResNet (Residual Network):** Ένα μοντέλο που χρησιμοποιεί συνδέσεις επιζωοτίας για την εξίσωση των προτύπων προς τα πίσω, βοηθώντας στην εξάλειψη της εξαφάνισης των κλίσεων. Είναι βαθύ και αποδοτικό.
3. **Inception (GoogLeNet):** Ένα μοντέλο που χρησιμοποιεί πολλές διαφορετικές πυραμίδες συνελικτικών φίλτρων παράλληλα για την ανίχνευση διαφορετικών κλιμάκων προτύπων.
4. **MobileNet:** Ένα ελαφρύ μοντέλο που στοχεύει στην αποδοτική χρήση υπολογιστικών πόρων, κατάλληλο για φορητές συσκευές.
5. **EfficientNet:** Ένα μοντέλο που συνδυάζει αποδοτικότητα και απόδοση, βελτιστοποιώντας τα βάρη, το βάθος και το πλάτος του δικτύου.

Αυτά τα προεκπαιδευμένα μοντέλα μπορούν να χρησιμοποιηθούν απευθείας ή να προσαρμοστούν για συγκεκριμένες εργασίες, όπως αναγνώριση αντικειμένων, ανίχνευση ανωμαλιών, σημειογραφία, μεταφράσεις, συνθετική οράσεις και άλλα.

6.9 ResNet (Residual Network)

Το ResNet (Residual Network) είναι ένα βαθύ συνελικτικό νευρωνικό δίκτυο που εισήγαγε την έννοια των συνδέσεων επιζωοτίας (skip connections) για την αντιμετώπιση του προβλήματος της εξαφάνισης των κλίσεων κατά την εκπαίδευση βαθύτερων νευρωνικών δικτύων. Αναπτύχθηκε από τους Kaiming He, Xiangyu Zhang, Shaoqing Ren και Jian Sun το 2015 [28], και αποτελεί ένα από τα πρώτα προεκπαιδευμένα μοντέλα που επιτυγχάνουν πολύ μεγάλο βάθος.

Το βασικό στοιχείο που καθιστά τα ResNet ξεχωριστά είναι οι συνδέσεις επιζωοτίας ή "residual connections". Αντί να προσπαθούμε να εκπαιδεύσουμε κάθε στρώμα να αντιστοιχίζει απευθείας την είσοδο στην έξοδο, το ResNet αποδίδει έμφαση στην εκπαίδευση των διαφορών, ή "residuals", μεταξύ των επιπέδων. Συγκεκριμένα, για ένα επίπεδο που λαμβάνει είσοδο x και παράγει έξοδο H(x), η συνδεση επιζωοτίας προσθέτει το x στην έξοδο, δημιουργώντας την εξίσωση H(x) + x. Αυτή η προσέγγιση βοηθά στη διατήρηση της πληροφορίας κατά την προώθηση προς τα εμπρός και ελαχιστοποιεί το πρόβλημα της εξαφάνισης των κλίσεων, δίνοντας τη δυνατότητα για βαθύτερα δίκτυα.

Το αρχικό ResNet περιλαμβάνει διάφορες παραλλαγές με διαφορετικό αριθμό επιπέδων (18, 34, 50, 101, 152) ανάλογα με το βάθος του δικτύου. Η πιο δημοφιλής έκδοση είναι το ResNet-50, το οποίο αποτελείται από 50 συνελικτικά και πλήρως συνδεδεμένα επίπεδα. Παρακάτω παρουσιάζεται η γενική δομή του ResNet:

1. **Εισαγωγικό Επίπεδο:**
   - Εισαγωγή της εικόνας εισόδου.
2. **Συνελικτικά Επίπεδα:**
   - Πολλά στρώματα συνέλιξης, όπου τα φίλτρα εξάγουν χαρακτηριστικά από την εικόνα.
3. **Συνδέσεις Επιζωοτίας (Residual Blocks):**
   - Αυτά τα βασικά τμήματα περιέχουν μια ή περισσότερες συνελικτικές στρώσεις.
   - Η είσοδος του residual block συνδέεται με την έξοδο του με συνδέσεις πρόσθεσης.
   - Η συνδεδεμένη έξοδος από το residual block περνά από μια συνάρτηση ενεργοποίησης, όπως η ReLU.
4. **Συνελικτικά Επίπεδα (μετά τα Residual Blocks):**
   - Συνεχίζονται τα συνελικτικά επίπεδα για εξαγωγή πιο υψηλής επιπέδου αναπαραστάσεων.
5. **Υποδειγματοληψία (Pooling Layer):**
   - Μείωση της διάστασης των χαρακτηριστικών για απλοποίηση και αντιμετώπιση του υπερ-εκπαίδευσης.
6. **Πλήρως Συνδεδεμένα Επίπεδα (αν χρειάζονται):**
   - Τελικά επίπεδα για την ταξινόμηση ή την επίλυση της συγκεκριμένης εργασίας.

Το ResNet άνοιξε τον δρόμο για την ανάπτυξη βαθύτερων και πιο αποδοτικών νευρωνικών δικτύων, αποτελώντας ένα σημαντικό βήμα στην εξέλιξη της τεχνολογίας όρασης υπολογιστών.

6.10 MobileNet

Το MobileNet είναι μια αρχιτεκτονική συνελικτικού νευρωνικού δικτύου που σχεδιάστηκε για την ανάπτυξη αποδοτικών μοντέλων για υπολογιστικά περιορισμένες συσκευές, όπως κινητά τηλέφωνα και άλλες φορητές συσκευές από τον Andrew Howard [29] το 2017. Η κύρια έμφαση του MobileNet είναι η ελαχιστοποίηση του αριθμού των υπολογισμών και των παραμέτρων, χωρίς όμως να θυσιάζεται η απόδοση του μοντέλου στις εργασίες όρασης.

Κάποια χαρακτηριστικά του MobileNet:

1. **Depthwise Separable Convolution:** Το MobileNet χρησιμοποιεί την τεχνική του "Depthwise Separable Convolution", η οποία διαχωρίζει τη συνέλιξη σε δύο μέρη: μία συνέλιξη μόνο στον χώρο (depthwise convolution) και μία συνέλιξη 1x1 (pointwise convolution) για την σύνθεση των χαρακτηριστικών. Αυτό μειώνει σημαντικά τον αριθμό των υπολογισμών και των παραμέτρων.
2. **Width Multiplier:** Το MobileNet επιτρέπει τον ρύθμιση του "width multiplier", το οποίο κλιμακώνει τον αριθμό των καναλιών στις συνελικτικές στρώσεις. Αυτό επηρεάζει το μέγεθος του μοντέλου και την απόδοσή του.
3. **Resolution Multiplier:** Το "resolution multiplier" επιτρέπει την αλλαγή της ανάλυσης της εικόνας εισόδου. Αυτό μπορεί να επηρεάσει τόσο την απόδοση όσο και την πολυπλοκότητα του μοντέλου.
4. **Efficiency:** Λόγω των παραπάνω τεχνικών, το MobileNet είναι ιδιαίτερα αποδοτικό και κατάλληλο για χρήση σε συσκευές με περιορισμένους υπολογιστικούς πόρους.

Το MobileNet έχει πολλές παραλλαγές όπως το MobileNetV1, το MobileNetV2 και το MobileNetV3, καθένα με διαφορετικές βελτιώσεις και χαρακτηριστικά. Είναι κατάλληλο για εργασίες όπως αναγνώριση αντικειμένων, ανίχνευση προσώπων, ανίχνευση ανωμαλιών, σεναρία και πολλές άλλες εργασίες όρασης υπολογιστών σε περιορισμένες συσκευές.

7. Εκπαίδευση και δημιουργία μοντέλου αυτόνομης οδήγησης

Στο κεφάλαιο που ακολουθεί περιγράφεται η διαδικασία δημιουργίας του μοντέλου αυτόνομης οδήγησης. Αρχικά ορίζονται κάποια βασικά χαρακτηριστικά των εικόνων και πως αυτές αναλύονται από έναν υπολογιστή. Στη συνέχεια δίνεται ο ορισμός της κατηγοριοποίησης εικόνων και η διαδικασία επίλυσης τέτοιων προβλημάτων με τη χρήση της βαθιάς μάθησης. Έπειτα περιγράφεται ο τρόπος συλλογής των δεδομένων για την εκπαίδευση του μοντέλου καθώς και η ίδια η εκπαίδευσή του.

7.1 Εικόνες και υπολογιστής

Οι εικόνες αποτελούν πολύ σημαντική πηγή πληροφορίας και είναι αναπόσπαστο κομμάτι της υπολογιστικής όρασης. Οι ψηφιακές εικόνες που βρίσκονται στους ηλεκτρονικούς υπολογιστές αποτελούνται από εικονοστοιχεία ή Pixels και χωρίζονται σε 2 κατηγορίες:

1. Τη δυαδική εικόνα, η οποία έχει μόνο ένα χρώμα σε αποχρώσεις του γκρι (gray scale image)
2. Την έγχρωμη εικόνα

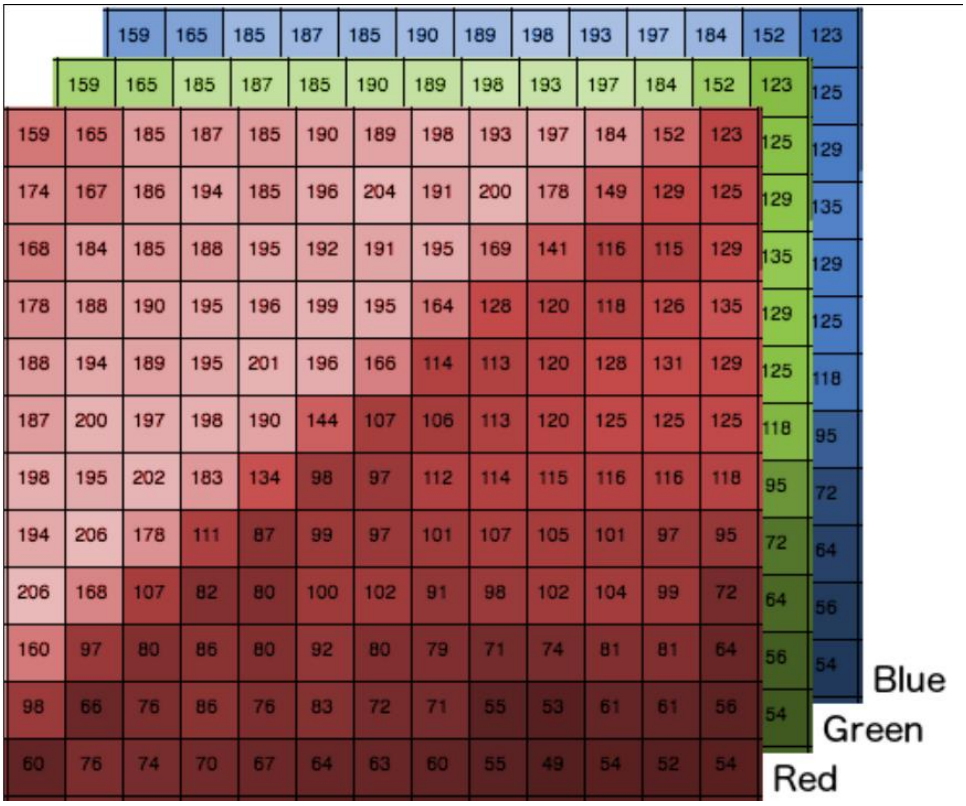
Η δυαδική εικόνα με διαστάσεις H x W αναπαρίσταται σαν ένας πίνακας δύο διαστάσεων H x W από ακέραιους αριθμούς. Κάθε τιμή των κελιών του πίνακα δηλώνει τη φωτεινότητα του pixel και συνήθως βρίσκεται στο διάστημα 0 έως 255 (256 –1) όπως φαίνεται στην εικόνα που ακολουθεί.

159	165	185	187	185	190	189	198	193	197	184	152	123
174	167	186	194	185	196	204	191	200	178	149	129	125
168	184	185	188	195	192	191	195	169	141	116	115	129
178	188	190	195	196	199	195	164	128	120	118	126	135
188	194	189	195	201	196	166	114	113	120	128	131	129
187	200	197	198	190	144	107	106	113	120	125	125	125
198	195	202	183	134	98	97	112	114	115	116	116	118
194	206	178	111	87	99	97	101	107	105	101	97	95
206	168	107	82	80	100	102	91	98	102	104	99	72
160	97	80	86	80	92	80	79	71	74	81	81	64
98	66	76	86	76	83	72	71	55	53	61	61	56
60	76	74	70	67	64	63	60	55	49	54	52	54

Εικόνα 31: Ασπρόμαυρη εικόνα σε pixels

Η έγχρωμη ψηφιακή εικόνα H x W αποτελείται από τρεις δυαδικές εικόνες. Επομένως το κάθε pixel περιέχει τρεις τιμές, όπου κάθε τιμή αντιστοιχεί στην απόχρωση του αντίστοιχου pixel κάθε δυαδικής εικόνας. Το πιο διαδεδομένο χρωματικό σύστημα για τις τρεις συνιστώσες, το οποίο χρησιμοποιείται και στην παρούσα εργασία, είναι το RGB, δηλαδή το κάθε χρώμα αποτελεί συνδυασμό των χρωμάτων κόκκινο,

πράσινο και μπλε. Άρα η κάθε έγχρωμη εικόνα αναπαρίσταται από έναν τρισδιάστατο πίνακα ακεραίων με διαστάσεις H x W x 3.



Εικόνα 32: Πολύχρωμη εικόνα σε pixels

7.2 Κατηγοριοποίηση εικόνων

Ως κατηγοριοποίηση εικόνων ορίζεται η εργασία αντιστοίχισης μιας ετικέτας σε μια εικόνα από ένα προκαθορισμένο σύνολο κατηγοριών. Πρακτικά αυτό σημαίνει ότι μία εικόνα έρχεται σαν είσοδος σε ένα σύστημα, αναλύεται και δίνεται ως έξοδος μια ετικέτα για την εικόνα σε μορφή μιας προκαθορισμένης ομάδας από πιθανές κατηγορίες. Επομένως αν δοθεί ως είσοδος μια εικόνα με διαστάσεις H x W pixels σε RGB χρωματικό σύστημα τότε σκοπός του συστήματος είναι να πάρει την H x W x 3 pixels εικόνα και να ταξινομήσει σωστά τα περιεχόμενά της.

Για τη δημιουργία ενός ταξινομητή εικόνων με τη χρήση της βαθιάς μάθησης και των νευρωνικών δικτύων ακολουθείται μια συγκεκριμένη διαδικασία η οποία περιγράφεται παρακάτω.

1. Συλλογή των δεδομένων: Το πρώτο βήμα για να δημιουργήσουμε ένα μοντέλο ταξινόμησης είναι να συγκεντρώσουμε το αρχικό σύνολο δεδομένων. Επομένως πρέπει να συγκεντρώσουμε ένα σύνολο εικόνων καθώς και τις ετικέτες που σχετίζονται με κάθε εικόνα. Αυτές οι ετικέτες πρέπει να προέρχονται από ένα πεπερασμένο σύνολο κατηγοριών, όπως σκύλος, γάτα, αλεπού. Ο αριθμός των εικόνων για κάθε κατηγορία πρέπει να είναι περίπου ίδιος αλλιώς ο ταξινομητής μας θα γίνει προκατειλημμένος προς την κλάση με τον μεγαλύτερο αριθμό. Η ανισορροπία της κλάσης είναι ένα πρόβλημα στη μηχανική μάθηση και υπάρχουν διάφοροι τρόποι να ξεπεραστεί.
2. Διαχωρισμός του συνόλου των δεδομένων: μετά τη συλλογή των δεδομένων ακολουθεί ο διαχωρισμός τους. Το σύνολο των δεδομένων πρέπει να χωριστεί σε δύο μέρη:
    i. Ένα σετ εκπαίδευσης: χρησιμοποιείται για να “μάθει” ο ταξινομητής πως μοιάζει κάθε κατηγορία κάνοντας προβλέψεις στα δεδομένα εισόδου και στη συνέχεια διορθώνεται όταν οι προβλέψεις είναι λανθασμένες.

ii. Ένα σετ δοκιμών: χρησιμοποιείται για την αξιολόγηση της απόδοσης ενός ταξινομητή μετά την εκπαίδευση.

Τα δύο αυτά σετ δεδομένων είναι σημαντικό να είναι ανεξάρτητα και να μην επικαλύπτονται

3. Εκπαίδευση του δικτύου: έχοντας το σετ δεδομένων εκπαίδευσης μπορεί να ξεκινήσει η εκπαίδευση του δικτύου. Στόχος είναι το δίκτυο να μάθει να αναγνωρίζει καθεμία από τις κατηγορίες στα δεδομένα μας με ετικέτα. Όταν το μοντέλο κάνει λάθος, μαθαίνει από αυτό το λάθος και βελτιώνεται.
4. Αξιολόγηση του δικτύου: Το τελευταίο βήμα της διαδικασίας είναι η αξιολόγηση του δικτύου. Για κάθε εικόνα από το σετ των δοκιμών δίνεται σαν είσοδος στο εκπαιδευμένο δίκτυο και εκείνο επιστρέφει σαν έξοδο την πρόβλεψή για την ετικέτα της εικόνας. Τα αποτελέσματα συγκρίνονται με τις ετικέτες που είχε αρχικά το σετ δοκιμών και υπολογίζεται ο αριθμός των σωστών προβλέψεων του ταξινομητή.

7.3 Συλλογή δεδομένων μοντέλου αυτόνομης οδήγησης

Η συλλογή των δεδομένων για τη δημιουργία του μοντέλου αυτόνομης οδήγησης χωρίζεται σε δύο κατηγορίες:

- Συλλογή δεδομένων για την κίνηση του οχήματος
- Συλλογή εικόνων όπου το όχημα σταματάει

7.3.1 Συλλογή δεδομένων για την κίνηση του οχήματος

Η συλλογή δεδομένων για την κίνηση του οχήματος πραγματοποιείται με εκτέλεση της κλάσης record_dataset.py, όπως αναφέρθηκε και στο κεφάλαιο 4.

Αρχικά κατά την εκτέλεση της record_dataset.py ρυθμίζεται η λειτουργία από τα pins του Jetson και στη συνέχεια δημιουργούνται οι κλάσεις MqttCar, DirectoryManager, KeyboardReader, SingleLed, Car, DistanceSensor και CarCamera οι οποίες περιγράφονται στα κεφάλαια 4 και 5. Επίσης αρχικοποιείται η τιμή της μεταβλητής του φακέλου όπου θα αποθηκευτούν τα αρχεία και η μεταβλητή για το ξεκίνημα της καταγραφής.

```
# Set Jetson GPIO Mode
GPIO.setmode(GPIO.BCM)
# Create mqtt car class
mqtt = MqttCar()
# Create directory manager class
dir_manager = DirectoryManager()
# Crete keyboard reader class
keyboard_reader = KeyboardReader()
# Create LED class
led = SingleLED(GPIO)
# Create Car class
car = Car(GPIO, 45)
# Create Distance sensor class
distance_sensor = DistanceSensor(GPIO)
# Create car camera class
car_camera = CarCamera(autopilot=False, record_stops=False, video_output=True)

# Set initial values
created_dir = ''
start_rec = ''
```

Εικόνα 33: Κώδικας εκκίνησης συλλογής δεδομένων

Στη συνέχεια ξεκινάει η σύνδεση με τον MQTT broker, ανάβουν οι led λυχνίες, δημοσιεύονται ορισμένα στοιχεία του οχήματος στα αντίστοιχα ορισμένα θέματα(topics) του MQTT και ενεργοποιείται και η κάμερα.

```
# Start mqtt
mqtt.start()
# Led startup
led.onStartCarLEDs()

# Publish init values to relevant topics
mqtt.publish(car.speed, const.car_speed_topic)
mqtt.publish(car.move, const.car_move_topic)
mqtt.publish(const.record_dataset_mode, const.car_mode_topic)
# Start camera parallel thread
car_camera.start()
```

Εικόνα 34: Κώδικας εκκίνησης συλλογής δεδομένων

Έπειτα ακολουθεί μια επαναληπτική δομή while η οποία διαβάζει από το πληκτρολόγιο το κουμπί που πάτησε ο χρήστης και πραγματοποιεί τις παρακάτω ενέργειες μέσα από μία δομή if – else.

- Βελάκι πάνω: το όχημα κινείται μπροστά
- Βελάκι κάτω: το όχημα κινείται πίσω
- Βελάκι δεξιά: το όχημα κινείται δεξιά
- Βελάκι αριστερά: το όχημα κινείται αριστερά
- Spacebar: το όχημα σταματάει
- Συν(+): αύξηση ταχύτητας
- Πλην(-): μείωση ταχύτητας
- l: ορισμός χαμηλής ταχύτητας
- m: ορισμός μέσης ταχύτητας
- h: ορισμός υψηλής ταχύτητας
- o: δημιουργία φακέλου, έναρξη εγγραφής δεδομένων & άναμα κόκκινου led
- p: τέλος εγγραφής και σβήσιμο κόκκινου led
- q: κλείσιμο κάμερας

```
while True:
    # Read keyboard input
    key_press = keyboard_reader.readKey()

    if key_press == 0:
        car.forward()
        mqtt.publish(car.move, const.car_move_topic)
    elif key_press == 1:
        car.backward()
        mqtt.publish(car.move, const.car_move_topic)
    elif key_press == 2:
        if car.direction == const.FWD:
            car.forwardRight()
            mqtt.publish(car.move, const.car_move_topic)
        elif car.direction == const.BWD:
            car.backwardRight()
            mqtt.publish(car.move, const.car_move_topic)
    elif key_press == 3:
        if car.direction == const.FWD:
            car.forwardLeft()
            mqtt.publish(car.move, const.car_move_topic)
        elif car.direction == const.BWD:
            car.backwardLeft()
            mqtt.publish(car.move, const.car_move_topic)
    elif key_press == ' ':
        car.stop()
        mqtt.publish(car.move, const.car_move_topic)
    elif key_press == '+':
        speed = car.increase_speed()
        mqtt.publish(speed, const.car_speed_topic)
    elif key_press == '-':
        speed = car.decrease_speed()
        mqtt.publish(speed, const.car_speed_topic)
    elif key_press == 'l':
        car.low_speed()
        mqtt.publish(car.speed, const.car_speed_topic)
    elif key_press == 'm':
        car.medium_speed()
        mqtt.publish(car.speed, const.car_speed_topic)
    elif key_press == 'h':
        car.high_speed()
        mqtt.publish(car.speed, const.car_speed_topic)
    elif key_press == 'o':
        created_dir, start_rec = dir_manager.create_directory()
        car_camera.start_recording(created_dir=created_dir, start_rec=start_rec)
        # Turn On Red LED
        led.turnOn(const.red_LED_pin)

    elif key_press == 'p':
        # Stop camera record
        car_camera.stop_recording()
        # Turn Off Red LED
        led.turnOff(const.red_LED_pin)

    elif key_press == 'q':
        car_camera.stop()
        car_camera.join()
        car_camera = None
    elif ord(key_press) == 3:
        break
    if car_camera is not None:
        car_camera.pass_csv_param(car_move=car.move,
                                  car_speed=car.speed,
                                  distance=distance_sensor.distance)
```

Εικόνα 35: Ενέργειες μέσω πληκτρολογίου για συλλογή δεδομένων

Τέλος ο τερματισμός της εκτέλεσης του προγράμματος πραγματοποιείται πατώντας τα πλήκτρα Ctrl+C και με αυτό τον τρόπο “καθαρίζουν” τα pins του Jetson, κλείνει η κάμερα και αποσυνδέεται το MQTT.

Για τη συλλογή των δεδομένων είναι απαραίτητη η δημιουργία πίστας η οποία θα μοιάζει όσο περισσότερο γίνεται σε κανονικό ασφαλτοστρωμένο οδόστρωμα. Η κάθε πίστα αποτελείται από μαύρα κομμάτια κανσόν χαρτιού διαστάσεων 50x70 cm κολλημένα μεταξύ τους. Η διαγράμμιση του δρόμου δημιουργήθηκε από κομμάτια λευκής χαρτοταινίας με σκοπό το όχημα να κινείται εντός των ορίων. Τα

κομμάτια του δρόμου ενώθηκαν σε πολλούς διαφορετικούς συνδυασμούς για τη δημιουργία πιστών ώστε να συγκεντρώσουμε όσο περισσότερα ανόμοια δεδομένα γίνεται.

Μετά την ολοκλήρωση κάθε διαδικασίας καταγραφής δεδομένων δημιουργείται στο σύστημα αρχείων του Jetson ένας φάκελος με το όνομα της πίστας όπου έγινε η καταγραφή (π.χ Track_1) ο οποίος περιέχει υποφακέλους με ονομασία την ημερομηνία και την ώρα που ξεκίνησε η καταγραφή. Κάθε υποφάκελος περιέχει:

1. Αρχείο CSV όπου κάθε εγγραφή αντιστοιχεί σε μία εικόνα και έχει την ημερομηνία και ώρα που τραβήχτηκε η εικόνα, την τιμή της απόστασης από τον αισθητήρα του οχήματος, την κίνηση του οχήματος δηλαδή την ετικέτα της εικόνας και την ταχύτητα
2. Όλες τις εικόνες που τραβήχτηκαν κατά τη διάρκεια εγγραφής των δεδομένων, όπως αναφέρεται και στο κεφάλαιο 4

7.3.2 Συλλογή δεδομένων για το σταμάτημα του οχήματος

Η συλλογή δεδομένων για το σταμάτημα του οχήματος πραγματοποιείται με την εκτέλεση της κλάσης record_stop_images.py. Η εκτέλεση της συγκεκριμένης κλάσης είναι σχεδόν ίδια με την recored_dataset.py. Η μόνη διαφορά βρίσκεται μέσα στη δομή while όπου η εφαρμογή διαβάζοντας την είσοδο που δίνει ο χρήστης από το πληκτρολόγιο κάνει τις παρακάτω ενέργειες:

- Spacebar: το σύστημα αποθηκεύει μια εικόνα και την αντιστοιχεί με μία ετικέτα ‘stop’.
- p: Κλείσιμο κάμερας

```
while True:
    key_pressed = keyboard_reader.readKey()

    if key_pressed == ' ':
        # Turn On green LED
        led.turnOn(const.green_LED_pin)
        time.sleep(0.5)
        led.turnOff(const.green_LED_pin)
        # Take photo
        car_camera.take_pic(created_dir=created_dir, start_rec=start_rec)
    elif key_pressed == 'p':
        car_camera.stop_photo_camera()
        car_camera.join()
        car_camera = None
    elif ord(key_pressed) == 3:
        break
    if car_camera is not None:
        car_camera.pass_csv_param(car_move=car.move,
                                  car_speed=car.speed,
                                  distance=distance_sensor.distance)
```

Εικόνα 36: Ενέργειες μέσω πληκτρολογίου για συλλογή δεδομένων για το σταμάτημα

Επομένως το σύστημα αποθηκεύει μία τις εικόνες μετά από εντολή του χρήστη. Για τη συλλογή των δεδομένων τοποθετήθηκαν διάφορα εμπόδια, στα οποία το όχημα πρέπει να σταματάει, σε διαφορετικές αποστάσεις και γωνίες κάθε φορά.

Μετά την ολοκλήρωση κάθε διαδικασίας καταγραφής δεδομένων σταματήματος δημιουργείται η ίδια δομή με φακέλους και αρχεία όπως με την καταγραφή δεδομένων κίνησης. Όλες οι εικόνες έχουν αντιστοίχιση σε ετικέτα σταματήματος ‘stop’.

7.4 Εκπαίδευση μοντέλου αυτόνομης οδήγησης

Η εκπαίδευση του μοντέλου πραγματοποιήθηκε δημιουργώντας ένα αρχείο .ipynb από το IPython Notebook γνωστό ως Jupyter Notebook. Το Jupyter Notebook είναι ένα διαδραστικό υπολογιστικό περιβάλλον, στο οποίο συνδυάζεται η εκτέλεση κώδικα με κείμενο, μαθηματικά, γραφικές παραστάσεις και πολυμέσα.

Αρχικά στο notebook της εκπαίδευσης του μοντέλου αυτόνομης οδήγησης εισάγονται οι βιβλιοθήκες που θα χρησιμοποιηθούν κατά την εκτέλεση του αρχείου. Έπειτα ορίζονται οι σταθερές:

- FILES_PATH: η διαδρομή όπου βρίσκονται όλοι οι φάκελοι με τα αρχείο της εκπαίδευσης
- MODEL_FILENAME: το όνομα του αρχείου που θα δημιουργηθεί μετά την εκπαίδευση
- BEST_MODEL_FILENAME: το όνομα του αρχείου που θα δημιουργηθεί με το λιγότερο ποσοστό σφαλμάτων
- LAST_EPOCH_FILENAME: το όνομα του αρχείου που θα δημιουργηθεί μετά την εκτέλεση της τελευταίας επανάληψης της δημιουργίας του μοντέλου
- PARENT_PATHS: Οι φάκελοι με τα δεδομένα που θα χρησιμοποιηθούν για την εκπαίδευση
- class_names: οι πιθανές τιμές (κλάσεις, ετικέτες) που μπορεί να αντιστοιχήσουν σε μία εικόνα

```
# FILES PATH
FILES_PATH='./files/'

# FILENAMES
# FILENAME = 'track9+10+11+12+stop_4'
FILENAME = 'track9+10+11+12+13_3'
MODEL_FILENAME = FILENAME + '_model_state.pt'
BEST_MODEL_FILENAME = FILENAME + '_best_model_state.pt'
LAST_EPOCH_MODEL = FILENAME + '_model_state.pt'

# PARENT PATHS
PARENT_PATHS = [
    FILES_PATH + 'track9',
    FILES_PATH + 'track10',
    FILES_PATH + 'track11',
    FILES_PATH + 'track12_224',
    FILES_PATH + 'track_13',
#     FILES_PATH + 'stop_data_new'
]

# CLASSES
class_names = ['forward', 'stop', 'forward_left', 'forward_right']
```

Εικόνα 37: Αρχικοποίηση παραμέτρων εκπαίδευσης μοντέλου

Έπειτα ορίζονται κάποιες σταθερές για τη διαδικασία του augmentation των εικόνων, της οποίας η περιγραφή ακολουθεί παρακάτω, που σχετίζονται με το ποιες μέθοδοι θα χρησιμοποιούνται.

```
# AUGMENTATION VALUES
RESIZE_IMAGES = True
COLOR_JITTER = True
CROP_IMAGES = False
RANDOM_AFFINE = False
```

Εικόνα 38: Αρχικοποίηση παραμέτρων augmentation

Στη συνέχεια ορίζονται κάποιες κλάσεις που θα βοηθήσου στην εκτέλεση του προγράμματος:

- Directory. Η κλάση directory χρησιμοποιείται για τους φακέλους και επιστρέφει όλους τους υποφακέλους που μπορεί να έχει ένας φάκελος
- ImageLoader. Με τη κλάση ImageLoader φορτώνουμε τις εικόνες από το σύστημα φακέλων και εφαρμόζουμε τεχνικές augmentation για τη διαδικασία της εκπαίδευσης
- ValImageLoader. Η κλάση αυτή χρησιμοποιείται για να φορτώσουμε τις εικόνες που θα χρησιμοποιηθούν στη διαδικασία της αξιολόγησης που παραγόμενου μοντέλου
- CsvLoader. Η κλάση CsvLoader φορτώνει τα αρχεία csv κάθε φακέλου και αντικαθιστά τις τιμές των ετικετών με αριθμούς π.χ ετικέτα forward με 0
- TrainTestDataSplitter. Η συγκεκριμένη κλάση χωρίζει τα δεδομένα, εικόνες και τις αντίστοιχες ετικέτες, σε 2 σετ δεδομένων, το σετ εκπαίδευσης που αποτελεί το 80% του συνόλου και το σετ δοκιμών το υπόλοιπο 20%

Ακολουθεί η έναρξη της εκτέλεσης του προγράμματος η οποία αρχίζει με την κλήση της μεθόδου find_all_images_and_csv_paths. Η μέθοδος αυτή υπολογίζει το πλήθος των εικόνων και των φακέλων που θα χρησιμοποιηθεί για τη δημιουργία του μοντέλου.

Επόμενο βήμα είναι η παρουσίαση δειγματοληπτικά 20 τυχαίων εικόνων από τα δεδομένα ώστε ο χρήστης να επιβεβαιώσει ότι τα δεδομένα είναι έγκυρα. Η παρουσίαση αυτή πραγματοποιείται καλώντας τη μέθοδο show_sample_images_grid(image_paths). Η μέθοδος δέχεται σαν είσοδο έναν πίνακα από διαδρομές των εικόνων και με τη χρήση της κλάσης ImageLoader φορτώνονται και παρουσιάζονται οι εικόνες στις οποίες έχει εφαρμοσθεί data augmentation.

Η διαδικασία του "data augmentation" αφορά τον τρόπο με τον οποίο μετασχηματίζουμε και παραμορφώνουμε τα υπάρχοντα δεδομένα εκπαίδευσης προκειμένου να δημιουργήσουμε νέα παραδείγματα. Στο πλαίσιο της βαθιάς μάθησης, η διαδικασία αυτή είναι σημαντική καθώς μπορεί να βοηθήσει στην αύξηση του όγκου των δεδομένων εκπαίδευσης, να βελτιώσει τη γενίκευση του μοντέλου και να αντιμετωπίσει το πρόβλημα της υπερεκπαίδευσης.

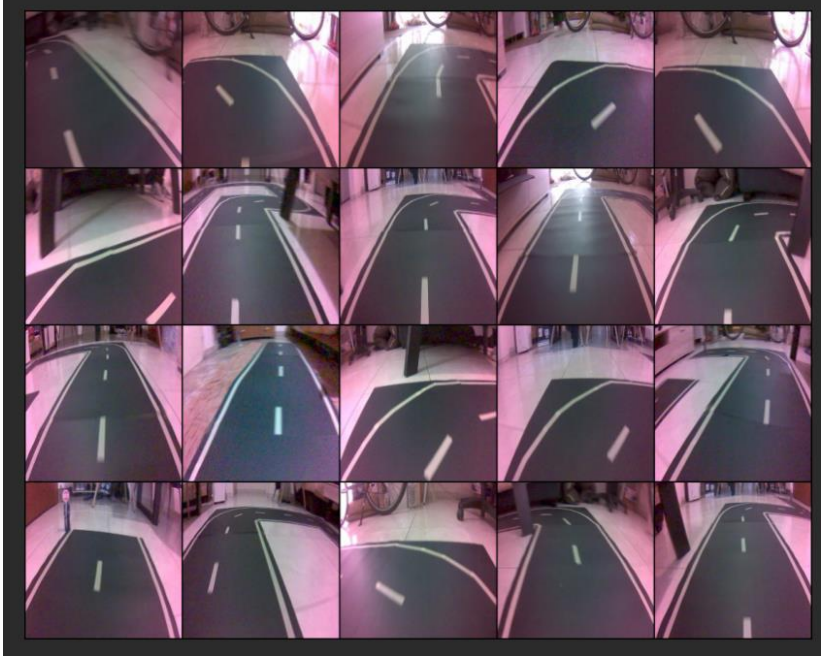
Οι τεχνικές data augmentation μπορεί να περιλαμβάνουν:

1. **Κατολίσθηση (Translation):** Μετατόπιση της εικόνας κατά μικρά ποσά οριζόντια και κατακόρυφα.
2. **Περιστροφή (Rotation):** Περιστροφή της εικόνας γύρω από το κέντρο της.
3. **Κλιμάκωση (Scaling):** Αλλαγή της κλίμακας της εικόνας, είτε αυξάνοντάς την είτε μειώνοντάς την.
4. **Ανακλαστική Αντανάκλαση (Horizontal/Vertical Flip):** Ανακλαστική αντανάκλαση της εικόνας οριζόντια ή κατακόρυφα.
5. **Σκίαση (Shearing):** Αλλαγή του γωνιακού σχήματος της εικόνας.
6. **Προσθήκη Θορύβου (Noise Addition):** Προσθήκη τυχαίου θορύβου στην εικόνα.
7. **Αλλαγή Φωτεινότητας (Brightness Adjustment):** Αύξηση ή μείωση της φωτεινότητας της εικόνας.

8. **Αλλαγή Χρωματικότητας (Color Adjustment):** Αλλαγή των χρωματικών τόνων της εικόνας.

Ο συνδυασμός αυτών των μετασχηματισμών μπορεί να δημιουργήσει μια ποικιλία από νέα παραδείγματα που είναι παρόμοια, αλλά διαφορετικά από τα αρχικά. Αυτό βοηθά στην εκπαίδευση του μοντέλου να αντιμετωπίσει διαφορετικές περιπτώσεις και παραλλαγές των δεδομένων, καθιστώντας το πιο ανθεκτικό και αποδοτικό.

```
sample_images = [np.random.choice(all_image_paths) for i in range(20)]
sample_images

im = ImageLoader()

def show_sample_images_grid(image_paths):
    images = [np.array(im.load_sample_image(img)) for img in image_paths]

    images = torch.as_tensor(images)

    images = images.permute(0, 3, 1, 2)

    grid_img = torchvision.utils.make_grid(images, nrow=5)

    plt.figure(figsize=(24, 12))

    plt.imshow(grid_img.permute(1, 2, 0))

    plt.axis('off');
show_sample_images_grid(sample_images)
```



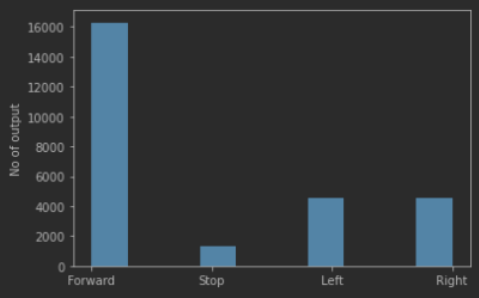
Εικόνα 39: Εμφάνιση δείγματος εικόνων

Για την καλύτερη ενημέρωση του χρήστη σχετικά με τα δεδομένα, είναι απαραίτητη η παρουσίαση του πλήθους των εικόνων ανά κατηγορία. Για τη δημιουργία του διαγράμματος φορτώθηκαν τα αρχεία CSV που αντιστοιχούν στις εικόνες μέσω της κλάσης CsvLoader και με τη χρήση της βιβλιοθήκης matplotlib.pyplot προκύπτει το αποτέλεσμα.

```
cl= CsvLoader()

output_classes = np.zeros(0)
for path in all_csv_paths:
    csv = cl.load_csv_to_df(path)
    data_classes = cl.map_class_to_int(csv)['Move'] # .astype('int64')
    output_classes = np.concatenate((output_classes,data_classes), axis=0)

print(len(output_classes))
```

```
26742
```

```
_ = plt.hist(output_classes)
_ = plt.ylabel('No of output')
positions = (0, 1, 2, 3)
labels = ("Forward", "Stop", "Left", "Right")
plt.xticks(positions, labels)
plt.show()

array = np.unique(output_classes, return_counts=True)[1]
# array = [array[0],0, array[1], array[2]]
print(array)
print(f'Forward count: {array[0]}')
print(f'Stop count: {array[1]}')
print(f'Forward Left count: {array[2]}')
print(f'Forward Right count: {array[3]}')
```




```
[16292  1350  4550  4550]
Forward count: 16292
Stop count: 1350
Forward Left count: 4550
Forward Right count: 4550
```

Εικόνα 40: Φόρτωση πλήθους ανά κατηγορία

Από το διάγραμμα συνεπάγεται πως οι εικόνες που αντιστοιχούν σε ετικέτα ‘Forward’ είναι πού περισσότερες από τις υπόλοιπες και γενικά δεν υπάρχει ισορροπία μεταξύ του πλήθους των κλάσεων, πράγμα που πρέπει να συμπεριληφθεί στην εκπαίδευση του μοντέλου μέσω της εφαρμογής βαρών . Για τη εισαγωγή των βαρών στη δημιουργία του μοντέλου γίνεται χρήση της κλάσης WeightedRandomSampler από τη βιβλιοθήκη pytorch, η οποία για τη δημιουργία της χρησιμοποιεί έναν πίνακα που περιέχει το βάρος κάθε κατηγορίας.

```
class_sample_count = np.array([len(np.where(train_classes==t)[0]) for t in np.unique(train_classes)])
weight = 1. / class_sample_count
train_classes_int = [int(a) for a in train_classes]


print(weight)
print(len(train_classes_int))

samples_weight = np.array([weight[t] for t in train_classes_int])
print(len(samples_weight))

samples_weight = torch.from_numpy(samples_weight)
print(len(samples_weight))
sampler =  torch.utils.data.sampler.WeightedRandomSampler(
    weights= samples_weight.type('torch.DoubleTensor'), num_samples = len(samples_weight),replacement=True)
```

Εικόνα 41: Ορισμός βαρών ανά κατηγορία

Κατά τη συνέχεια εκτέλεσης του προγράμματος πραγματοποιείται διαχωρισμός των δεδομένων σε 2 σετ, ένα σετ εκπαίδευσης και ένα σετ δοκιμών με χρήση της κλάσης TrainTestDataSplitter, η οποία αναφέρθηκε και παραπάνω. Μετά την ολοκλήρωση της διαδικασίας δημιουργούνται 2 πίνακες όπου η κάθε εγγραφή περιέχει μία εικόνα και την κατηγορία που της αντιστοιχεί.

Για την εισαγωγή των δεδομένων στην εκπαίδευση του μοντέλου είναι απαραίτητη η χρήση της κλάσης DataLoader, που ανήκει στη βιβλιοθήκη pytorch και δέχεται σαν παραμέτρους τον πίνακα των δεδομένων που δημιουργήθηκε στο προηγούμενο βήμα, το πλήθος κάθε ομάδας εικόνων που θα φορτώνεται κατά την εκπαίδευση και μία μεταβλητή WeightedRandomSampler που δημιουργήθηκε από τα βάρη των δεδομένων.

```
train_loader = torch.utils.data.DataLoader(train_data, batch_size=128, shuffle=False,
                                           sampler=sampler, num_workers=6)
if SPLIT_DATA:
    test_loader =  torch.utils.data.DataLoader(test_data, batch_size=128, shuffle=True, num_workers=6)
len(train_data)
```

Εικόνα 42: Φόρτωση δεδομένων εκπαίδευσης

Για την εκπαίδευση και τη δημιουργία του μοντέλου αυτόνομης οδήγησης επιλέχθηκε το προ-εκπαιδευμένο μοντέλο ResNet18 για 4 κλάσεις με συνάρτηση διασταυρούμενης εντροπίας ως συνάρτηση κόστους και προσαρμοστική εκτίμηση ροπών (ADAM) ως μέθοδος βελτιστοποίησης.

```
# Create Model
model = models.resnet18(pretrained=True)
model.fc = torch.nn.Linear(512, len(class_names))
device = torch.device('cuda')
model = model.to(device)

# Set training params

# optimizer = optim.SGD(model.parameters(), lr=1e-2)
optimizer = optim.Adam(model.parameters(), lr=1e-4)
loss_fn = nn.CrossEntropyLoss()
```

Εικόνα 43: Δημιουργία μοντέλου

Για την αξιολόγηση των αποτελεσμάτων του μοντέλου που θα δημιουργηθεί είναι απαραίτητη η δημιουργία ενός συνόλου από δεδομένα όμοια με αυτά που θα χρησιμοποιηθούν για την εκπαίδευση χωρίς όμως την προσθήκη της μεθόδου “data augmentation”. Επομένως, από τα αντίστοιχους φακέλους φορτώνονται οι εικόνες και τα αρχεία csv, όπου υπάρχει η αντιστοίχιση εικόνας-κλάση και δημιουργείται το τελικό σετ δεδομένων αξιολόγησης (val_loader2).

```
val_image_paths= []
val_csv_paths = []


val_parent_directory = Directory(FILES_PATH +'val_set')

val_subdirectories = val_parent_directory.getSubdirectories()


for subdirectory in sorted(val_subdirectories):
    subdirectory_path_val = FILES_PATH +'val_set' + '/' + subdirectory
    imageLoader = ImageLoader(subdirectory_path_val)
    image_paths = imageLoader.load_image_paths()
    image_paths.sort(key=lambda f: int(re.sub('\D', '', f)))
    val_image_paths += image_paths

    csvLoader = CsvLoader(subdirectory_path_val)
    csvPath = csvLoader.load_csv_path()
    val_csv_paths += csvPath
val_csv_paths
```

Εικόνα 44: Φόρτωση εικόνων δεδομένων αξιολόγησης

```
val_output_classes = np.zeros(0)
for path in val_csv_paths:
    csv = cl.load_csv_to_df(path)
    data_classes = cl.map_class_to_int(csv)['Move'] # .astype('int64')
    val_output_classes = np.concatenate((val_output_classes,data_classes), axis=0)
```

Εικόνα 45: Φόρτωση κατηγοριών δεδομένων αξιολόγησης

```
im2=ValImageLoader()
val_images2 =  [torch.tensor(im2.load_image(path)) for path in val_image_paths]
```

```
val_data2 = [(val_images2[i], val_output_classes[i]) for i in range(len(val_images2))]
len(val_data2)
```

```
2056
```

```
val_loader2 =  torch.utils.data.DataLoader(val_data2, batch_size=128, shuffle=False, num_workers=6)
```

Εικόνα 46: Φόρτωση δεδομένων αξιολόγησης

Η βασική διαδικασία εκπαίδευσης των νευρωνικών δικτύων είναι ο εκπαιδευτικός βρόχος (training loop). Είναι ένας επαναλαμβανόμενος κύκλος βημάτων που συνδέονται μεταξύ τους και

εκτελούνται πολλές φορές μέχρις ότου το μοντέλο συγκλίνει και εκπαιδευτεί στα δεδομένα. Ο εκπαιδευτικός βρόχος υλοποιείται με τη μέθοδο training_loop η οποία δέχεται σαν παραμέτρους τον αριθμό των επαναλήψεων, τη μέθοδο βελτιστοποίησης, το προ-εκπαιδευμένο μοντέλο, τη συνάρτηση κόστους και τα δεδομένα εκπαίδευσης. Κατά την εκτέλεση αυτής της μεθόδου πραγματοποιείται μια for loop δομή επανάληψης. Σε κάθε επανάληψη καλείται μία μέθοδος για την εκπαίδευση του μοντέλου και μία για την αξιολόγησή του. Οι δύο αυτές μέθοδοι περιγράφονται παρακάτω. Έπειτα τυπώνονται τα αποτελέσματα της αξιολόγησης της κάθε επανάληψης.

```
def training_loop(n_epochs, optimizer, model, loss_fn, train_loader):
    history = defaultdict(list)
    best_accuracy = 0

    for epoch in range(1, n_epochs + 1):
        print(f'Epoch {epoch}/{n_epochs}')
        print('-' * 10)
        train_acc, train_loss = train_epoch(
            optimizer,
            model,
            loss_fn = loss_fn,
            train_loader = train_loader)

        print(f'Train loss {train_loss} accuracy {train_acc}')
        val_acc, val_loss = evaluate_epoch(
            model,
            data_loader=val_loader2,
            loss_fn=loss_fn)
        print(f'Val loss {val_loss} accuracy {val_acc}')
        print()
        print('{} Epoch {} END'.format(datetime.datetime.now(), epoch))
        print('-' * 20)
        history['train_acc'].append(train_acc)
        history['train_loss'].append(train_loss)
        history['val_acc'].append(val_acc)
        history['val_loss'].append(val_loss)
        if val_acc > best_accuracy:
            torch.save(model.state_dict(), FILES_PATH + BEST_MODEL_FILENAME)
            best_accuracy = val_acc

    print(f'Best val accuracy: {best_accuracy}')

    return model, history
```

Εικόνα 47: Εκπαιδευτικός βρόγχος

Με την κλήση της μεθόδου train_epoch πραγματοποιούνται οι ακόλουθες ενέργειες για κάθε επανάληψη:

1. Φορτώνονται τα δεδομένα σε παρτίδες (batches) και παρουσιάζονται στο μοντέλο για εκπαίδευση. Το μέγεθος της κάθε παρτίδας ορίζεται ως παράμετρος όταν φορτώνονται τα δεδομένα
2. Τα δεδομένα προωθούνται μέσα από το μοντέλο με σκοπό την παραγωγή προβλέψεων
3. Πραγματοποιείται ο υπολογισμός του σφάλματος μεταξύ των προβλέψεων και των πραγματικών τιμών
4. Υπολογίζονται οι παράγωγοι του σφάλματος ως προς τα βάρη του μοντέλου, χρησιμοποιώντας τον αλγόριθμο οπισθοδρόμησης
5. Ενημερώνονται τα βάρη του μοντέλου με τις νέες τιμές κάνοντας χρήση του αλγόριθμου βελτιστοποίησης που έχει επιλεγεί. Στη συγκεκριμένη περίπτωση έχει επιλεγεί ο αλγόριθμος Adam
6. Στο τέλος η μέθοδος επιστρέφει το ποσοστό των σωστών προβλέψεων ως προς τις συνολικές καθώς και το μέσο σφάλμα

```python
def train_epoch(optimizer, model, loss_fn, train_loader):
    """Method which contains the training logic"""
    model = model.train()

    losses = []
    correct_predictions = 0
    total = 0

    for imgs, labels in train_loader:
        imgs = imgs.to(device=device)
        labels = labels.to(device=device, dtype=torch.int64)
        outputs = model(imgs)

        _, preds = torch.max(outputs, dim=1)

        loss = loss_fn(outputs, labels)

        correct_predictions += torch.sum(preds == labels)
        total += labels.shape[0]
        losses.append(loss.item())

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    return  correct_predictions.double() / total, np.mean(losses)
```

Εικόνα 48: Ενέργειες κάθε επανάληψης

Με την κλήση της μεθόδου evaluate_epoch πραγματοποιείται η αξιολόγηση του μοντέλου για κάθε επανάληψη του βρόχου. Πιο συγκεκριμένα:

1. Φορτώνονται δεδομένα σε παρτίδες από ένα σετ δεδομένων που ονομάζεται σύνολο επικύρωσης και είναι διαφορετικό από εκείνο που χρησιμοποιείται για την εκπαίδευση
2. Τα δεδομένα προωθούνται μέσα από το μοντέλο με σκοπό την παραγωγή προβλέψεων
3. Πραγματοποιείται ο υπολογισμός του σφάλματος μεταξύ των προβλέψεων και των πραγματικών τιμών

4. Στο τέλος η μέθοδος επιστρέφει το ποσοστό των σωστών προβλέψεων ως προς τις συνολικές καθώς και το μέσο σφάλμα

```
val_loader2 =  torch.utils.data.DataLoader(val_data2, batch_size=128, shuffle=False, num_workers=6)

def evaluate_epoch(model, data_loader, loss_fn):
    """Method which contains the evaluation logic"""
    model = model.eval()

    losses = []
    correct_predictions = 0
    total = 0

    with torch.no_grad():
        for imgs, labels in data_loader:
            imgs = imgs.to(device=device)
            labels = labels.to(device=device, dtype=torch.int64)

            outputs = model(imgs)
            _, predicted = torch.max(outputs, dim=1)

            loss = loss_fn(outputs, labels)
            correct_predictions += int((predicted == labels).sum())

            total += labels.shape[0]
            losses.append(loss.item())

    return correct_predictions / total, np.mean(losses)
```

Εικόνα 49: Αξιολόγηση κάθε επανάληψης

Η διαδικασία εκπαίδευσης και αξιολόγησης είναι παρόμοια μέχρι ένα σημείο. Η διαφορά είναι ότι κατά την αξιολόγηση δεν ενημερώνεται το μοντέλο. Μετά το πέρας της εκπαίδευσης το μοντέλο που αποθηκεύεται είναι εκείνο με την καλύτερη ακρίβεια.

Για την παραγωγή του μοντέλου πραγματοποιήθηκαν δοκιμές με διάφορους αριθμούς επαναλήψεων (5, 10, 15, 20, 30) αλλά επιλέχθηκε ο αριθμός 5 καθώς παραπάνω το μοντέλο έμοιαζε να κάνει overfitting. Το overfitting είναι ένα πρόβλημα στην εκπαίδευση νευρωνικών δικτύων. Συμβαίνει όταν το μοντέλο «μαθαίνει» με μεγάλη ακρίβεια τα δεδομένα εκπαίδευσης και τα προβλέπει με μεγάλη επιτυχία, όμως αποτυγχάνει να γενικεύσει σε νέα αόριστα δεδομένα και οδηγείται σε χαμηλή απόδοση σε αυτά τα δεδομένα.

```
base_model, history = training_loop(
    n_epochs = 5,
    optimizer = optimizer,
    model = model,
    loss_fn = loss_fn,
    train_loader = train_loader,
)
```

```
----------
Train loss 0.005711156792735957 accuracy 0.9982424650362725
Val loss 0.41355731666964646 accuracy 0.9134241245136187

2021-01-30 09:14:08.682694 Epoch 4 END
--------------------
Epoch 5/5
----------
Train loss 0.007425727356118306 accuracy 0.9979807045097598
Val loss 0.4806714583845699 accuracy 0.9095330739299611

2021-01-30 09:15:28.945492 Epoch 5 END
--------------------
Best val accuracy: 0.9197470817120622
```

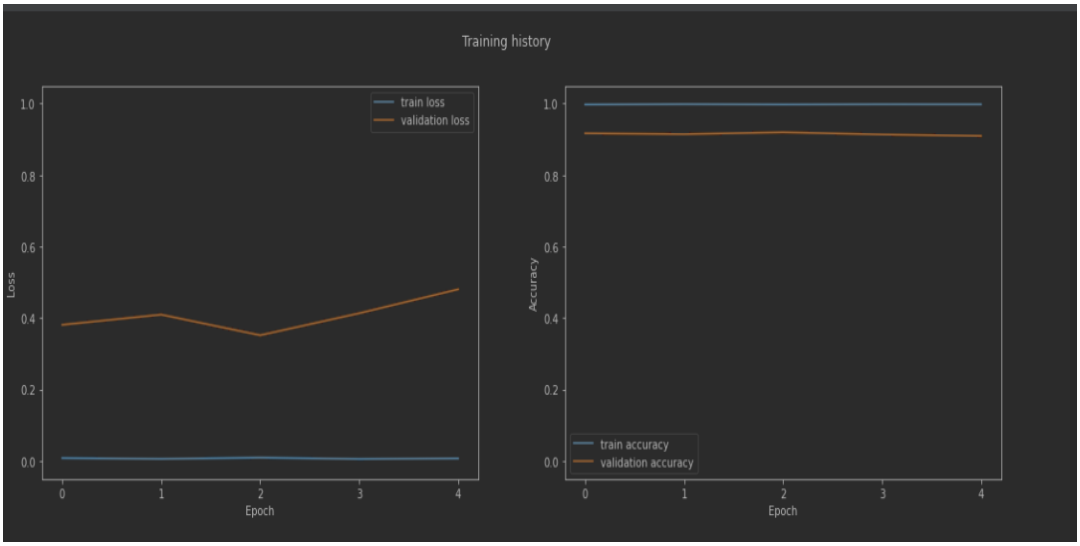
Εικόνα 50: Αποτελέσματα εκπαιδευτικού βρόγχου

Η καλύτερη ακρίβεια του μοντέλου για αριθμό επαναλήψεων 5 είναι 92% όπως φαίνεται από την παραπάνω εικόνα.

Στη συνέχεια του αλγόριθμου δημιουργούνται τα διαγράμματα ακρίβειας και σφάλματος ανά επανάληψη χρησιμοποιώντας τη βιβλιοθήκη matplotlib.

```
def plot_training_history(history):

    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(18, 6))
    ax1.plot(history['train_loss'], label='train loss')
    ax1.plot(history['val_loss'], label='validation loss')
    ax1.xaxis.set_major_locator(MaxNLocator(integer=True))
    ax1.set_ylim([-0.05, 1.05])
    ax1.legend()

    ax1.set_ylabel('Loss')
    ax1.set_xlabel('Epoch')
    ax2.plot(history['train_acc'], label='train accuracy')
    ax2.plot(history['val_acc'], label='validation accuracy')
    ax2.xaxis.set_major_locator(MaxNLocator(integer=True))
    ax2.set_ylim([-0.05, 1.05])
    ax2.legend()
    ax2.set_ylabel('Accuracy')
    ax2.set_xlabel('Epoch')
    fig.suptitle('Training history')


plot_training_history(history)
```

Εικόνα 51: Δημιουργία διαγραμμάτων ακρίβειας και σφάλματος ανά επανάληψη



Εικόνα 52: Διαγράμματα ακρίβειας και σφάλματος ανά επανάληψη

Παρατηρείται ότι η μεγαλύτερη ακρίβεια είναι στην Τρίτη επανάληψη καθώς και τα λιγότερα σφάλματα. Με την τύπωση των διαγραμμάτων ολοκληρώνεται και το πρόγραμμα της εκπαίδευσης του μοντέλου και ακολουθεί η εφαρμογή και η δοκιμή του.

8. Δοκιμή μοντέλου αυτόνομης οδήγησης και συμπεράσματα

Η εφαρμογή του παραγόμενου μοντέλου αυτόνομης οδήγησης είναι εξίσου σημαντική καθώς μας οδηγεί σε συμπεράσματα σχετικά με την οδηγική συμπεριφορά του οχήματος

8.1 Δοκιμή μοντέλου αυτόνομης οδήγησης

Μετά την ολοκλήρωση της εκπαίδευσης και την αποθήκευση του μοντέλου αυτόνομης οδήγησης του οχήματος ακολουθεί η δοκιμή του. Η δοκιμή ξεκινάει εκτελώντας την κλάση autopilot.py πάνω στον κεντρικό υπολογιστή του οχήματος.

Η εκτέλεση της κλάσης ξεκινάει με την ενεργοποίηση των GPIO του Jetson και τη σύνδεσή του με τον MQTT broker. Στη συνέχεια φορτώνεται το μοντέλο και ενεργοποιείται η κάμερα ώστε να δίνει σαν είσοδο στο μοντέλο τις εικόνες που «τραβάει» κατά τη διάρκεια της κίνησης.

```
def load_trt_model():
    print('Start loading TRT model')
    model_trt = TRTModule()
    model_trt.load_state_dict(torch.load(FILES_PATH + model_name + '_trt.pt'))
    print('TRT model loaded')
    return model_trt


GPIO.setmode(GPIO.BCM)
mqtt = MqttCar()

device = (torch.device('cuda') if torch.cuda.is_available() else torch.device('cpu'))
print(f"Training on device {device}.")
car_camera = None
try:
    loaded_model = load_trt_model()
    preprocessor = ImagePreProcessor(224, 224)

    mqtt.start()
    led = SingleLED(GPIO)
    led.onStartCarLEDs()

    car = Car(GPIO, speed=42)

    car_camera = CarCamera(autopilot=True, record_stops=False, video_output=True)
    car_camera.start()
    keyboard_reader = KeyboardReader()
    mqtt.publish(car.move, const.car_move_topic)
    mqtt.publish(car.speed, const.car_speed_topic)
    mqtt.publish(const.autopilot_mode, const.car_mode_topic)
    counter = 0
    output = 'stop'
    output_percent = 0
    speed_limit_detected = False
```

Εικόνα 53: Εκκίνηση δοκιμής μοντέλου

Στο τέλος δημοσιεύονται πληροφορίες σε topics του MQTT ότι το μοντέλο έχει φορτωθεί και είναι έτοιμο να λειτουργήσει, ανάβοντας και τα 3 led λαμπάκια του οχήματος.

Έπειτα ξεκινάει μία επαναληπτική δομή “while” όπου σε κάθε επανάληψη το όχημα παίρνει μία εικόνα από την κάμερα την περνάει μέσα από το μοντέλο και παίρνει σαν αποτέλεσμα την κίνηση που πρέπει να πραγματοποιήσει. Επίσης δημοσιεύει το κάθε αποτέλεσμα στο αντίστοιχο MQTT topic.

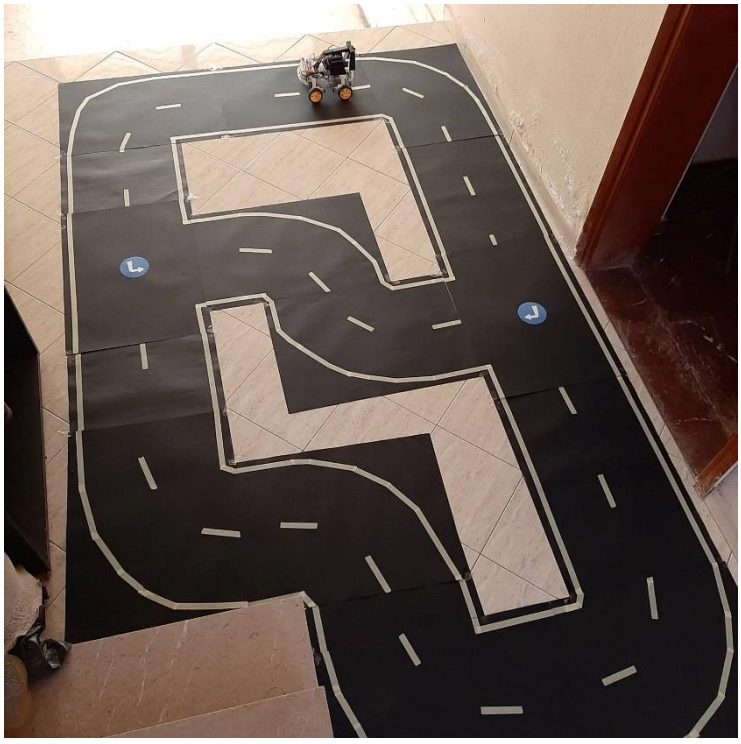
```
while True:

    image = car_camera.read(showMoves=True, move=output, output_percent=output_percent)

    if mqtt.speed_value is not None:
        car.set_speed(mqtt.speed_value)

    if mqtt.autopilot_status == 'ON':
        led.turnOn(const.blue_LED_pin)
        res = loaded_model(preprocessor.preprocessImage(image).to(device=device).half())
        _, index = torch.max(res, 1)

        sortedRes, indices = torch.sort(res, descending=True)

        percentage = torch.nn.functional.softmax(res, dim=1)[0] * 100
        output = class_names[index[0]]
        output_percent = percentage[index[0]].item()
        print(class_names[index[0]], percentage[index[0]].item())

        if (output == 'speed_limit' and speed_limit_detected) or (output == 'speed_limit_end' and not speed_limit_detected):
            print(print(class_names[indices[0][1]], percentage[indices[0][1]].item()))
            output = class_names[indices[0][1]]

        counter = counter + 1
        if output == 'stop':
            if output != car.move:
                mqtt.publish(output, const.car_move_topic)
            car.stop()

        elif output == 'forward_left':
            if output != car.move:
                mqtt.publish(output, const.car_move_topic)
            car.forwardLeft()
        elif output == 'forward_right':
            if output != car.move:
                mqtt.publish(output, const.car_move_topic)
            car.forwardRight()
        elif output == 'forward':
```

Εικόνα 54: Επαναληπτική δομή δοκιμής μοντέλου

Η διακοπή της χρήσης του αυτόματου πιλότου πραγματοποιείται με εντολή από το πληκτρολόγιο.

Το όχημα δοκιμάστηκε σε διάφορες πίστες, διαφορετικές από εκείνες που χρησιμοποιήθηκαν για τη συλλογή των δεδομένων εκπαίδευσης του μοντέλου. Ακολουθούν ορισμένες ενδεικτικές εικόνες.

Εικόνα 55: Πίστα δοκιμών 1



Εικόνα 56: Πίστα δοκιμών 2

8.2 Συμπεράσματα

Η εργασία αυτή είχε σαν αποτέλεσμα τη δημιουργία ενός πλήρως λειτουργικού οχήματος, με μία πλήρη διαδικασία τεχνητής νοημοσύνης, από τη συλλογή των δεδομένων έως και την εφαρμογή του παραγόμενου μοντέλου. Το μοντέλο επιτρέπει στο όχημα να κινείται αυτόνομα σε τυχαία δημιουργημένες πίστες που δεν έχει ξαναδεί ποτέ.

Από τις δοκιμές που πραγματοποιήθηκαν προέκυψαν τα εξής συμπεράσματα:

1. Το όχημα όταν κινούνταν σε χαμηλή ταχύτητα ανταποκρινόταν πολύ καλά καθώς δεν έφευγε από την πορεία του. Όσο αυξανόταν η ταχύτητα το μοντέλο δεν ανταποκρινόταν τόσο καλά καθώς αυξανόταν η ανάγκη για ταχύτερη επεξεργασία εικόνων, η οποία είχε ένα όριο λόγω της υπολογιστικής δύναμης του Jetson Nano. Το παραπάνω είχε σαν αποτέλεσμα το όχημα να βρίσκεται εκτός δρόμου και να σταματάει
2. Στις περιπτώσεις των εμφανιζόμενων εμποδίων το όχημα σταματούσε όταν εντόπιζε πιο μεγάλα αντικείμενα ενώ τα μικρότερα δυσκολευόταν περισσότερο να τα εντοπίσει.
3. Όσον αφορά την οριζόντια σήμανση του δρόμου το όχημα δυσκολευόταν σε ορισμένες περιπτώσεις, ειδικά όταν είχε μεγάλη ταχύτητα, να αναγνωρίσει τα σήματα με αποτέλεσμα να τα προσπερνά χωρίς κάποια αντίδραση

Παρόλα' αυτά υπάρχουν τομείς για περαιτέρω έρευνα ώστε να υπάρξει συνολική βελτίωση στην οδηγική συμπεριφορά του οχήματος και να ελαχιστοποιηθεί η πιθανότητα λανθασμένου χειρισμού.

8.2.1 Περιορισμοί του οχήματος

Ένα από τα πιο σημαντικά εμπόδια για τη δοκιμή σε πραγματικά οχήματα είναι ότι το όχημα που κατασκευάστηκε και χρησιμοποιήθηκε έχει πολλές διαφορές από ένα πραγματικό αυτοκίνητο. Στο όχημα δεν υπάρχει άξονας στις ρόδες ώστε να στρίβουν. Το στρίψιμο πραγματοποιείται με μείωση της ταχύτητας κίνησης των τροχών. Επίσης οι διαστάσεις του είναι σχετικά μεγάλες με το «οδόστρωμα» που χρησιμοποιήθηκε.

8.2.2 Υπολογιστικοί πόροι

Όπως αναφέρθηκε και παραπάνω σε περιπτώσεις όπου χρειαζόταν γρηγορότερη επεξεργασία εικόνας και ταχύτερη απάντηση του μοντέλου για την κατηγοριοποίηση το όχημα βρισκόταν εκτός δρόμου ή μπερδευόταν. Επομένως κρίνεται απαραίτητη η χρήση ενός ισχυρότερου υπολογιστή ώστε να είναι η οδήγηση ασφαλής.

Η αύξηση των υπολογιστικών πόρων θα βοηθούσε να διαχωριστεί η εύρεση οριζόντιας σήμανσης και εμποδίων από το υπάρχων μοντέλο. Πιο συγκεκριμένα θα δημιουργούνταν ένα άλλο μοντέλο εντοπισμού αντικειμένων, το οποίο θα εντόπιζε εμπόδια και σήμανση και θα έτρεχε παράλληλα με το μοντέλο πλοήγησης.

Εικόνα 57: μοντέλου εντοπισμού αντικειμένων

Τέλος κατά τη διάρκεια συλλογής των δεδομένων ο αισθητήρας απόστασης δεν ανταποκρινόταν σωστά λόγω έλλειψης υπολογιστικής ισχύος γι' αυτό και τα δεδομένα του δεν χρησιμοποιήθηκαν.

8.2.3 Οδόστρωμα και πίστα

Σαν οδόστρωμα χρησιμοποιήθηκε μαύρο χαρτί ώστε να μοιάζει όσο γίνεται περισσότερο σε δρόμο. Όμως στην πραγματικότητα το χρώμα, η ποιότητα και το μέγεθος των δρόμων διαφέρει από σημείο σε σημείο. Άρα η δοκιμή του μοντέλου σε πραγματικά οχήματα δυσκολεύει ακόμα περισσότερο.

8.2.4 Αισθητήρες

Στη συγκεκριμένη υλοποίηση χρησιμοποιήθηκε μόνο ένας αισθητήρας απόστασης, τοποθετημένος στο μπροστινό μέρος του οχήματος. Για μια πιο ολοκληρωμένη λύση και εφαρμογή σε πραγματικά οχήματα απαραίτητη είναι η χρήση πολλών αισθητήρων διαφορετικών ειδών π.χ. LiDAR για την ανίχνευση και αναγνώριση των γύρω αντικειμένων και η συνεργασία όλων αυτών με τα μοντέλα τεχνητής νοημοσύνης.

8.3 Τελικές σκέψεις

Αυτή η μεταπτυχιακή εργασία απέδειξε ότι είναι ένας εξαιρετικός τρόπος μέσω του οποίου μπορεί να μελετηθεί η τεχνητή νοημοσύνη και η μηχανική μάθηση μέσω μιας πρακτικής προσέγγισης. Αποδεικνύεται επίσης ότι τα αυτόνομα και αυτό-οδηγούμενα οχήματα, αν και σε μοντέλα μικρότερης κλίμακας, αναμένεται να αποτελέσουν μεγάλο μέρος στον τομέα των μετακινήσεων μελλοντικά και τομέα επενδύσεων σε πολλούς τομείς. Παρόλα αυτά, η υλοποίησή τους απαιτεί προσεκτική ανάπτυξη, δοκιμές και την αντιμετώπιση πολλών τεχνικών, νομικών και κοινωνικών προκλήσεων.

Βιβλιογραφία

[1] «Grobotronics,» [Ηλεκτρονικό]. Available: https://grobotronics.com/robot-smart-car-4wd-chassis-26cm.html.

[2] «Camera Mount,» Thingiverse, [Ηλεκτρονικό]. Available: https://www.thingiverse.com/thing:2707486/files.

[3] «Cr-20 Pro,» Creality, [Ηλεκτρονικό]. Available: https://www.creality.com/.

[4] «Polylactic acid,» [Ηλεκτρονικό]. Available: https://en.wikipedia.org/wiki/Polylactic_acid.

[5] «AUTODESK Tinkercad,» [Ηλεκτρονικό]. Available: https://www.tinkercad.com/.

[6] «Jetson Nano 2GB Full Case,» Thingiverse, [Ηλεκτρονικό]. Available: https://www.thingiverse.com/thing:4649425/files.

[7] «Panasonic NCR18650PF,» HELLAS digital, [Ηλεκτρονικό]. Available: https://www.hellasdigital.gr.

[8] «Intenso Q10000,» Intenso, [Ηλεκτρονικό]. Available: https://www.intenso.de.

[9] «IMX219-160 Camera,» Waveshare, [Ηλεκτρονικό]. Available: https://www.waveshare.com/wiki/IMX219-160_Camera.

[10] «SAMSUNG MB-MC32GA,» e-shop, [Ηλεκτρονικό]. Available: https://www.e-shop.gr/samsung-mb-mc32ga-eu-evo-plus-32gb-micro-sdhc-u1-class-10-adapter-p-PER.577044.

[11] «HP ZBook 15 G2 Mobile Workstation,» HP, [Ηλεκτρονικό]. Available: https://support.hp.com/ie-en/document/c04488080.

[12] «JetPack SDK,» NVIDIA Developer, [Ηλεκτρονικό]. Available: https://developer.nvidia.com/embedded/jetpack.

[13] «balenaEtcher,» Balena, [Ηλεκτρονικό]. Available: https://etcher.balena.io/.

[14] «Get Started With Jetson Nano Developer Kit,» NVIDIA Developer, [Ηλεκτρονικό]. Available: https://developer.nvidia.com/embedded/learn/get-started-jetson-nano-devkit#intro.

[15] «NVIDIA Jetson Linux Driver Package Software Features,» NVIDIA , [Ηλεκτρονικό]. Available: https://docs.nvidia.com/jetson/archives/l4t-archived/l4t-3261/index.html#page/Tegra%20Linux%20Driver%20Package%20Development%20Guide/hw_setup_jetson_io.html.

[16] «Autonomous_car,» Github, [Ηλεκτρονικό]. Available: https://github.com/KouXou/Autonomous_car.

[17] «pandas,» [Ηλεκτρονικό]. Available: https://pandas.pydata.org/.

[18] «MQTT,» [Ηλεκτρονικό]. Available: https://mqtt.org/.

[19] «eclipse-mosquitto,» dockerhub, [Ηλεκτρονικό]. Available: https://hub.docker.com/_/eclipse-mosquitto.

[20] «docker,» [Ηλεκτρονικό]. Available: https://www.docker.com/.

[21] «WebSocket,» Wikipedia, [Ηλεκτρονικό]. Available: https://en.wikipedia.org/wiki/WebSocket.

[22] «mqtt_car.py,» Github, [Ηλεκτρονικό]. Available: https://github.com/KouXou/Autonomous_car/blob/master/mqtt_connection/mqtt_car.py.

[23] «Angular,» [Ηλεκτρονικό]. Available: https://angular.io/.

[24] «Typescript,» [Ηλεκτρονικό]. Available: https://www.typescriptlang.org/.

[25] «HTML,» Wikipedia, [Ηλεκτρονικό]. Available: https://en.wikipedia.org/wiki/HTML.

[26] «ngx-mqtt,» Github, [Ηλεκτρονικό]. Available: https://github.com/sclausen/ngx-mqtt.

[27] Y. B. a. A. C. Ian Goodfellow, «Deep Learning».*MIT Press.*

[28] K. He, «Deep Residual Learning for Image Recognition,» *CoRR abs/1512.03385,* 2015.

[29] A. G. Howard, «MobileNets: Efficient Convolutional Neural Networks for Mobile,» *CoRR abs/1704.04861,* 2017.