Державний вищий навчальний заклад
“Прикарпатський національний університет імені Василя Стефаника”
Кафедра інформаційних технологій

УДК 004

ДИПЛОМНИЙ ПРОЕКТ

Тема: Розробка автономного мобільного робота на основі машинного навчання

Спеціальність: 121 Інженерія програмного забезпечення

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДП.ІПЗ-29.ПЗ
(позначення)

Рецензент
проф. Кузь М.В.
(посада) (підпис) (дата) (розшифровка підпису)

Студент
ІПЗ-41 Хортюк С.В.
(шифр групи) (підпис) (дата) (розшифровка підпису)

Нормоконтролер
проф. Кузь М.В.
(посада) (підпис) (дата) (розшифровка підпису)

Керівник дипломного проекту
доц. Козленко М.І.
(посада) (підпис) (дата) (розшифровка підпису)

Допускається до захисту

Завідувач кафедри

доц. Козленко М.І.
(посада) (підпис) (дата) (розшифровка підпису)

2020
(рік)

Державний вищий навчальний заклад
«Прикарпатський національний університет імені Василя Стефаника»
Факультет математики та інформатики Кафедра інформаційних технологій
Спеціальність Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Завідувач кафедри_____
_____
„_____”_____ 20___р.

ЗАВДАННЯ
НА ВИКОНАННЯ ДИПЛОМНОГО ПРОЕКТУ

Студенту Хортюку Сергію Володимировичу
(прізвище, ім’я, по батькові студента)

1. Тема проекту Розробка автономного мобільного робота на основі машинного навчання

затверджена розпорядженням по факультету математики та інформатики від „25” жовтня 2019р. № 7

2. Термін здачі студентом закінченого проекту 22 травня 2020 р.

3. Вихідні дані до дипломного проекту технології програмування програмної частини – Python.

4. Зміст пояснювальної записки (перелік питань, що їх належить опрацювати)
1. Аналіз предметної області та постановка задачі
2. Вибір технологій для реалізації завдання. Архітектура та алгоритми
3. Реалізація програмного забезпечення
4. Бізнес план

5. Перелік графічного матеріалу (з точним забезпеченням обов’язкових креслень): “Мета роботи”, “Зображення робота”, “Зобраення основних програмних модулів”, “Зображення етапу отримання картинки з камери”, “Зображення етапу отримання команди від користувача”, “Зобрадення етапу обробки зображення нейронною мережею”, “Зображення подачі сигналів на органи управління”, “Зображення збереження данних отриманих з камери”, “Архітектура нейронної мережі”, “Схематичний вигляж нейронної мережі”, “Ефективність нейронної мережі”, “Схема підключення апаратної частини”, “Зображення апаратної частини робота”, “Зображення робота на трасі”, “Висновок”

6. Дата видачі завдання _____

Керівник _____ _____
(підпис) (розшифровка підпису)

Завдання прийняв до виконання _____ _____
(підпис) (розшифровка підпису)

КАЛЕНДАРНИЙ ПЛАН

Номер і назва етапів дипломного проекту	Термін виконання етапів проекту	Примітка
1. Аналіз предметної області та постановка задачі	10.09.2019	виконав
2. Вибір технологій для реалізації завдання. Архітектура та алгоритми	12.01.2020	виконав
3. Реалізація програмного забезпечення	04.03.2020	виконав
4. Бізнес-план	09.05.2020	виконав
5. Оформлення пояснювальної записки	18.05.2020	виконав

Студент _____ Хортюк С.В.
(підпис) (розшифровка підпису)

Керівник проекту _____ Козленко М.І.
(підпис) (розшифровка підпису)

РЕФЕРАТ

Пояснювальна записка: 63 сторінки (без додатків), 41 рисунок, 22 джерела, 2 додатки на 34 сторінках.

Ключові слова: автономний робот, нейронна мережа, Raspberry PI, Python, Donkey Car.

Об`єктом дослідження є: автономний мобільний робот на основі машинного навчання.

Мета роботи: Аналіз та розробка автономного мобільного робота на основі машинного навчання для участі у міжнародному конкурсі з робототехніки Sumo Challenge.

Стислий опис тексту пояснювальної записки:

В даному дипломному проекті розглядається розробка програмного та апаратного забезпечення, а також їх архітектури для автономного мобільного робота на основі машинного навчання. Який приймав участь у міжнародному конкурсі з робототехніки Sumo Challenge, та здобув перше місце в категорії Fast and Safe (швидкість та безпека).

ABSTRACT

Explanatory note: 63 pages (without appendix), 41 figures, 22 references, 2 appendix on 34 pages.

Key words: autonomous robot, neural network, Raspberry PI, Python, Donkey Car.

Object of study: Analysis and development of an autonomous mobile robot based on machine learning to participate in the international robotics competition Sumo Challenge.

Brief description of the text of the explanatory note:

This diploma project considers the development of software and hardware, as well as their architecture for autonomous mobile robot based on machine learning. He took part in the international robotics competition Sumo Challenge, and won first place in the category of Fast and Safe.

ЗМІСТ

ВСТУП

На сьогоднішній день ми можемо спостерігати, як швидко розвиваються технології, і поступово комп'ютер починає заміняти людину в різних сферах діяльності. Адже людина не в змозі так швидко оперувати великими об'ємами даних та на їх основі приймати правильні рішення. Хоча досить часто від цих рішень залежить життя інших людей. Саме це і є причиною того, чому автономні автомобілі зараз настільки популярні та розвиваються з неймовірною швидкістю.

Завданням дипломної роботи є розробка автономного мобільного робота на основі машинного навчання для участі у міжнародному конкурсі з робототехніки Sumo Challenge. Захід проходив у Технологічному Університеті Лодзі (Politechnika Łódzka) (м. Лодзь, Республіка Польща).

Робот готувався для категорії TomTom Self-Driving Cars. В цій категорії машина, обладнана комп'ютерним зором та штучними нейронними мережами, повинна була проїхати трасу, яка відтворювала умови реальної дороги, без будь-якої допомоги зі сторони людини. Оцінювали автомобіль за те, як швидко він їде та його акуратність на трасі.

Для досягнення мети дипломної роботи поставлено такі завдання:

- Розглянути основні принципи і методи розробки та моделювання автономного мобільного робота;
- Проаналізувати правила конкурсу та знайти оптимальне вирішення поставленого завдання;
- На основі проаналізованих даних розробити програмне забезпечення та робота для участі в конкурсі з робототехніки.

На проаналізованих даних, розробити програмне забезпечення та робота для участі в конкурсі з робототехніки.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Розвиток автономних мобільних роботів

Робот – це машина, яка програмується людиною за допомогою комп'ютера, та здатна автоматично виконувати визначений ряд дій. Роботи широко застосовуються на виробництві для виконання багато повторних простих дій, що зазвичай виконувались людьми [1].

Перші роботи були маніпулятивними, тобто керування здійснювалось людиною за допомогою зовнішнього керуючого пристрою. Такі машини призначені для виконання одноманітної роботи, такої як складання, переміщення. До таких роботів також належать рятувальні машини – пристрої, які замінюють людину в небезпечному середовищі. Вони виконують такі ризикові завдання, як пошук вцілілих людей після катастрофи, працюють на великих водних глибинах, у космосі. Наприклад, після японського землетрусу в Японії 2011 року, загін PackBot (військові роботи) було направлено на розслідування атомної електростанції Фукусіма Даїчі для оцінки ступеня шкоди [2].

Після закінчення Другої світової війни спостерігався стрімкий ріст в галузі робототехніки. Розпочалась технологічна гонка з метою створення автономного робота.

Автономний робот – це робот, який виконує поведінку або завдання з високим ступенем самостійності (без зовнішньої допомоги). Автономна робототехніка зазвичай вважається поєднанням штучного інтелекту, робототехніки та інформаційної інженерії [1].

Робота можна вважати автономним, якщо він виконує наступні вимоги:

- Отримує інформацію про зовнішнє довкілля;
- Уникає ситуацій, шкідливих для себе, майна або людини;
- Працює без втручання людини;

- Переміщує себе повністю або частково в межах свого робочого простору без втручання людини.

Для автономного робота також властива адаптація до нового середовища, та знаходження нових шляхів розв’язання поставлених перед ним задач.

Повністю автономна роботизована система, з'явилася лише у другій половині 20-го століття. Перший програмований робот з цифровим керуванням Unimate було встановлено 1961 року для підняття гарячих шматків металу з машини для лиття під тиском і їх складання [2].

Історія автоматизації зазнала чіткого прориву в розвитку технологій на початку 90-х років минулого століття. NASA розгорнула свою першу автономну робототехнічну систему Sojourner на поверхні Марса. Цей невеликий робот (всього 23 фунти або 10,5 кг) виконував напів автономні операції на поверхні Марса в рамках місії Mars Pathfinder; обладнаний програмою уникнення перешкод, Соджорнер мав змогу планувати та орієнтуватися на маршрути для вивчення поверхні планети. Можливість приїжджих орієнтуватися з невеликими даними про своє оточення та навколишнє оточення дозволила йому реагувати на незаплановані події та предмети [2].

У 1994 році FDA очистив один з найуспішніших хірургічних апаратів, що підтримуються роботами. Кіберніж був винайдений Джоном Р. Адлером, і перша система була встановлена в Стенфордському університеті в 1991 році. Ця радіохірургічна система інтегрувала керовані зображеннями операції з робототехнічним позиціонуванням. Зараз Cyberknife розгорнуто для лікування пацієнтів з пухлинами мозку або хребта. Рентгенівська камера відстежує зміщення і компенсує рух, викликаний диханням [2].

Пізніше 2000 році Honda створила ASIMO - який, на той час, вважався "найдосконалішим у світі гуманоїдним роботом". ASIMO може бігати, ходити, спілкуватися з людьми, розпізнавати обличчя, навколишнє середовище, голоси, поставу і навіть взаємодіяти з його оточенням. Компанія Sony також розробила своїх власних автономних роботів Sony Dream, маленьких гуманоїдних роботів, розроблених для розваг [2].

На сьогодні автономні мобільні роботи застосовуються в різних сферах діяльності, ось декілька найпопулярніших сфер в яких вони вже добре зарекомендували себе:

- Марсоходи, які працюють в автономному режимі, досліджуючи різні планети, та передаючи інформацію людині;
- Навантажувачі на фабриках та великих підприємствах;
- Сільськогосподарські роботи для автоматичного збору/посіву врожаю;
- Роботи - кур'єри;
- Військові роботи;
- Автопілоти.

У жовтні 2000 року ООН підрахувала, що у світі налічується 742 500 промислових роботів, причому більше половини з них використовуються в Японії [3].

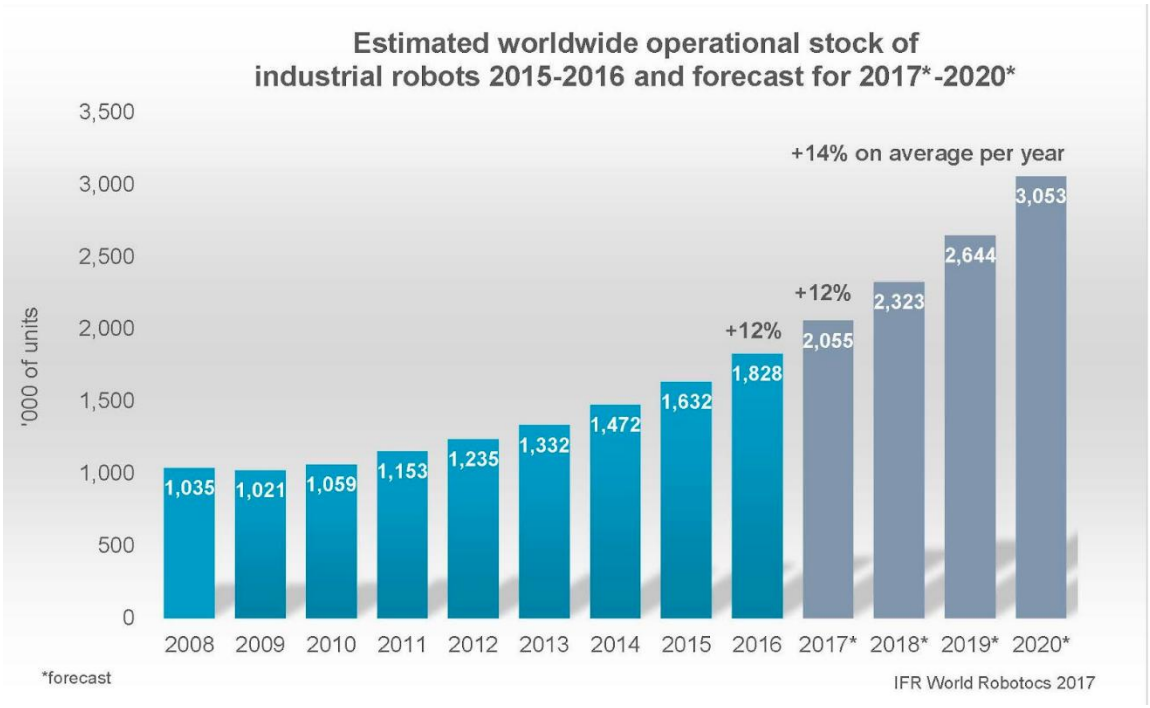

Рисунок 1.1 - Графік приросту кількості промислових роботів

На сьогоднішній день спостерігається експоненціальне зростання кількості автоматизованих роботів, що використовуються у промисловості (рис.1.1).

Це спричинило стрімкий ріст популярності різного роду змагання серед автономних мобільних роботів. Саме тому команда Прикарпатського національного університету взяла участь у міжнародному конкурсі з робототехніки Sumo Challenge. Захід проходив вже вдванадцяте у Технологічному Університеті Лодзі (Politechnika Łódzka) (м. Лодзь, Республіка Польща).

Команда робототехніків Прикарпатського університету успішно виступила в категорія TomTom Self-Driving Cars. За програмою тут відбулися змагання машин, обладнаних комп'ютерним зором та штучними нейронними мережами. Нагороду за найкращу швидку та безпечну машину (Fast and Safe) здобув автономний робот, який був спроектований та розроблений в межах дипломного проекту [4].

1.2 Пошук та аналіз існуючих аналогів на ринку

Сьогодні провідні автомобільні компанії направили свої зусилля на розробку та будівництво машин, що являються автономними роботами, а керування якими відбувається за допомогою штучного інтелекту на основі машинного навчання.

Твердим лідером в розробці автономних автомобілів (автопілота) на даний момент можна вважати компанію Tesla. Саме вони перші запустили серійне виробництво автомобілів, які були обладнані автопілотом та могли керуватись без втручання водія [5].

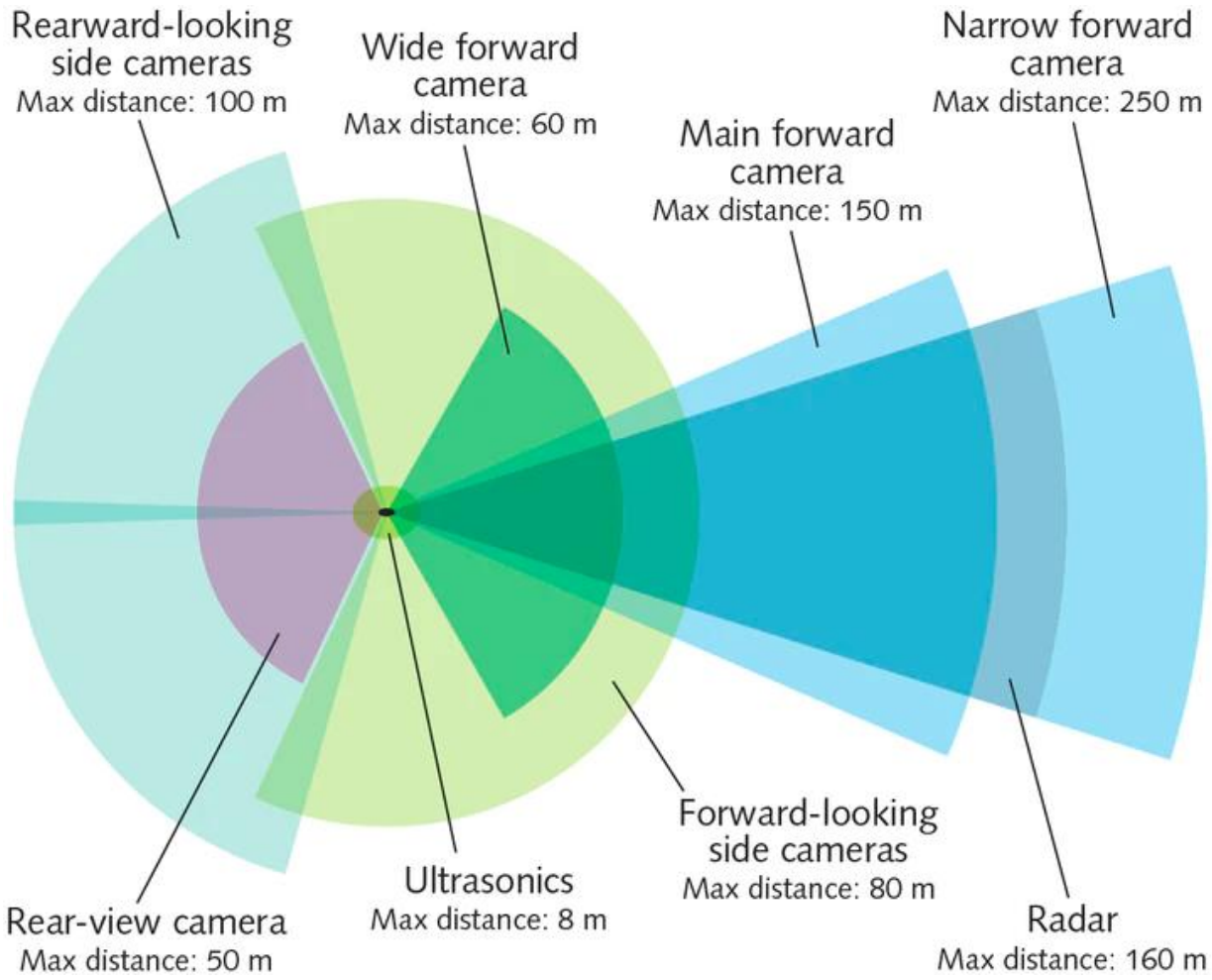

Рисунок 1.2 - Ілюстрація роботи сенсорів автомобіля Tesla

Парк Tesla налічує приблизно 500 000 транспортних засобів – це фантастичний ресурс. Кожен з автомобілів обладнаний великою кількістю сенсорів, зокрема вісім камер об'ємного звуку, що забезпечують 360 градусів видимості навколо машини на відстані до 250 метрів; дванадцять оновлених ультразвукових датчиків, які доповнюють це бачення та допомагають виявити як тверді, так і м'які предмети на відстані майже 500 метрів; радар, що спрямований вперед, і надає додаткові дані з покращеною обробкою про світ, які дають змогу побачити об’єкти через сильний дощ, туман, пил і навіть автомобіль попереду (рис. 1.2) [5].

Щоб зрозуміти всі ці дані, бортовий комп'ютер запускає нову розроблену Tesla нейронну мережу для програм зору, сонарної та радіолокаційної обробки даних. Разом ця система надає уявлення про світ, до якого водій не може

отримати доступ, оскільки можливості людського зору не дають змоги бачити в усіх напрямках одночасно, а також програють датчикам в здатності бачити на далеку відстань та в умовах обмеженого огляду, спричиненого погодними умовами, перешкодами тощо [6].

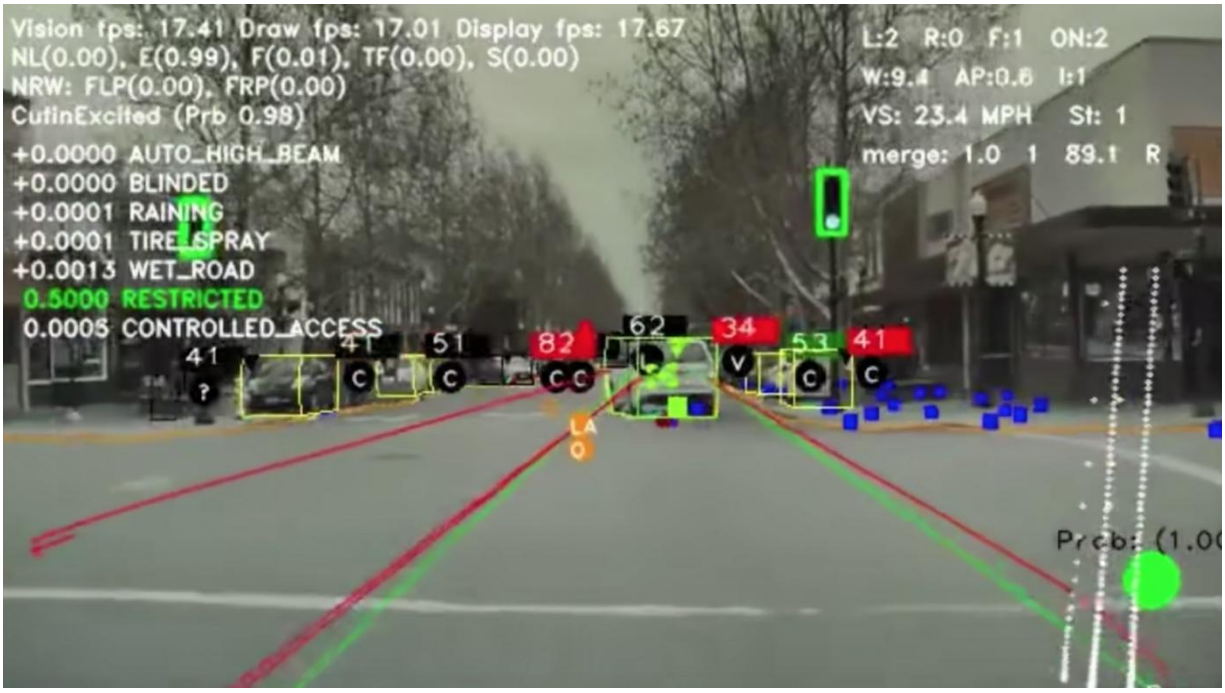

Рисунок 1.3 - Абстрактне зображення сцени з візуалізованими об'єктами

Будь-який раз, коли автономний робот робить невірний прогноз щодо машини чи пішохода, він може зберегти знімок даних, щоб пізніше завантажити та додати до навчального набору Tesla. На сьогодні, машина може завантажувати абстрактні зображення сцен (де об'єкти візуалізуються як кольорово-кубові форми та видається інформація про рівень пікселів), створені нейронними мережами та комп'ютерним зором, а не завантажувати відео (рис 1.3). Це кардинально зменшує пропускну здатність і вимоги до зберігання та завантаження цих даних [6].

Насправді всі транспортні засоби Tesla - незалежно від того, ввімкнено автопілот чи ні – надсилають дані безпосередньо в хмару. Це зумовлено постійною роботою різного роду сенсорів та камер.

Програмне забезпечення ефективно аналізує отримані дані з усіх транспортних засобів Tesla за допомогою зовнішніх сенсорів, а також обробляє

інформацію про водіїв, отриману внутрішніми сенсорами, які збирають різного роду дані, такі як розміщення рук водія на органах керування та способи їх взаємодії з автомобілем. Це допомагає покращувати роботу штучного інтелекту автономного робота.

На основі проаналізованих даних, можна виділити декілька важливих етапів розробки автономного робота:

- Забезпечення автомобіль достатньою кількістю датчиків, задля охоплення інформації про навколишнє середовище, об’єкти та перешкоди в ньому;
- Швидка та своєчасна обробка інформації, яка була отримана з різного роду сенсорів;
- Побудова алгоритмів прийняття рішень на основі проаналізованих даних.

1.3 Постановка задачі

При проектуванні та програмуванні автономного мобільного робота, необхідно заздалегідь чітко визначити тип задачі, яку він повинен виконувати. В моєму випадку метою була участь в конкурсі з робототехніки, тому потрібно було чітко зрозуміти правила конкурсу, та відштовхуючись від них, розпочати розробку робота, який відповідав би вимогам змагань. В категорії “TomTom Self-Driving Cars” були чітко визначені наступні правила:

- Роботи виступають по черзі, один за одним, отже, немає потреби турбуватися про те, як об'їжджати інших роботів та уникати зіткнень з ними;
- Автономний автомобіль повинен зробити 10 кіл поспіль без сторонньої людської допомоги;
- У випадку повного з’їзду робота з траси (всі чотири колеса знаходяться за межами білої лінії) передбачено додавання штрафного часу – 2 секунди. Біла лінія вважається частиною траси, а заїзд на неї не вважається порушенням;

- Час проходження кола не буде зараховано у випадку, якщо робот знаходився за межами траси;
- Протягом одного кола робот повинен проїхати всі чотири частини траси, інакше час кола не буде зарахований;
- Перемогу здобуває автономний робот, який проїхав коло за найшвидший час;
- Людина може замінити модель нейронної мережі лише раз в період проходження десяти кіл;
- Автомобіль відповідає умовам категорії "Швидкий та Безпечний", якщо протягом 10 кіл поспіль є хоча б одне бездоганне коло;
- Коло вважається бездоганним у випадку, коли автомобіль залишається всередині траси на одне повне коло і не торкається білої лінії жодним з коліс.

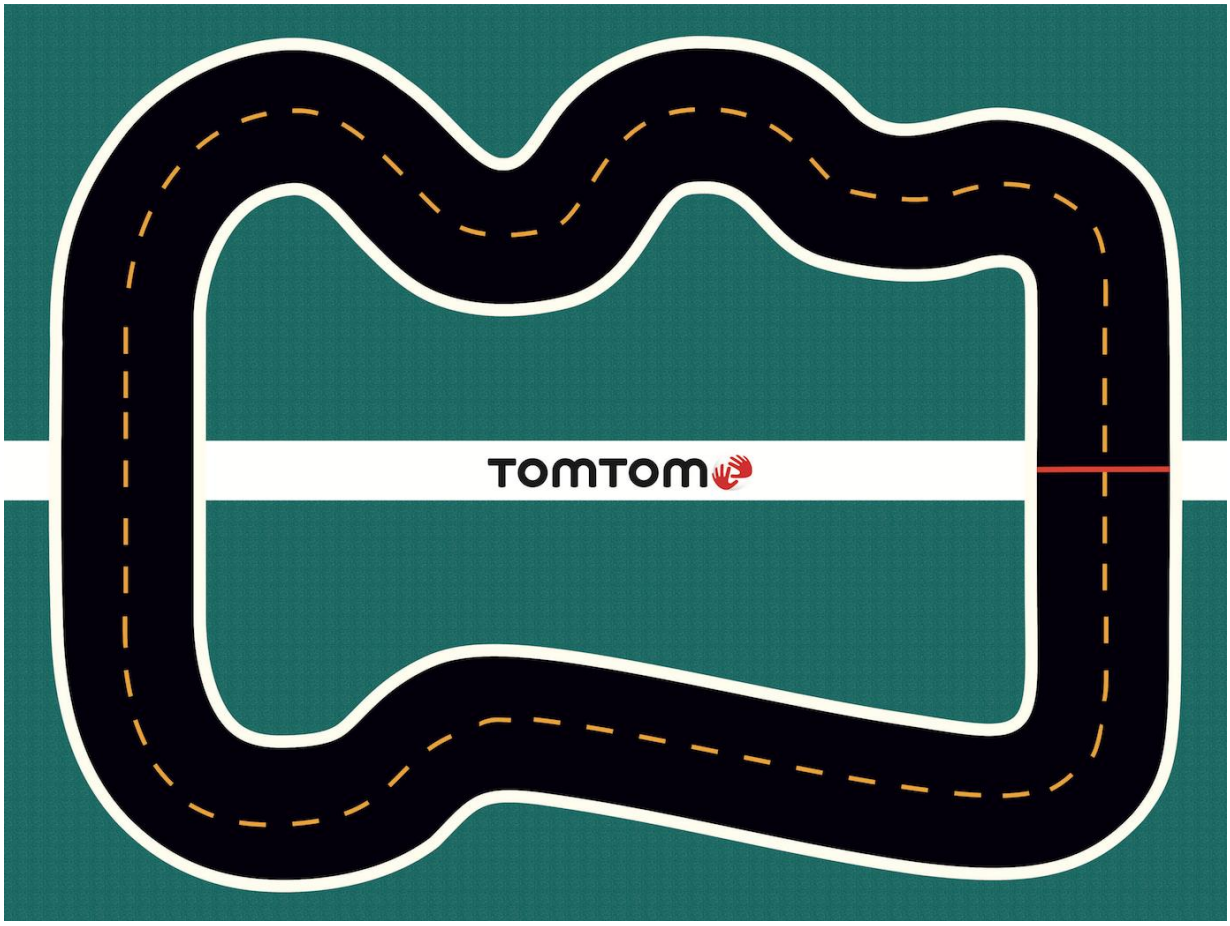

Рисунок 1.4 - Схема траси конкурсу

На рисунку 1.4 наведена схема траси конкурсу Sumo Challenge в категорії TomTom Self-Driving Cars, по якій будуть пересуватися автономні роботи. Рух буде відбуватися за годинниковою стрілкою. Межа старту та фінішу позначена червоною лінією. Ускладнення для проходження траси додає те, що вона є заокругленою, а також на ній розташована шикана (послідовність поворотів та заокруглень на трасі у формі латинської літери «S») з метою уповільнення робота. В правилах згадувалось про чотири контрольні точки, що знаходяться у верхньому лівому, верхньому правому, нижньому лівому та нижньому правому кутах траси. Кожен робот повинен проїхати усі чотири контрольні точки. Це зроблено для того, щоб автомобіль проїхав повне коло, не зрізаючи поворотів.

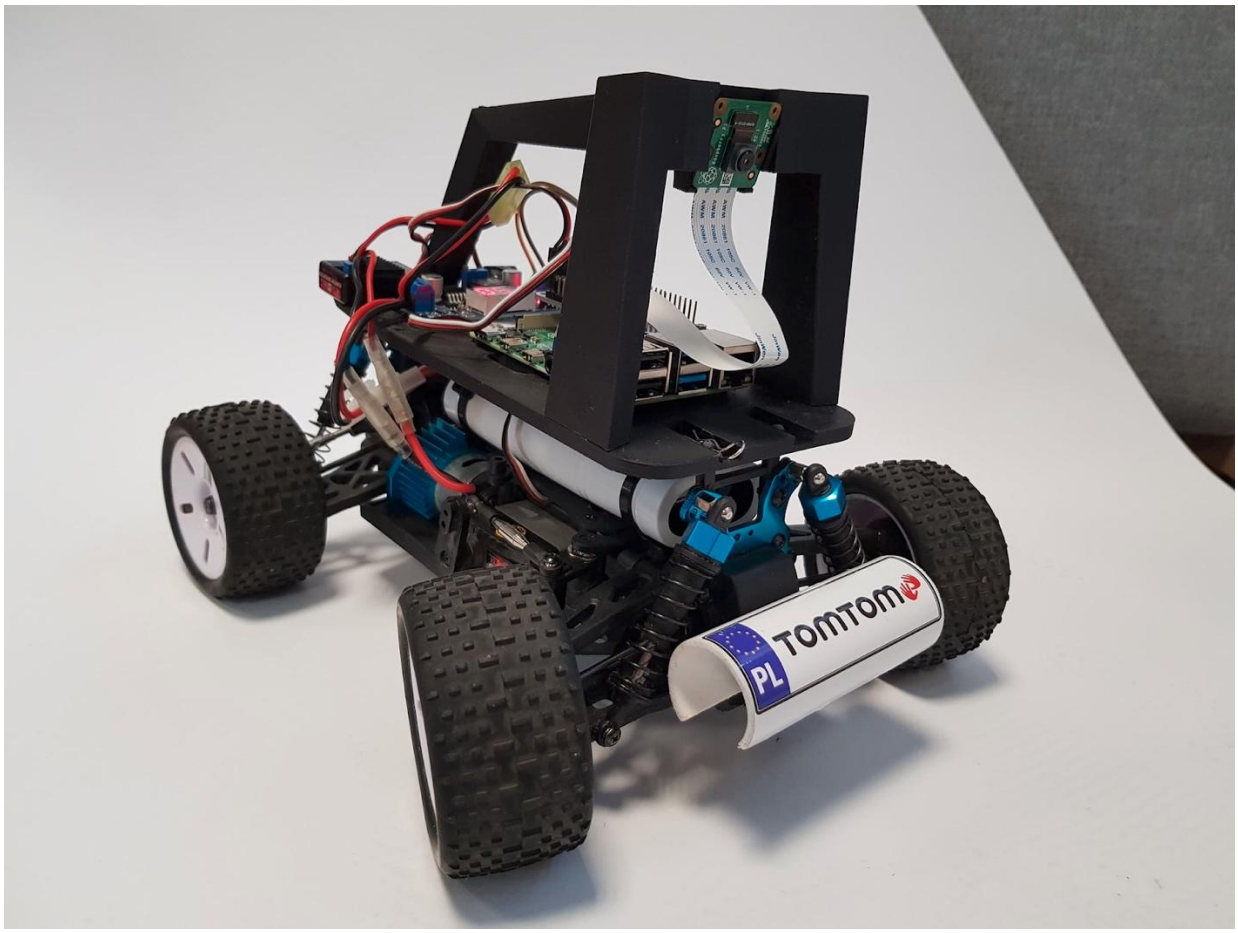

Рисунок. 1.5 - Зібрана модель робота

Для отримання перемоги у змаганнях було створено поетапний план розробки робота:

- Крок 1. Створення конструкції робота:
  - Перевірка наявності та справності всіх модулів;
  - Під'єднання та тестування роботи модулів;
- Крок 2. Конфігурація робота:
  - Встановлення необхідного програмного забезпечення;
  - Налаштування швидкості робота та елементів управління ним;
- Крок 3. Підготовка моделі нейронної мережі:
  - Збір даних, отриманих роботом, шляхом управління ним людиною;
  - Тренування моделі на основі зібраних даних;
- Крок 4. Тестування моделі:
  - Перевірка стабільності роботи моделі (максимальна кількість пройдених кіл без втручання зі сторони людини);
  - Перевірка безпечності проходження кіл (мінімальна кількість перетинів білої лінії за одне коло);
  - Перевірка швидкості проходження кола (мінімальний середній час проходження десяти кіл).

2 ВИБІР ТЕХНОЛОГІЙ ДЛЯ РКАЛІЗАЦІЇ ЗАВДАННЯ. АРХІТЕКТУРА ТА АЛГОРИТМИ

2.1 Python

Python – це інтерпретована, високорівнева, динамічна мова програмування, яка на сьогоднішній день використовується майже у всіх напрямках програмування. Вагомою перевагою є те, що мова фокусується на читабельності, оскільки це є одним з двадцяти програмних принципів (The Zen of Python by Tim Peters: Readability counts) [7].

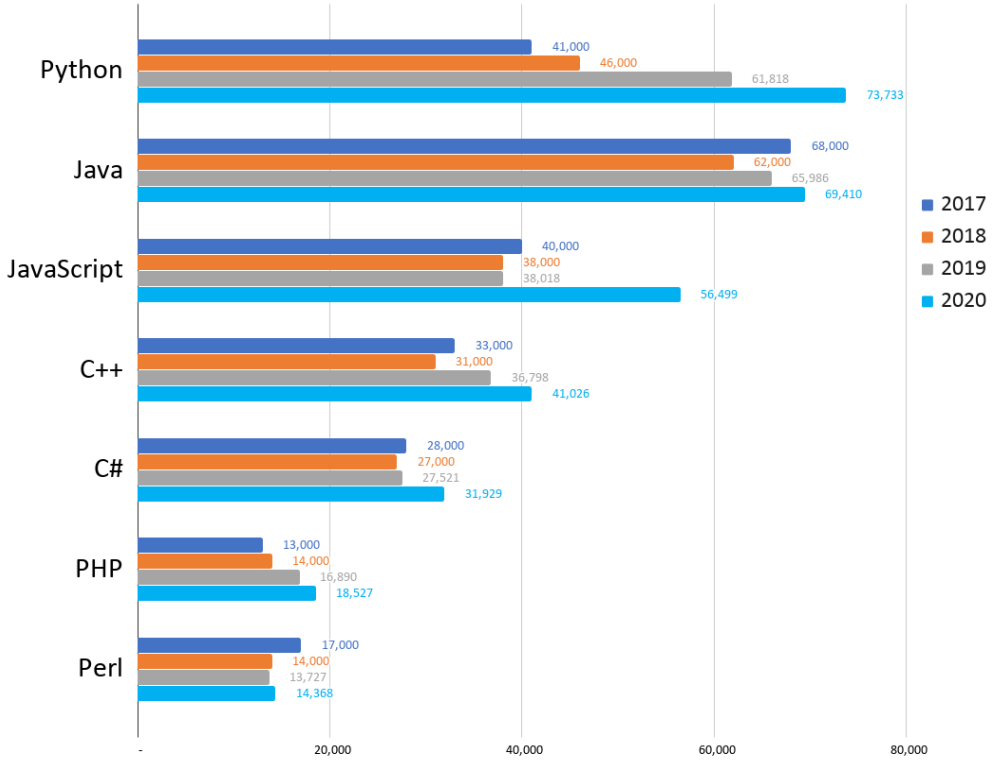

Рисунок 2.1 - Рейтинг мов програмування

Синтаксис та велика кількість бібліотек в Python допомагає програмістам вирішити поставлену перед ними задачу за значно меншу кількість кроків порівняно з Java або C++. Python, як мову програмування, було засновано в 1991 році розробником Ґвідо Ван Россумом. Велика кількість організацій широко використовує Python не тільки як основну мову програмування, а як мову для

написання окремих сервісів та скриптів для автоматизації, спричинено це високою гнучкістю та швидкістю вирішення задач. Зазвичай на Python передбачають імперативне та об'єктно-орієнтоване функціональне програмування. Нижче наведений графік популярності мови програмування Python за 2017-2020 роки (рис.2.1) [7].

Python пропонує стислий і читабельний код. Хоча складні алгоритми та універсальні робочі процеси стоять за машинним навчанням та штучним інтелектом (AI), простота Python дозволяє розробникам писати надійні системи. Програмісти можуть докладати всіх зусиль для роботи над вирішенням проблеми машинного навчання, а не зосереджуватися на технічних нюансах мови.

Крім того, Python приваблює багатьох розробників, оскільки її легко вивчити. Код Python зрозумілий людям, що спрощує побудову моделей для машинного навчання.

Багато програмістів кажуть, що Python більш інтуїтивно зрозумілий, ніж інші мови програмування. Інші вказують на безліч рамок, бібліотек та розширень, які спрощують реалізацію різних функціональних можливостей. Загальновизнано, що Python підходить для спільної реалізації, коли задіяні кілька розробників. Оскільки Python є мовою загального призначення, він може виконувати набір складних завдань машинного навчання та дозволяє швидко будувати прототипи, що дозволяють протестувати ваш продукт для цілей машинного навчання.

Впровадження алгоритмів AI та ML може бути складним і вимагає багато часу. Важливо створити добре структуроване та перевірене середовище, щоб розробники могли розробити найкращі рішення щодо розробки.

Щоб скоротити час програмування, розробники звертаються до декількох фреймворків і бібліотек Python. Бібліотека програмного забезпечення - це попередньо написаний код, який програмісти використовують для вирішення загальних завдань програмування. Python має багатий набір технологій, широкий

спектр бібліотек для штучного інтелекту та машинного навчання. Ось деякі з них:

- Keras, TensorFlow та Scikit-learn для машинного навчання;
- Pandas для аналізу даних загального призначення;
- NumPy для високоефективних наукових обчислень та аналізу даних;
- SciPy для розширених обчислень;
- Seaborn для візуалізації даних.

Саме тому я обрав мову програмування Python як основну мову реалізації дипломного проекту.

2.2 Python бібліотеки

Бібліотека Python – це сукупність функцій та методів, що дозволяє виконувати безліч дій без написання коду. Наприклад, бібліотека зображень Python PIL (Python Image Library) є однією з основних бібліотек для управління зображеннями в Python. Pillow - це активно розвинена гілка PIL. Open-CV (Open Computer Vision) - це бібліотека Python (також пов'язується з C++, C# тощо), спрямована на комп'ютерне бачення та обробку зображень у реальному часі. Вона використовує Numpy, іншу бібліотеку для числових операцій. Кожна бібліотека в Python містить величезну кількість корисних модулів, які ви можете імпортувати для свого щоденного програмування [8].

Велика стандартна бібліотека Python, яку зазвичай називають однією з найбільших її сильних сторін, надає інструменти, придатні для багатьох завдань. Для інтернет-додатків підтримуються багато стандартних форматів і протоколів, таких як MIME і HTTP. Він включає в себе модулі для створення графічних інтерфейсів користувача, підключення до реляційних баз даних, генерування псевдовипадкових чисел, арифметику з десятковими знаками довільної точності, що маніпулює регулярними виразами, та тестування [8].

Деякі частини стандартної бібліотеки охоплені специфікаціями (наприклад, реалізація інтерфейсу шлюзу веб-сервера (WSGI) wsgiref відповідає

PEP 333). Вони визначаються кодом, внутрішньою документацією та наборами тестів. Однак, оскільки більша частина стандартної бібліотеки є кросплатформним кодом Python, лише декілька модулів потребують зміни або переписування для варіантів реалізації [9].

Станом на листопад 2019 року індекс пакунків Python (PyPI - офіційне сховище сторонніх програмних продуктів Python), містить понад 200 000 пакетів з широким спектром функціональності, включаючи:

- Автоматизацію;
- Аналіз даних;
- Документацію;
- Машинне навчання;
- Системне адміністрування;
- Обробку зображень;
- Веб-фреймворки;
- Обробка тексту;
- Робота з базами даних;
- Та багато інших.

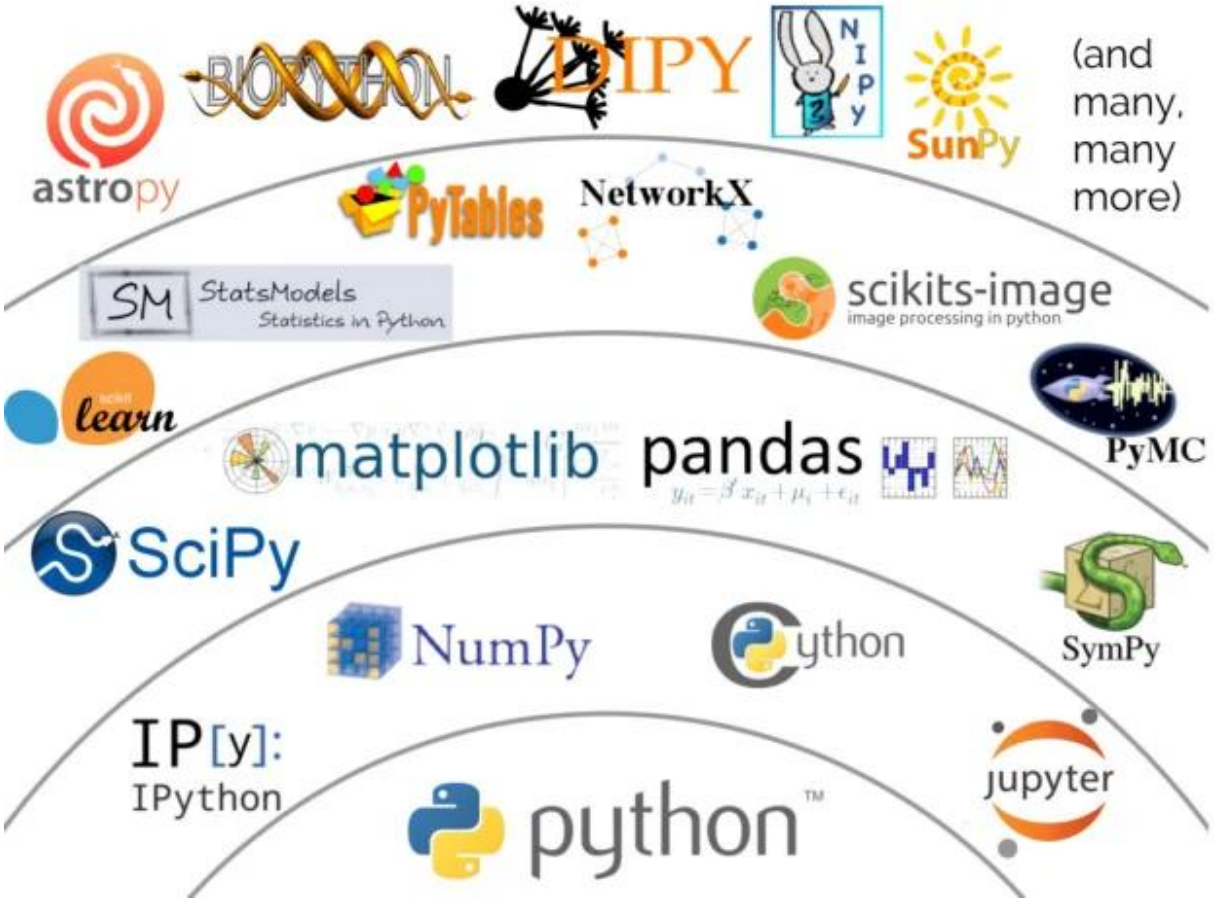

Рисунок 2.2 - Найпопулярніші бібліотеки на мові Python

На рисунку 2.2 зображені найпопулярніші бібліотеки, які використовуються інженерами при розробці програмного забезпечення на мові програмування Python.

2.3 Бібліотека Donkeycar

Donkeycar – це мінімалістична та модульна бібліотека для розробки автономних мобільних роботів на Python. Вона створена для любителів та студентів з акцентом на швидке експериментування та простоті внесення змін та доповнень користувачами [10].

Дивне ім’я розробники бібліотеки пояснюють тим, що метою бібліотеки є побудова чогось корисного, а саме віслюки були одними з перших одомашнених

зграйних тварин, також відомо що вони вперті та безпечні для малюків. Як тільки машина зможе пересуватися з одного боку міста в інший, розробники обіцяють перейменувати її в назву якоїсь небесної істоти [10].

Бібліотека має велику кількість вбудованих та протестованих рішень, які працюють «з коробки». Це позбавляє програміста від зайвого клопоту та надає змогу краще сфокусуватися на вирішені основної задачі. Також не менш важливим плюсом є те, що Donkeycar підтримує велику кількість сенсорів.

Для того, щоб ваш робот запрацював, вам достатньо буде написати всього десяток рядків коду, в яких необхідно описати модулі, які присутні на роботі, в якому режимі запускати робота та з якою частотою оновлювати вхідні дані (насправді бібліотека обладнана можливістю більш тонкої конфігурації робота, про цю конфігурацію буде розказано в третьому розділі дипломного проекту).

```
#Define a vehicle to take and record pictures 10 times per second.

import time
from donkeycar import Vehicle
from donkeycar.parts.cv import CvCam
from donkeycar.parts.datastore import TubWriter
V = Vehicle()

IMAGE_W = 160
IMAGE_H = 120
IMAGE_DEPTH = 3

#Add a camera part
cam = CvCam(image_w=IMAGE_W, image_h=IMAGE_H, image_d=IMAGE_DEPTH)
V.add(cam, outputs=['image'], threaded=True)

#warmup camera
while cam.run() is None:
    time.sleep(1)

#add tub part to record images
tub = TubWriter(path='./dat',
          inputs=['image'],
          types=['image_array'])
V.add(tub, inputs=['image'], outputs=['num_records'])

#start the drive loop at 10 Hz
V.start(rate_hz=10)
```

Рисунок 2.3 - Приклад вихідного коду програми

На (рис. 2.3) наведено приклад вихідного коду, який ініціалізує самого робота, після чого ініціалізує та додає до робота камеру, як окремий модуль, також визначає, що робот буде робити 10 фото з камери за секунду, та зберігати їх. Проаналізувавши цей код, можна з упевненістю сказати, що програмісту дійсно потрібно думати тільки про основну задачу та не перейматися тим, як саме підключається той чи інший модуль.

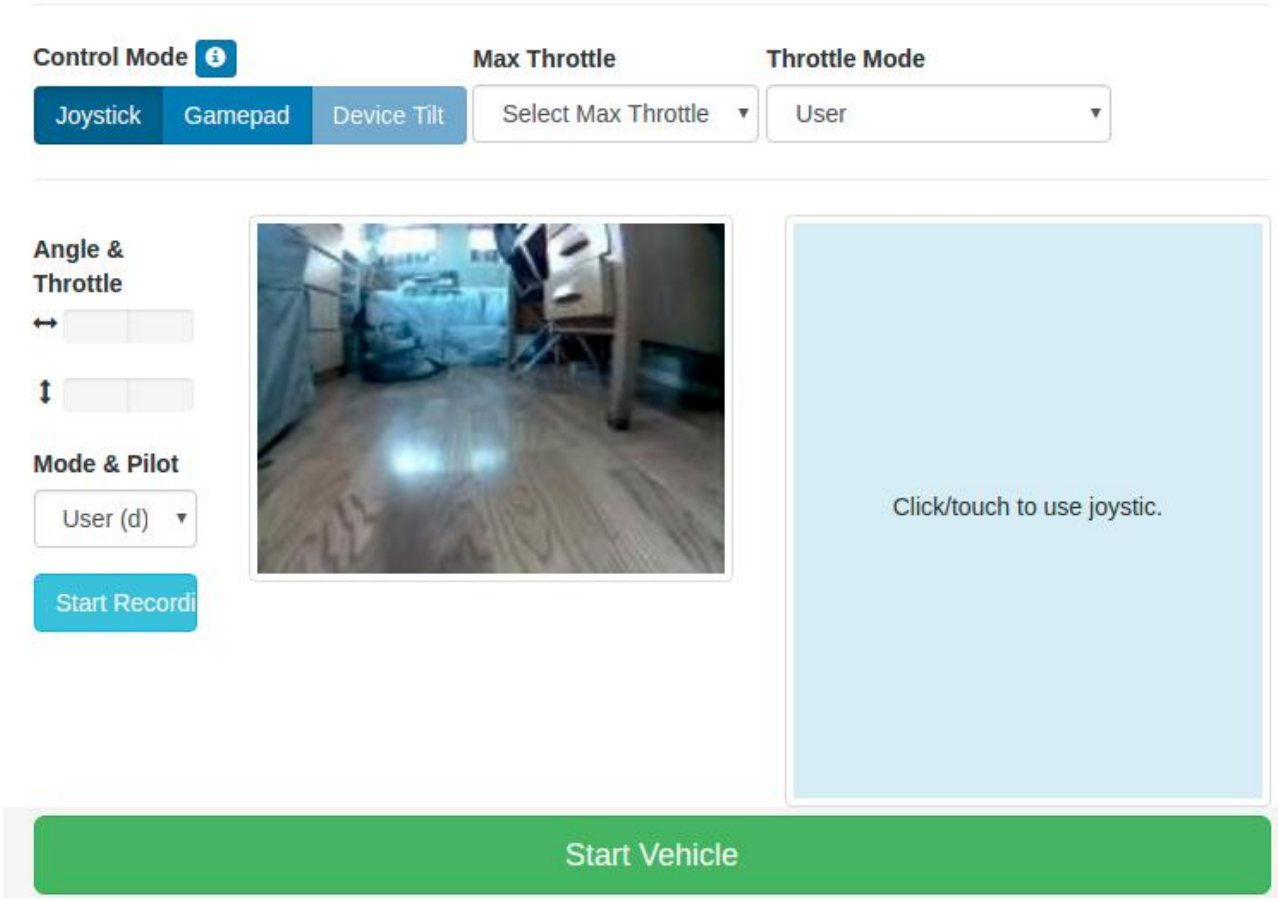

Рисунок 2.4 - Веб інтерфейс бібліотеки

Також в бібліотеці присутній зручний та зрозумілий веб інтерфейс (рис. 2.4). Після приєднання до нього, вам стануть доступні наступні дії:

- Запис. Необхідно натиснути Start Recording (розпочати запис), щоб почати записувати зображення, отримувати дані про кут повороту та поточну швидкість робота;

- Режим автоматичної швидкості. Надає можливість встановити постійну швидкість. Це використовується в гонках, якщо у вас є модель, яка контролює тільки кут повороту коліс, але не контролює швидкість;
- Режим пілота. Вмикає ручне керування роботом;
- Максимальна швидкість. Вмикає максимальну швидкість, яку може розвинути мобільний автомобіль.

Виконувати ці всі команди можна навіть з мобільного телефону. Для цього не потрібно встановлювати якісь сторонні додатки. Все що знадобиться – це тільки браузер.

Як альтернативу веб інтерфейсу, для керування роботом можна використовувати ігровий контролер, який спілкується з роботом за допомогою Bluetooth сигналу. Багатьом людям легше керувати мобільним роботом за допомогою ігрового контролера. Існує кілька частин, які забезпечують цю опцію.

Веб-контролер (який ввімкнуто за замовчуванням) може бути замінений додаванням однієї лінії коду. Після цього роботом можна буде керувати за допомогою джойстика. Бібліотека підтримує велику кількість джойстиків, проте на практиці поведінка може змінюватися залежно від моделі джойстика. Код за замовчуванням був написаний та протестований із контролером Sony PS3 Sixaxis. Інші контролери можуть працювати, але знадобляться альтернативні шляхи встановлення Bluetooth та налаштування програмного забезпечення для правильної осі та кнопок.

2.4 Програмна архітектура

Архітектура програмного забезпечення являє собою ранній набір вирішень задачі або ж системи задач. Вона представляє собою абстракцію цілої системи в цілому, враховуючи навіть найменші програмні модулі та способи їх взаємодії між собою. При розробці програмної архітектури слід звертати увагу на наступні пункти:

- Підтримка. Програма розділена на окремі програмні модулі, які можна легко підтримувати, повторно використовувати, аналізувати, тестувати та змінювати;
- Функціональна стабільність. Програма повинна стабільно виконувати свою функцію. Якщо в одному з її модулів сталась помилка, система повинна вміти правильно її обробити та продовжити свою роботу;
- Ефективність. Програма повинна виконувати поставлену перед нею задачу чи систему задач за розумний проміжок часу, при цьому використовуючи мінімальну кількість ресурсів;
- Зручність використання. Програма повинна володіти зручним та легким інтерфейсом, захистом від вводу некоректних даних з сторони користувача.

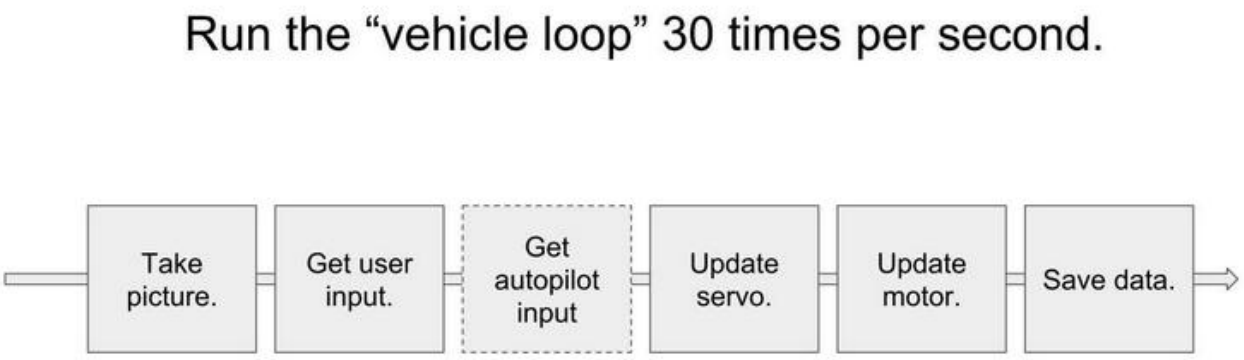

Рисунок 2.5 - Основні модулі програмного забезпечення

В дипломному проекті була розроблена модульна програмна архітектура, де кожен програмний модуль виконує певну задачу або її частину, що дозволяє забезпечити всі вище перелічені пункти. Усіма модулями керує головний клас Vehicle. Ми можемо додавати, видаляти змінювати будь який модуль окремо, після чого тестувати його просто додавши до основного класу. Я виділив 6 основних модулів (рис. 2.5).

Основний цикл програми, який реалізовано в класі Vehicle, має виконуватися 30 разів на секунду та опитувати або керувати окремими програмними модулями. Нижче наведені дії, які будуть виконуватися кожного разу при проходженні основного циклу:

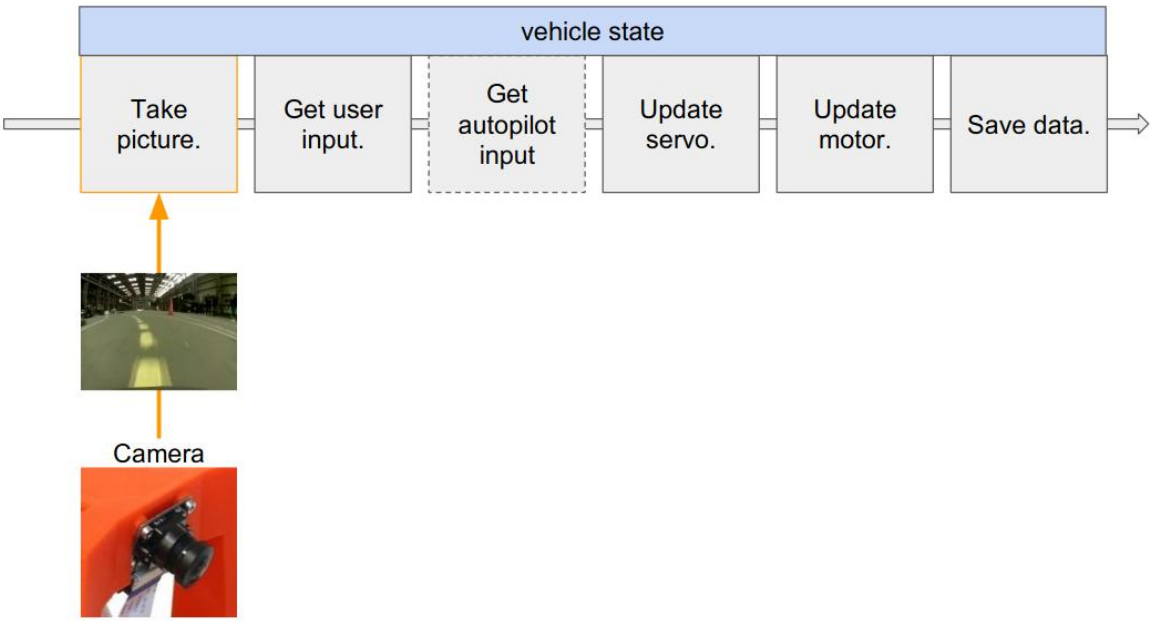

Рисунок 2.6 - Отримання картинки з камери

- Необхідно отримати зображення з камери, обрізати його до певних розмірів (в нашому випадку – це 120 x 160 пікселів) та нормалізувати. Також можливий збір інформації з інших сенсорів якщо такі присутні та підключені (рис. 2.6);

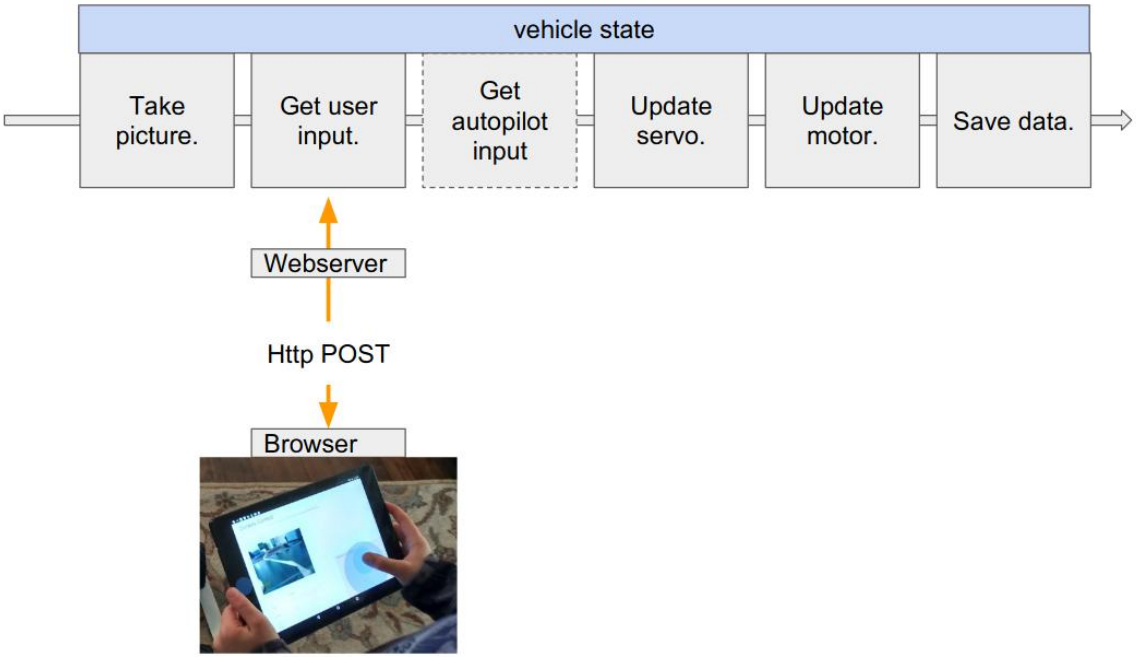

Рисунок 2.7 - Отримання команд від користувача

– Наступним кроком необхідно показати користувачу зображення, яке бачить робот. Це можна зробити за допомогою веб інтерфейсу, в якому користувач зможе в реальному часі бачити, те що бачить робот. А також, якщо робот працює в режимі ручного керування, необхідно зчитати команди, які були відправлені користувачем (рис. 2.7):

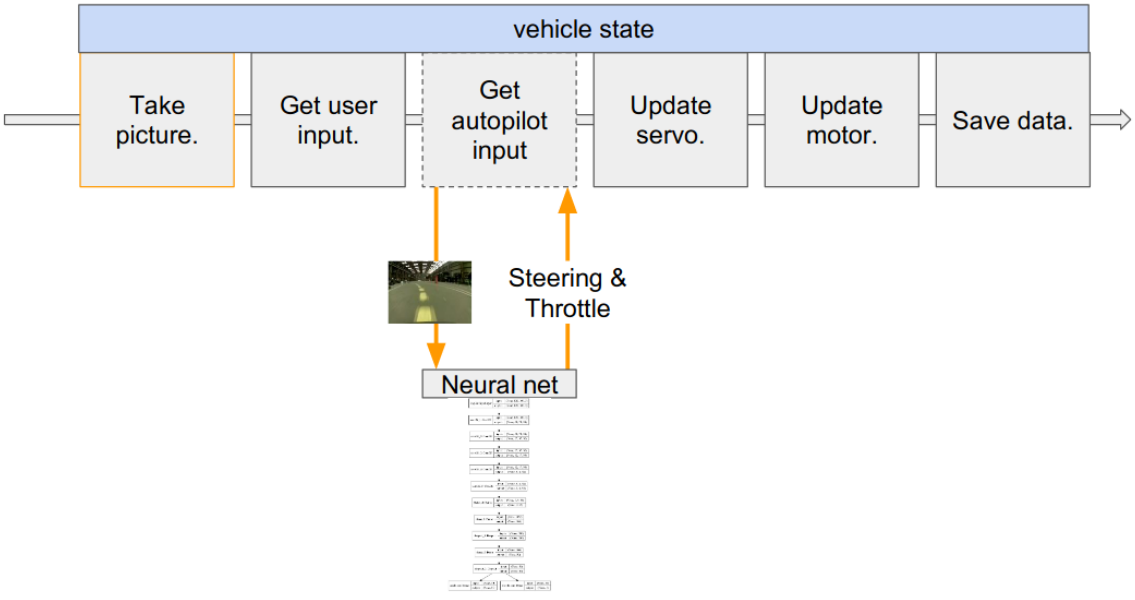

Рисунок 2.8 - Обробка зображення нейронною мережею

– Якщо робот працює в режимі автопілота, отримане зображення необхідно обробити за допомогою нейронної мережі, після чого потрібно отримати вихідні значення (рис. 2.8);

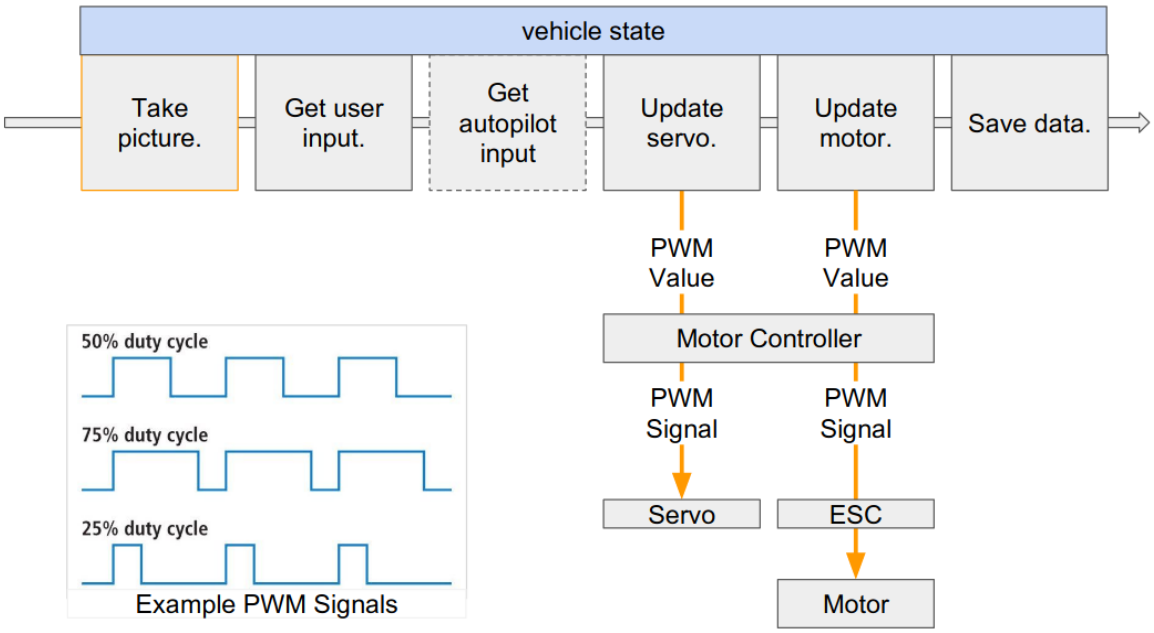

Рисунок 2.9 - Подача сигналів на органи управління роботом

– Після отримання керуючих команд від користувача, або ж від моделі нейронної мережі, потрібно обробити ці команди та в правильній формі подати на відповідні органи управління роботом. Генерується широтно-імпульсна модуляція (PWM), яка й буде керувати сервоприводом та ходовим мотором в залежності від тривалості імпульсів (рис. 2.9);

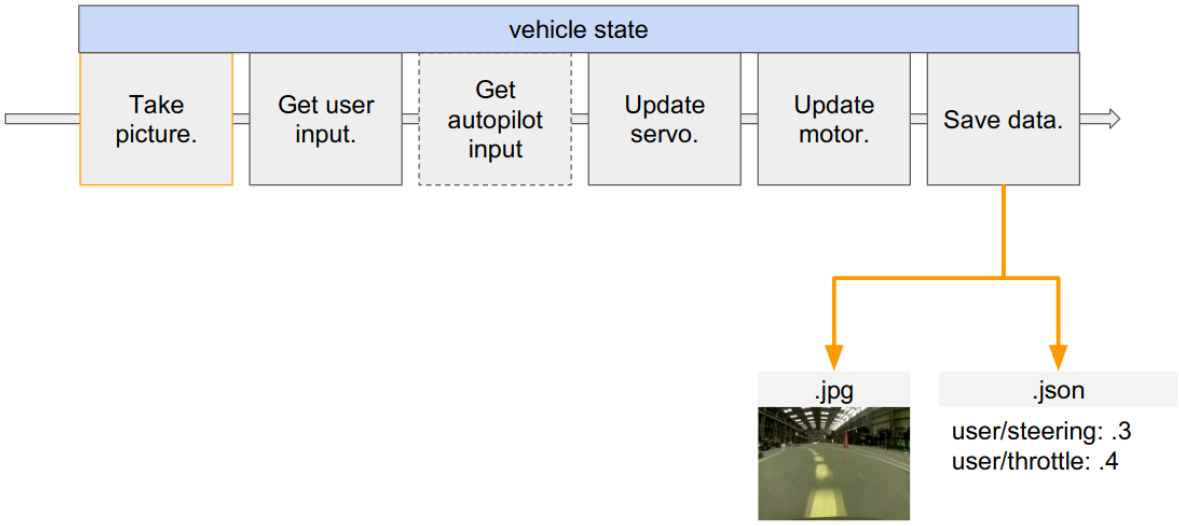

Рисунок 2.10 - Збереження отриманого зображення та даних

– Останнім кроком буде збереження отриманого з камери зображення в форматі .jpg та створення відповідного запису в форматі JSON (JavaScript Object Notation). В якому буде міститись інформація про те, в яку сторону повертав робот, з якою швидкістю він їхав в даний момент часу та шлях до зображення, на основі якого автопілот або ж користувач прийняв таке рішення (рис. 2.10).

2.5 Архітектура нейронної мережі

Оскільки основним та найбільш інформативним джерелом інформації, яку отримує робот про навколишнє середовище, є камера, нейронна мережа повинна вміти розпізнавати зображення, знаходити на них ключові елементи та на їх основі приймати рішення, від яких буде залежати алгоритм дій в конкретній ситуації [19].

Keras / Tensorflow Autopilots

```
img_in = Input(shape=(120, 160, 3), name='img_in')
x = img_in
x = Convolution2D(24, (5,5), strides=(2,2), activation='relu')(x)
x = Convolution2D(32, (5,5), strides=(2,2), activation='relu')(x)
x = Convolution2D(64, (5,5), strides=(2,2), activation='relu')(x)
x = Convolution2D(64, (3,3), strides=(2,2), activation='relu')(x)
x = Convolution2D(64, (3,3), strides=(1,1), activation='relu')(x)

x = Flatten(name='flattened')(x)

x = Dense(100, activation='relu')(x)
x = Dropout(.1)(x)
x = Dense(50, activation='relu')(x)
x = Dropout(.1)(x)

#categorical output of the angle
angle_out = Dense(15, activation='softmax', name='angle_out')(x)

#continous output of throttle
throttle_out = Dense(1, activation='relu', name='throttle_out')(x)
```

Рисунок 2.11 - Архітектура нейронної мережі.

Архітектура нейронної мережі виглядає наступним чином (рис. 2.11):

- 5 шарів згорткової нейронної мережі (CNN або ConvNet). В машинному навчанні даний тип мереж активно застосовують для аналізу зображень, відео, класифікації зображень, а також для обробки мовлення (Natural Language Processing);
- Flatten (вирівнювання) передбачає перетворення всієї об'єднаної матриці мап зображення в один стовпчик, який потім подається в нейронну мережу для обробки;
- Наступним кроком є використання вектора, який ми отримали вище, як вхід для нейронної мережі, використовуючи функцію Dense в бібліотеці Keras [22];
- Dropout реалізується шляхом випадкового вибору вузлів для викидання з заданою ймовірністю (наприклад, 20%) кожного циклу оновлення ваги.

Dropout використовується лише під час навчання моделі та не використовується при прогнозуванні.

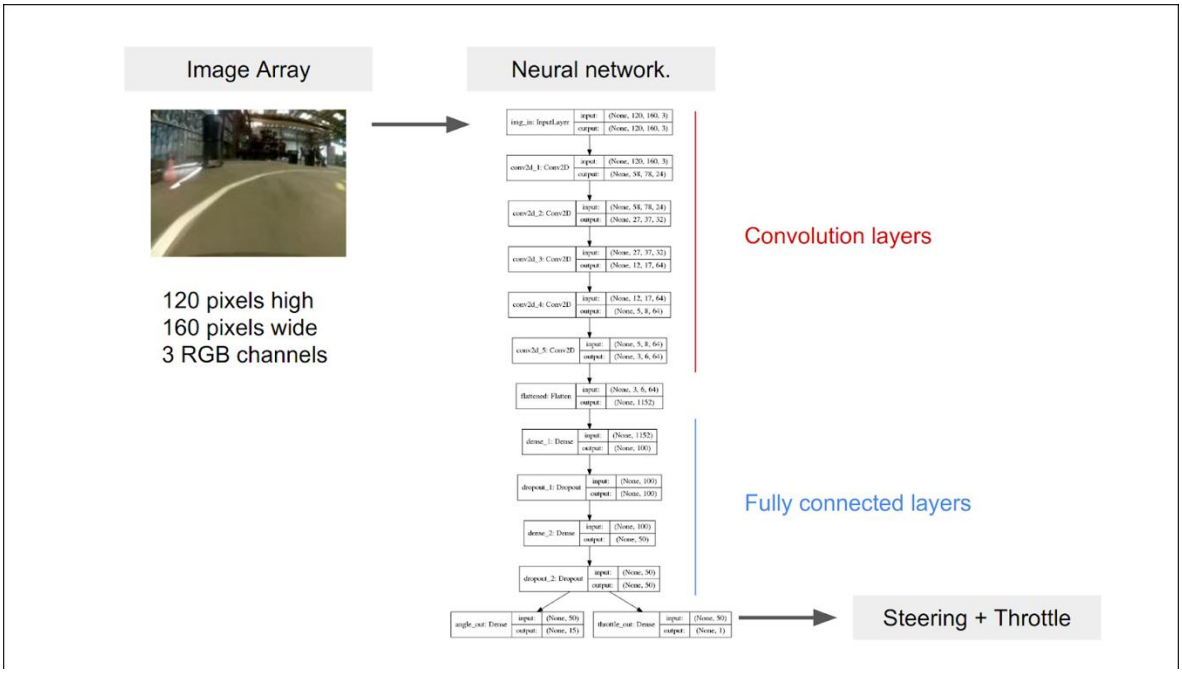

Рисунок 2.12 - Схематичний вигляд нейронної мережі.

Схематичний вигляд нейронної мережі та те, як зображення крок за кроком проходить по ній, зображено на рисунку 2.12.

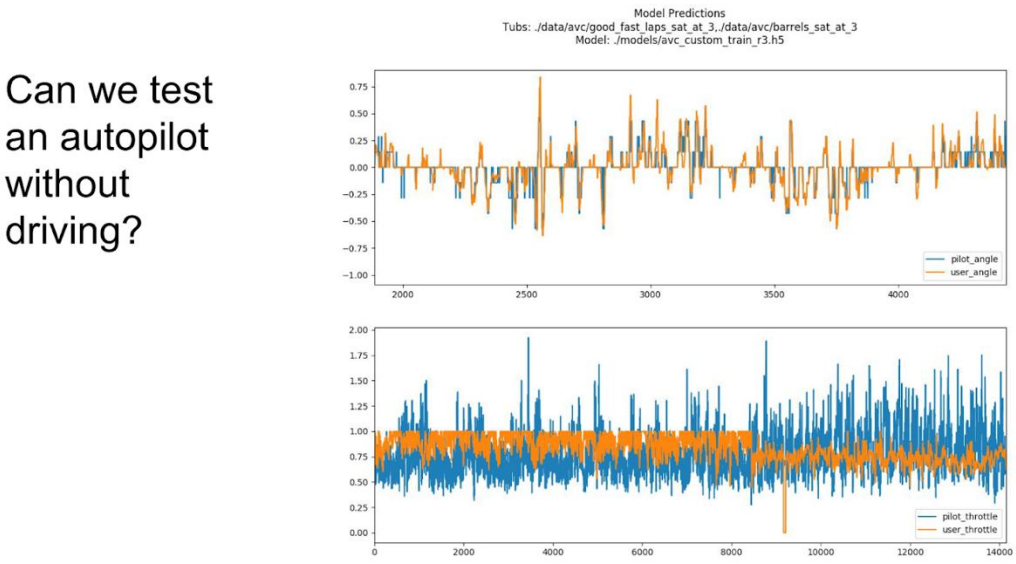

Рисунок 2.13 - Ефективності нейронної мережі .

На рисунку 2.13 зображено графік, який допоможе оцінити ефективність тренованої моделі. Жовта лінія показує значення, яке вибрав користувач під час тренувальної їзди, а синьою лінією показано значення, яке вибрала натренована модель нейронної мережі. Можна помітити, що в більшості випадків значення збігаються, а в деяких взагалі ідентичні. Це говорить нам про те, що мережа натренована достатньо та її можна перевіряти на трасі [12].

2.6 Архітектура апаратної частини

Фінальним етапом буде підключення апаратної частини.

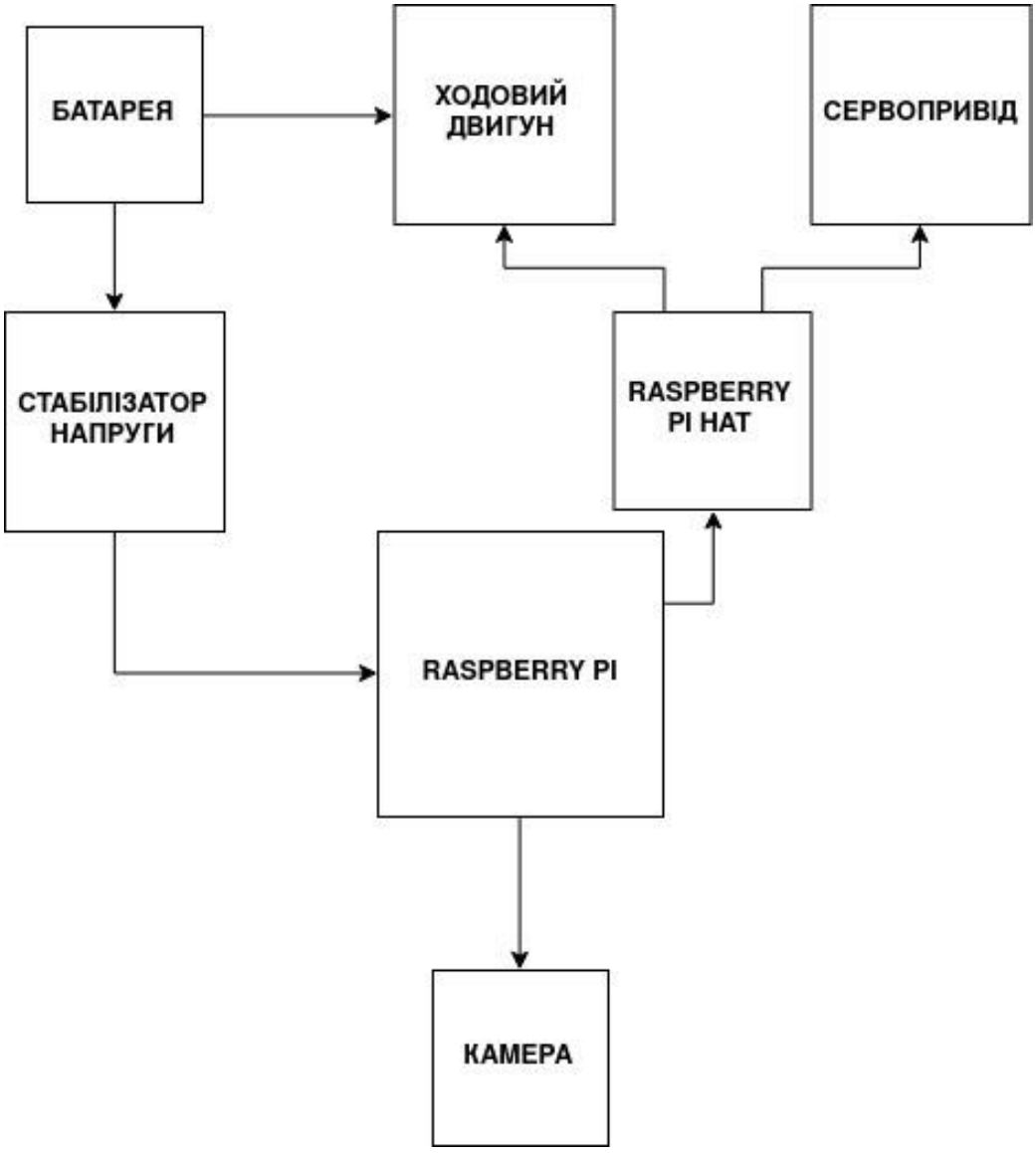

Рисунок 2.14 - Схема підключення апаратної частини.

Зробити це необхідно так, як показано на рисунку 2.14. Елемент живлення (батарея) нам необхідно підключити до стабілізатора напруги та до ходового мотора одночасно. Стабілізатор напруги налаштувати на вихідну напругу в 5В (щоб безпечну напругу для Raspberry Pi), після чого підключити до головного комп'ютера (Raspberry Pi). До Raspberry підключаємо камеру за допомогою спеціального роз'єму. Після чого необхідно приєднати Raspberry Pi Hat, за допомогою якого буде здійснюватися керування ходовим мотором та сервоприводом [15].

3 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Код проекту

Структура проекту виглядає наступним чином (рис. 3.1).

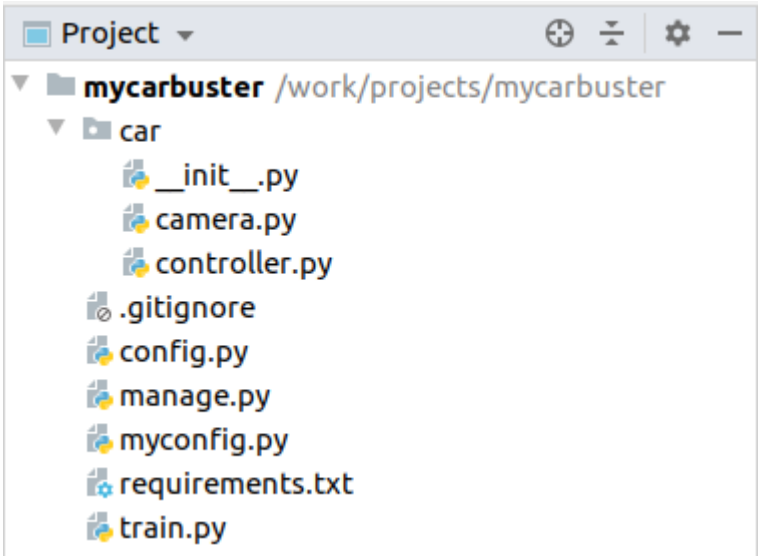

Рисунок 3.1 - Структура проекту

Через скрипт manage.py можна запускати автономного робота в різних режимах роботи, а також тренувати моделі нейронних мереж на основі зібраних роботом даних. Нижче наведено коротку допомогу по використанню скрипта. Отож, в скрипта є два режими роботи. Розглянемо їх детальніше:

- Режим drive. Використовується для запуску робота в режимі їзди. Також можна додати наступні додаткові опції:
  - --model – для вибору моделі яку буде використовувати робот;
  - --js – активує режим їзди з контроллером;
  - --type –дозволяє вибрати тип нейронної мережі;
  - --camera – вибір типу камери;
  - --meta – мета дані.
- Режим train. Використовується для тренування моделі нейронної мережі на основі зібраних даних:

- --tub – шлях до директорії із зібраними даними;
- --file – шлях до файлу, в якому записані шляхи до директорій з даними для тренування;
- --type – ця опція дозволяє вибрати тип нейронної мережі;
- --continuous – застосовується для вдосконалення вже нетренованої моделі.

```
"""
Scripts to drive a donkey 2 car

Usage:
    manage.py (drive) [--model=<model>] [--js]
        [--type=(linear|categorical|rnn|imu|behavior|3d|localizer|latent)]
        [--camera=(single|stereo)] [--meta=<key:value> ...]
    manage.py (train) [--tub=<tub1,tub2,..tubn>] [--file=<file> ...]
        (--model=<model>) [--transfer=<model>]
        [--type=(linear|categorical|rnn|imu|behavior|3d|localizer)]
        [--continuous]


Options:
    -h --help          Show this screen.
    --js               Use physical joystick.
    -f --file=<file>   A text file containing paths to tub files, one per line.
                       Option may be used more than once.
    --meta=<key:value> Key/Value strings describing describing a piece of meta
                       data about this drive. Option may be used more than
                       once.
"""
```

Рисунок 3.2 - Коротка допомога по використанню скрипта

Також можна запустити скрипт з додатковим параметром --help або -h, щоб отримати допомогу (рис. 3.2).

```
absl-py==0.8.1
appdirs==1.4.4
astor==0.8.0
astunparse==1.6.3
attrs==19.3.0
black==19.10b0
cachetools==4.1.0
certifi==2019.9.11
chardet==3.0.4
Click==7.0
decorator==4.4.0
dnspython==1.16.0
docopt==0.6.2
donkeycar==2.5.8
eventlet==0.25.2
Flask==1.1.2
gast==0.3.3
google-auth==1.14.3
google-auth-oauthlib==0.4.1
google-pasta==0.2.0
greenlet==0.4.15
grpcio==1.29.0
h5py==2.10.0
idna==2.8
imageio==2.8.0
imageio-ffmpeg==0.4.2
itsdangerous==1.1.0
Jinja2==2.11.2
Keras-Preprocessing==1.1.2
Markdown==3.2.2
MarkupSafe==1.1.1
monotonic==1.5
moviepy==1.0.3
moviepy==1.0.3
numpy==1.17.2
oauthlib==3.1.0
opt-einsum==3.2.1
pandas==1.0.3
pathspec==0.8.0
Pillow==7.1.2
proglog==0.1.9
protobuf==3.12.0
pyasn1==0.4.8
pyasn1-modules==0.2.8
python-dateutil==2.8.1
python-engineio==3.12.1
python-socketio==4.5.1
pytz==2019.3
regex==2020.5.14
requests==2.22.0
requests-oauthlib==1.3.0
rsa==4.0
scipy==1.4.1
six==1.12.0
tensorboard==2.2.1
tensorboard-plugin-wit==1.6.0.post3
tensorflow==2.2.0
tensorflow-estimator==2.2.0
termcolor==1.1.0
toml==0.10.1
tornado==4.5.3
tqdm==4.46.0
typed-ast==1.4.1
urllib3==1.25.9
Werkzeug==1.0.1
wrapt==1.12.1
```

Рисунок. 3.3 - Список необхідних бібліотек

Також для роботи програми нам необхідно встановити додаткові бібліотеки, список яких перелічений у файлі requirements.txt та наведений нижче (рис. 3.3)

Не менш важливою частиною проекту є файл з конфігураціями (config.py), в якому перелічені всі налаштування, з якими буде працювати автономний мобільний робот. Оскільки є велика кількість різного роду конфігурацій, буде перелічено тільки найважливіші з них:

```
22  # ================================VEHICLE=====================================
23  # the vehicle loop will pause if faster than this speed.
24  DRIVE_LOOP_HZ = 20
25
26  # the vehicle loop can abort after this many iterations, when given a
27  # positive integer.
28  MAX_LOOPS = None
29
30  # ================================CAMERA======================================
31  # (PICAM|WEBCAM|CVCAM|CSIC|V4L|MOCK)
32  CAMERA_TYPE = "WEBCAM"
33  IMAGE_W = 160
34  IMAGE_H = 120
35
36  # default RGB=3, make 1 for mono
37  IMAGE_DEPTH = 3
38  CAMERA_FRAMERATE = DRIVE_LOOP_HZ
```

Рисунок 3.4 - Файл конфігурації робота

– Конфігурація частоти оновлення (рис. 3.4) основного циклу, який виконує управління роботом. Також на рисунку присутні конфігурація камери, а саме її тип, ширина, висота та глибина картинки і, звісно, частота, з якою камера буде робити нові знімки та зберігати їх.

```
# ============================STEERING=====================================
# channel on the 9685 pwm board 0-15
STEERING_CHANNEL = 1

# pwm value for full left steering
STEERING_LEFT_PWM = 500

# pwm value for full right steering
STEERING_RIGHT_PWM = 290

# ============================THROTTLE=====================================
THROTTLE_CHANNEL = 0  # channel on the 9685 pwm board 0-15
THROTTLE_FORWARD_PWM = 700  # pwm value for max forward throttle
THROTTLE_STOPPED_PWM = 370  # pwm value for no movement
THROTTLE_REVERSE_PWM = 220  # pwm value for max reverse throttle
```

Рисунок 3.5- Файл конфігурації робота.

– Нижче (рис. 3.5) показана конфігурація положення керуючих коліс, максимального кута їх повороту (як в ліву так і в праву сторону), також максимальної швидкості робота (як при їзді вперед так і при їзді назад), а також нульова швидкість (при якій робот не рухається).

```
# (linear|categorical|rnn|imu|behavior|3d|localizer|latent)
DEFAULT_MODEL_TYPE = "linear"

# how many records to use when doing one pass of gradient decent. Use a
# smaller number if your gpu is running out of memory.
BATCH_SIZE = 128

# what percent of records to use for training. the remaining used for
# validation.
TRAIN_TEST_SPLIT = 0.8

# how many times to visit all records of your data
MAX_EPOCHS = 100

```

Рисунок 3.6 - Файл конфігурації робота

– Також в цьому файлі ми можемо задати конфігурації (рис. 3.6) відносно типу моделі, скільки записів використовувати при виконанні одного проходу градієнтного спуску, який відсоток записів використовувати для

тренувань (решта використовується для перевірки) та скільки епох тренувати модель.

```
class CameraConfiguration:
    def __init__(self, vehicle: Vehicle, cam_type: str = "single"):
        self._vehicle = vehicle
        self._cam_type = cam_type

    def configure_camera(self):
        print(f"[*] Camera type = {cfg.CAMERA_TYPE}")
        self._configure_single_camera()

    def _configure_single_camera(self):
        if cfg.CAMERA_TYPE == "PICAM":
            cam = PiCamera(
                image_w=cfg.IMAGE_W, image_h=cfg.IMAGE_H, image_d=cfg.IMAGE_DEPTH
            )
        elif cfg.CAMERA_TYPE == "WEBCAM":
            cam = Webcam(
                image_w=cfg.IMAGE_W, image_h=cfg.IMAGE_H, image_d=cfg.IMAGE_DEPTH
            )
        elif cfg.CAMERA_TYPE == "CVCAM":
            cam = dk_image.CvCam(
                image_w=cfg.IMAGE_W, image_h=cfg.IMAGE_H, image_d=cfg.IMAGE_DEPTH
            )
        elif cfg.CAMERA_TYPE == "MOCK":
            cam = MockCamera(
                image_w=cfg.IMAGE_W, image_h=cfg.IMAGE_H, image_d=cfg.IMAGE_DEPTH
            )
        else:
            raise Exception(f"Unknown camera type: {cfg.CAMERA_TYPE}")

        self._vehicle.add(cam, inputs=[], outputs=["cam/image_array"], threaded=True)
```

Рисунок 3.7 - Клас CameraConfiguration

Вище наведені тільки основні конфігурації, на які слід звернути увагу при створенні автономного мобільного робота.

В модулі car знаходяться два файли camera.py та controller.py. У файл camera.py імплементовано клас CameraConfiguration (рис. 3.7) основна задача якого – це визначення типу камер (тип камери вказаний в конфігураційному файлі) та в залежності від типу камери її конфігурація, правильне підключення до робота з усіма необхідними параметрами. У випадку, якщо тип камери не вказаний або ж несумісний, програма видасть помилку.

```python
class ControllerConfiguration:
    def __init__(self, vehicle: Vehicle):
        self._vehicle = vehicle
        self._controller = None

    def configure(self, use_joystick: bool):
        if use_joystick or cfg.USE_JOYSTICK_AS_DEFAULT:
            # modify max_throttle closer to 1.0 to have more power modify
            # steering_scale lower than 1.0 to have less responsive steering
            self._controller = controller.get_js_controller(cfg)

            if cfg.USE_NETWORKED_JS:
                network_js = controller.JoyStickSub(cfg.NETWORK_JS_SERVER_IP)
                self._vehicle.add(network_js, threaded=True)
                self._controller.js = network_js

        else:
            # This web controller will create a web server that is capable
            # of managing steering, throttle, and modes, and more.
            self._controller = LocalWebController()

        self._vehicle.add(
            self._controller,
            inputs=["cam/image_array"],
            outputs=["user/angle", "user/throttle", "user/mode", "recording"],
            threaded=True,
        )

    @property
    def controller(self):
        return self._controller

```

Рисунок 3.8 - Клас ControllerConfig

Відповідно у файлі controller.py імплементовано клас ControllerConfig (рис. 3.8), задача якого визначити, чи робота запущено в режимі керування фізичним контролером, та якщо це так, тоді правильно сконфігурувати і додати його як об’єкт до робота. Функція configure перевіряє, чи було передано use_joystick або чи було сконфігуровано використовування джойстика за замовчуванням у конфігураційному файлі. І якщо це так, вона конфігурує контролер з відповідними параметрами, після чого додає його до самого автомобіля, а також зберігає сам об’єкт контроллера, який ми можемо отримати за допомогою методу controller.

```python
def drive(
    cfg,
    model_path: str = None,
    use_joystick=False,
    model_type=None,
    camera_type="single",
    meta="linear",
):
    """
    Construct a working robotic vehicle from many parts. Each part runs as a
    job in the Vehicle loop, calling either it's run or run_threaded method
    depending on the constructor flag `threaded`. All parts are updated one
    after another at the framerate given in cfg.DRIVE_LOOP_HZ assuming each
    part finishes processing in a timely manner. Parts may have named
    outputs and inputs. The framework handles passing named outputs to parts
    requesting the same named input.
    """
    print("[*] Started configure vehicle...")

    # Initialize car
    vehicle = dk.vehicle.Vehicle()

    print("[*] Setup camera...")
    CameraConfiguration(vehicle, camera_type).configure_camera()
    print("[*] Camera is ready...")

    print("[*] Setup joystick...")
    ControllerConfiguration(vehicle).configure(use_joystick)
    print("[*] Joystick is ready...")

```

Рисунок 3.9 - Створення об’єкту робота та додавання камери та контроллера.

Основна функція (drive), що реалізована в скрипті manage.py налаштовує та конфігурує всі необхідні модулі для робота. Для початку необхідно створити об’єкт класу робота (Vehicle) та додати до неї камеру та контроллер (рис. 3.9).

```python
# this throttle filter will allow one tap back for esc reverse
th_filter = ThrottleFilter()
vehicle.add(th_filter, inputs=["user/throttle"], outputs=["user/throttle"])

# See if we should even run the pilot module.
# This is only needed because the part run_condition only accepts boolean
class PilotCondition:
    def run(self, mode):
        return mode != "user"

vehicle.add(PilotCondition(), inputs=["user/mode"], outputs=["run_pilot"])

class RecordTracker:
    def __init__(self):
        self.last_num_rec_print = 0
        self.force_alert = 0

    def run(self, num_records):
        if num_records is None:
            return 0

        if self.last_num_rec_print != num_records or self.force_alert:
            self.last_num_rec_print = num_records

            if num_records % 10 == 0:
                print(f"[I] Recorded {num_records} records")

        return 0

rec_tracker_part = RecordTracker()
vehicle.add(rec_tracker_part, inputs=["tub/num_records"], outputs=["records/alert"])
```

Рисунок 3.10 - Створення та додавання ThrottleFilter та PilotCondition

Коли камера та контролер готові, наступним кроком буде налаштування та додавання (рис. 3.10) до робота об’єктів ThrottleFilter (необхідний для визначення напрямку руху робота. Вперед або назад) та PilotCondition (додає умову яка визначає хто керує роботом).

```
class RecordTracker:
    def __init__(self):
        self.last_num_rec_print = 0
        self.force_alert = 0

    def run(self, num_records):
        if num_records is None:
            return 0

        if self.last_num_rec_print != num_records or self.force_alert:
            self.last_num_rec_print = num_records

            if num_records % 10 == 0:
                print(f"[I] Recorded {num_records} records")

        return 0

rec_tracker_part = RecordTracker()
vehicle.add(rec_tracker_part, inputs=["tub/num_records"], outputs=["records/alert"])

if cfg.AUTO_RECORD_ON_THROTTLE and isinstance(ctr, JoystickController):
    # then we are not using the circle button. hijack that to force a
    # record count indication
    def show_record_amount_status():
        rec_tracker_part.last_num_rec_print = 0
        rec_tracker_part.force_alert = 1

    # Todo: move to controller
    ctr.set_button_down_trigger("circle", show_record_amount_status)
```

Рисунок 3.11 - Створення та додавання RecordTracker

Наступним кроком ми додаємо до робота об’єкт RecordTracker (рис. 3.11), який буде виводити повідомлення в термінал, коли кількість збережених записів буде кратна 10, що в свою чергу буде інформувати користувача про кількість збережених записів під час навчальної їзди. Після того, як об’єкт додано, необхідно перевірити, чи увімкнена опція AUTO_RECOED_ON_THROTTLE та чи присутній джойстик. Якщо це так, ми вмикаємо логування кількості зроблених записів, а також реєструємо функцію, яка буде виводити кількість зроблених записів на даний момент при натисканні на коло (кнопка на джойстику).

```
class ImgPreProcess:
    """
    preprocess camera image for inference.
    normalize and crop if needed.
    """

    def __init__(self, cfg):
        self.cfg = cfg

    def run(self, img_arr):
        return normalize_and_crop(img_arr, self.cfg)

inf_input = "cam/normalized/cropped"
vehicle.add(
    ImgPreProcess(cfg),
    inputs=["cam/image_array"],
    outputs=[inf_input],
    run_condition="run_pilot",
)
inputs = [inf_input]
```

Рисунок 3.12 - Створення та додавання ImagePreProcessor

Додавання об’єкту ImagePreProcess (рис. 3.12), який відповідає за обробку зображень до того, як вони будуть збережені, необхідний для нормалізації та обрізання картинки відповідно до конфігурацій, які були задані користувачем. Нормалізація зображення – це типовий процес обробки зображень, який змінює діапазон значень інтенсивності пікселів. Зазвичай його призначення – перетворити вхідне зображення в діапазон значень пікселів більш відомих або нормальних для органів чуття – звідси походить термін «нормалізація». Обрізка зображення необхідна для зменшення кількості пікселів, розміру зображення, а також, це дозволить зберігати зображення з однаковою шириною та висотою, що значно полегшить подальшу їх обробку.

```
class DriveMode:
    def run(self, mode, user_angle, user_throttle, pilot_angle, pilot_throttle):
        if mode == "user":
            return user_angle, user_throttle

        elif mode == "local_angle":
            return pilot_angle, user_throttle

        else:
            return pilot_angle, pilot_throttle * cfg.AI_THROTTLE_MULT

vehicle.add(
    DriveMode(),
    inputs=[
        "user/mode",
        "user/angle",
        "user/throttle",
        "pilot/angle",
        "pilot/throttle",
    ],
    outputs=["angle", "throttle"],
)
```

Рисунок 3.13 - Створення та додавання DriveMode

Об’єкт DriveMode (рис. 3.13) необхідний для визначення, хто в даний момент керує автономним мобільним роботом.

```
    from donkeycar.parts.actuator import PCA9685, PWMSteering, PWMThrottle

    steering_controller = PCA9685(
        cfg.STEERING_CHANNEL, cfg.PCA9685_I2C_ADDR, busnum=cfg.PCA9685_I2C_BUSNUM
    )
    steering = PWMSteering(
        controller=steering_controller,
        left_pulse=cfg.STEERING_LEFT_PWM,
        right_pulse=cfg.STEERING_RIGHT_PWM,
    )

    throttle_controller = PCA9685(
        cfg.THROTTLE_CHANNEL, cfg.PCA9685_I2C_ADDR, busnum=cfg.PCA9685_I2C_BUSNUM
    )
    throttle = PWMThrottle(
        controller=throttle_controller,
        max_pulse=cfg.THROTTLE_FORWARD_PWM,
        zero_pulse=cfg.THROTTLE_STOPPED_PWM,
        min_pulse=cfg.THROTTLE_REVERSE_PWM,
    )

    vehicle.add(steering, inputs=["angle"])
    vehicle.add(throttle, inputs=["throttle"])
```

Рисунок 3.14 - Створення та додавання PWMSteering і PWMThrottle

Тепер необхідно додати два об’єкти (рис. 3.14), без яких автономний робот не зможе пересуватися в просторі.

PWMSteering – об’єкт, який відповідає за управління сервоприводом. Сервопривід – це невеликий пристрій, який включає в себе двопровідний двигун постійного струму, зубчастий привід, потенціометр, інтегральну схему і вихідний вал. За допомогою сервоприводу, робот зможе виконувати керування керуючими колесами.

PWMThrottle – об'єкт, який відповідає за управління ходовим мотором. Саме він дозволяє керувати швидкістю їзди робота та напрямком руху.

На цьому програмна частина, яка відповідає за керування роботом завершена.

3.2. Реалізація апаратної частини

Реалізація апаратної частини є не менш важливою, тому що саме від неї залежать всі технічні характеристики та можливості автономного мобільного робота.

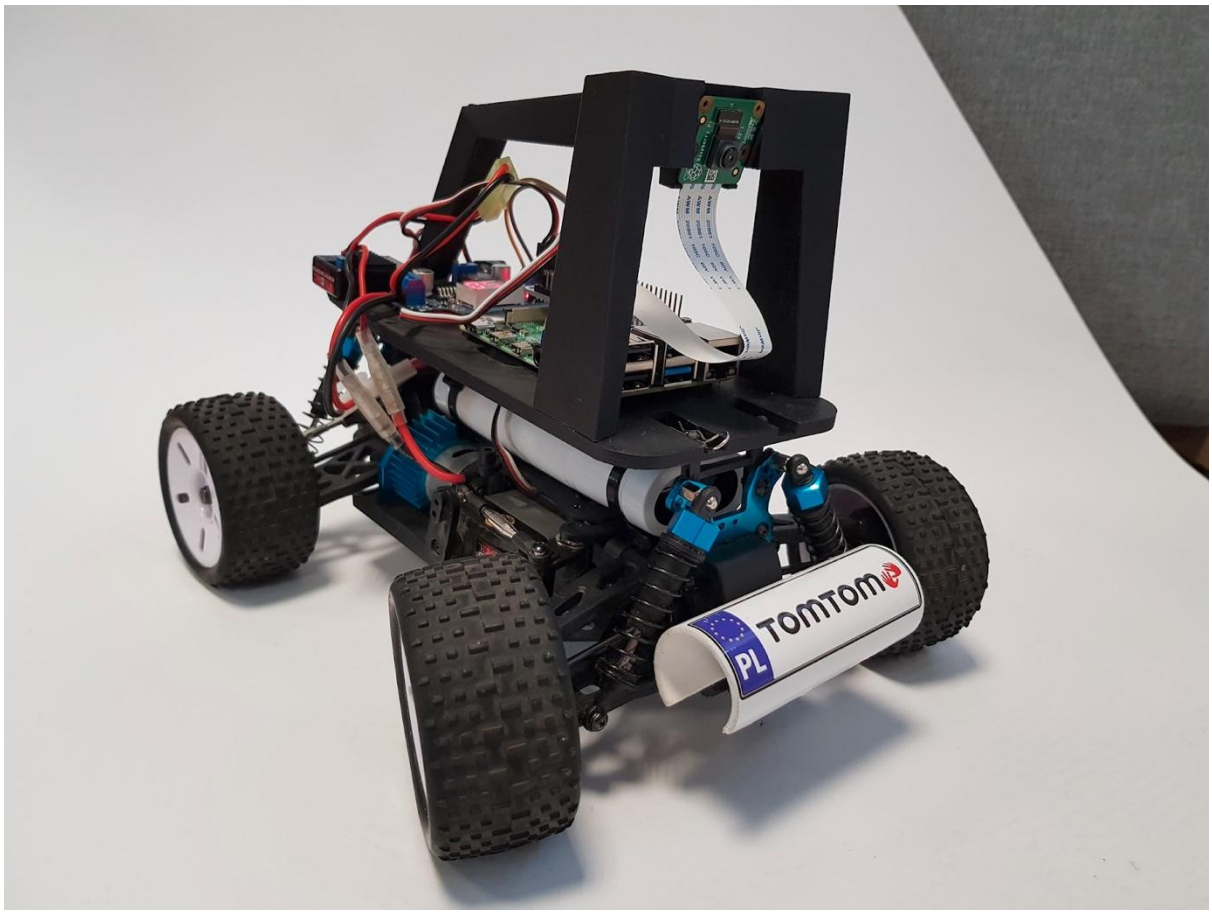

Рисунок 3.15 - База автономного робота

Базу автономного робота взяту з машинки на радіокеруванні моделі Himoto (рис. 3.15). Вона обладнана повним приводом, який забезпечує однаковий крутний момент на чотирьох колесах, що, в свою чергу, допомагає роботу краще триматися на трасі. Також база Himoto володіє наступними характеристиками:

- Масштаб: 1:18;
- Тип двигуна: Електричний;
- Максимальна швидкість: 40 км./год.;
- Вага: 600 гр.;
- Акумулятор: LiPo, ємністю: 1500 мА·год, напругою: 7.4 В;
- Габарити: 255 х 185 х 90 мм.

Рисунок 3.16 - Raspberry PI 4 model B

Raspberry PI 4 model B (рис. 3.16). Саме цю плату було обрано для виконання всіх обчислень, обробки зображень, зчитування показників з різного роду сенсорів, обробки інформації та подання відповідних сигналів на органи управління (сервопривід і мотор).

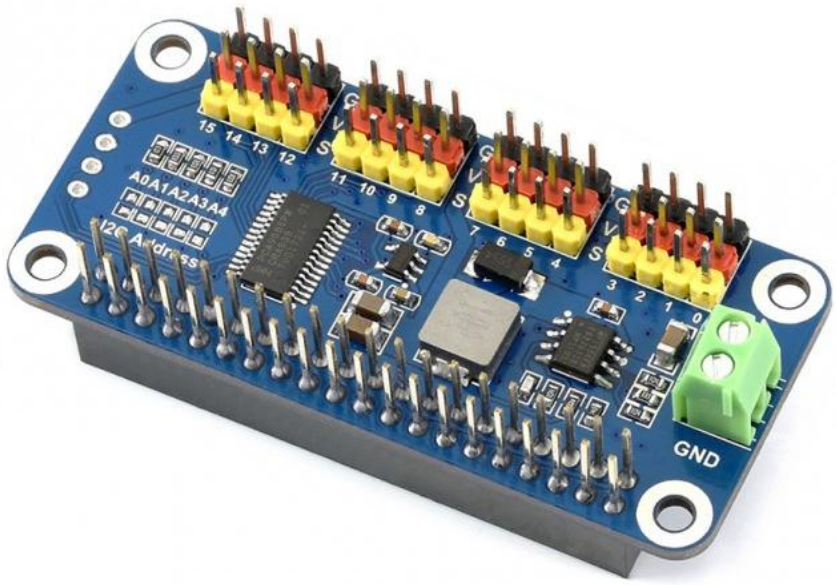

Рисунок 3.17 - Servo Driver HAT

Потрібно сказати, що Raspberry Pi є досить потужним у більшості випадків, але він не настільки хороший у забезпеченні точного виходу ШІМ (широтно імпульсної модуляції). Нам потрібен Servo Driver HAT (рис. 3.17) через обмежену кількість вихідних ШІМ каналів на Raspberry, а також через те, що Servo Driver HAT забезпечує більш чистий сигнал.

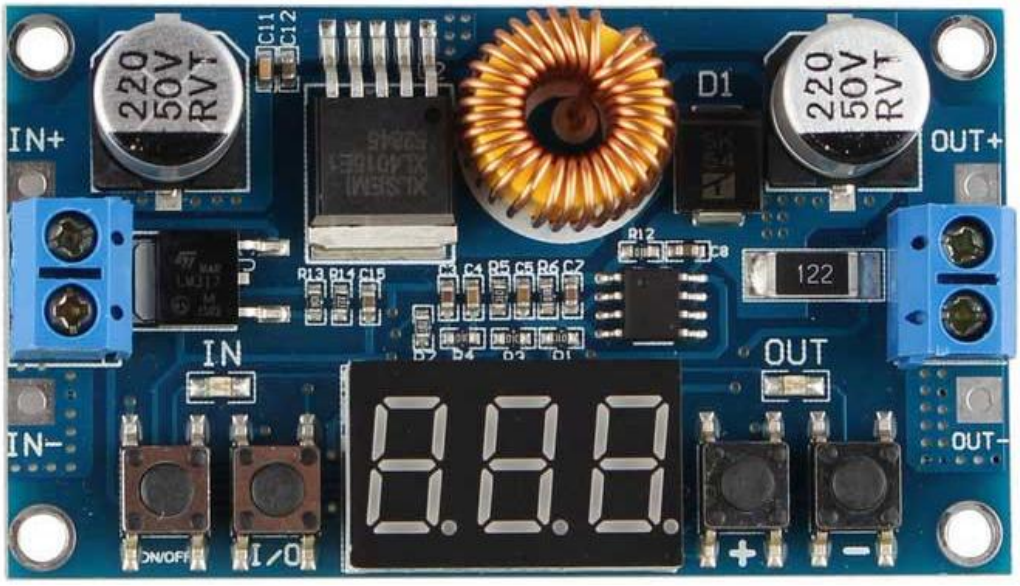

Рисунок 3.18 - Модуль перетворювача блоку живлення

Для того, щоб забезпечувати стабільне живлення для Raspberry Pi, я використав модуль перетворювача блоку живлення (рис. 3.18) через його особливості:

- Бортовий лічильник напруги може відображати значення вхідної або вихідної напруги, а два світлодіоди вказують на поточний відображення (вхід або вихід);
- Кнопка відключення. (натисніть на неї, щоб закрити вихід);
- Можливість встановити напругу виходу за замовчуванням;
- Всі задані параметри енергонезалежні;
- 5А, висока потужність, висока ефективність, низька пульсація;
- Вхідна напруга 5,0 ~ 36В;
- Регульований діапазон вихідної напруги 1,2 В ~ 32 В;
- Вихідний струм до 5А;

- Вихідна потужність до 75 Вт;
- Висока ефективність конверсії до 96%;
- Вбудований термозахист та захист від короткого замикання;
- Розмір: 66 * 39 * 18мм.

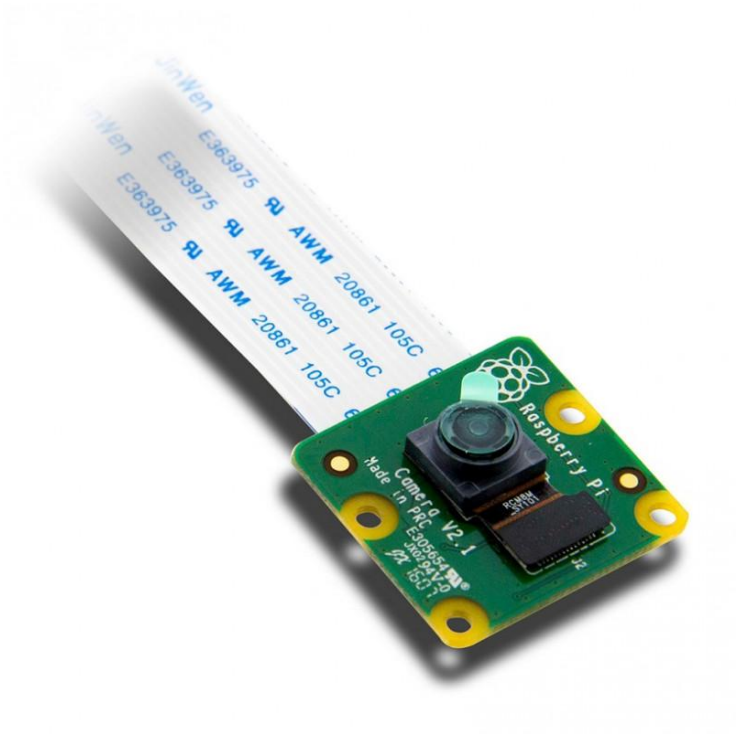

Рисунок 3.19 - Raspberry PI 8MP Camera module V2

Raspberry PI 8MP Camera module V2 (рис. 3.19) є головним модулем для сприйняття інформації про навколишній світ на роботі. Він має надзвичайно якісний 8 Мп (мегапіксельний) датчик зображення SONY IMX219 та об'єктив із фіксованим фокусом. Цей модуль камери V2 здатний до статичних зображень 3280 x 2464 пікселів, а також підтримує відео в наступних розширеннях:

- 1920 x 1080 пікселів з частотою в 30 fps;
- 1280 x 720 пікселів з частотою в 60 fps;
- 640 x 480 пікселів з частотою в 90 fps.

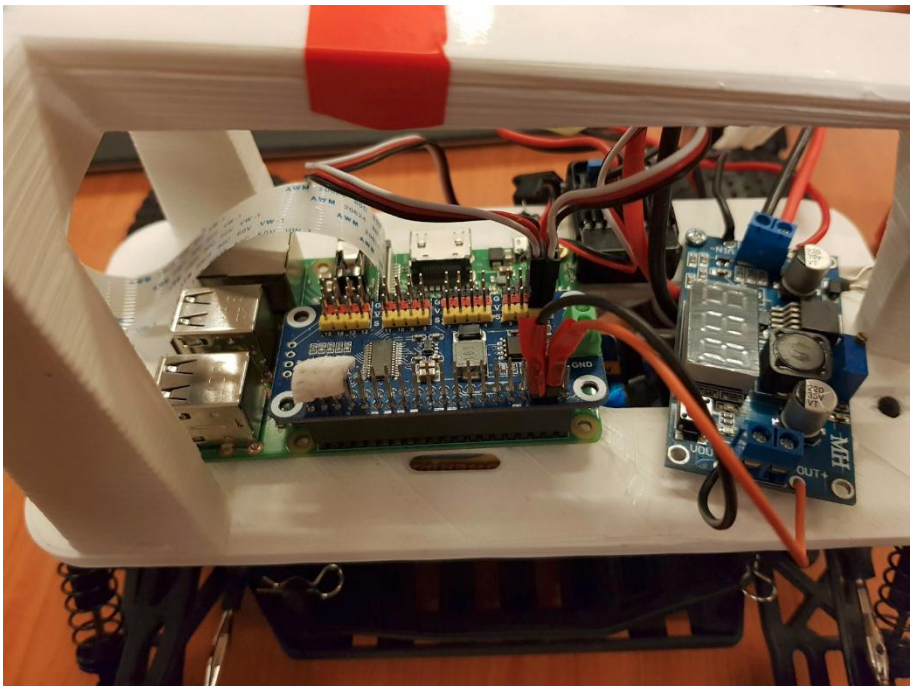

Рисунок 3.20 - З’єднання всіх складових апаратної частини

Після з'єднання всіх необхідних для роботи робота апаратних частин (рис. 3.20) та перевірки їх справності, як незалежно один від одного, так і в зібраному вигляді. Також необхідно переконатися, що кнопка аварійної зупинки робота працює справно, а всі дані зібрані автономним роботом під час навчальної їзди зберігаються коректно. Тепер час зарядити батарею та підготуватися до збору даних для тренування нейронної мережі.

3.3 Тренування нейронної мережі

Перш ніж перейти до тренування нейронної мережі, необхідно чітко розуміти, що достатньо даних для тренування та вони відфільтровані. Якщо говорити про кількість, то чим більше даних ми зможемо зібрати, тим краще робот буде орієнтуватися на трасі та зможе приймати правильні рішення в різноманітних ситуаціях.

Рисунок 3.21 - Збір даних для тренування нейронної мережі

Нас влаштовує випадок, коли робот повністю вивчить трасу і буде чітко знати, які команди на органи керування необхідно подавати в даний момент часу. 16000 записів – саме стільки вдалося зібрати під час неперервної їзди (рис. 3.21), але це ще не все. Тепер ці всі дані необхідно перевірити, а в деяких випадках скорегувати. Цей процес займає найбільше часу та вимагає уважності. Щоб спростити цей довгий процес, було розроблено скрипт (на мові Python) з GUI (графічним інтерфейсом користувача), який зчитував картинки з відповідними метаданими, які записувались в файл з форматом JSON (JavaScript Object Notation), та відображав користувачу. Після цього можна було скорегувати дані та зберегти запис або ж видалити повністю [16].

Сам процес тренування виконується за допомогою скрипта train.py. Зазвичай цей процес займає від 10 до 40 хвилин – все залежить від потужності комп'ютера, на якому тренується модель, від кількості даних для тренування та, звісно, від кількості заданих епох.

3.4 Тестування та порівняння моделей нейронної мережі

Для того, щоб вибрати найкращу модель, необхідно визначити найважливіші критерії, на які потрібно звертати увагу. Час, стабільність та акуратність – саме ці три складові були взяті за основу та саме вони визначали якість нейронної моделі.

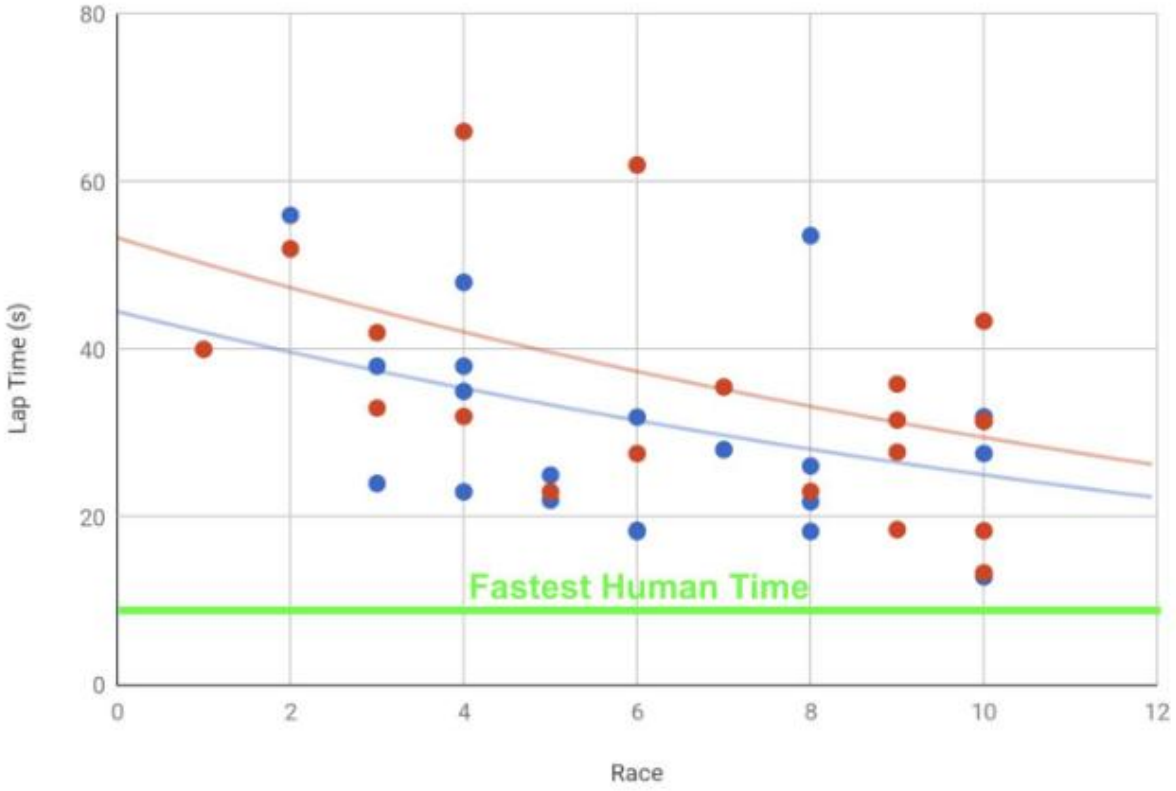

Рисунок 3.22 - Графік ефективності моделі

На рисунку 3.22 зображено графік, на якому можна побачити найкращий час, за який робот подолав коло. Зелена лінія показує найкращий час, за який робот зміг подолати коло під управлінням людини. Червоні та сині точки показують різні варіанти тренованих моделей нейронних мереж. Їх тренування відбувалося по різному, якісь тренували на меншій кількості даних [15].

4 БІЗНЕС-ПЛАН

4.1 Бізнес-модель продукту

Щоб економічно обґрунтувати даний продукт необхідно скласти бізнес план, який буде описувати взаємодію двох суб'єктів - нас, як виробника продукції, та кінцевого споживача, який використовуватиме автономного робота.

Створення автономного робота з елементами машинного навчання вирішує проблему доступності роботів та сприяє популяризації розробки програмного забезпечення задля задоволення побутових потреб населення.

Шаблон бізнес-моделі і техніки моделювання А.Остельвальдера дозволяють зробити бізнес-модель інтуїтивно очевидною для спеціалістів різного рівня. Бізнес-модель А.Остервальдера складається з 9 частин:

- Ключові партнери;
- Ключова діяльність;
- Ресурси;
- Продукт (пропозиція цінності);
- Взаємодія з клієнтами;
- Логістика;
- Ключовий клієнт;
- Структура витрат;
- Структура доходів.

Описавши всі перераховані вище пункти ми отримаємо цілісну картину розробки та реалізації нашого продукту. Тому пройдемось по кожній із складових бізнес-моделі та розберемо головні питання.

1. Ключові партнери. Головне питання – хто нам допомагає?

Ключовими партнерами, які допоможуть нам реалізувати проект, будуть, перш за все, виробники іграшок, зокрема компанії, які виробляють іграшкові моделі машин. Також, варто залучити допомогу маркетингових компаній, які

допоможуть ефективно просувати наш продукт за допомогою реклами та інших засобів, та торгові компанії, які дозволять нам успішно продавати наш продукт.

Також важливими партнерами стануть наші клієнти, які матимуть змогу комунікувати з розробниками, висловлювати свої зауваження та побажання щодо нового функціоналу. Клієнти зможуть надати відгуки про систему та її функціонал, що дозволить підвищувати рівень якості нашого продукту, вводити нові функції та покращувати вже наявні особливості системи.

2. Ключова діяльність. Головне питання – що ми робимо?

Наше пріоритетне завдання, звичайно ж, розробка, підтримка та вдосконалення автономного машинного робота з елементами машинного навчання. Також важливими напрямами нашої діяльності є розробка маркетингової стратегії, активний маркетинг через соціальні мережі та інші засоби масової інформації, пошук нових клієнтів та партнерів.

3. Ресурси. Головне питання – якими ресурсами ми володіємо?

Для здійснення цілей проекту потрібно мати необхідні економічні ресурси – сукупність засобів, за допомогою яких ми можемо досягти мети і одержати очікуваний результат. Будуть використані такі економічні ресурси: трудові (фахівці з програмної інженерії), фінансові (оплата праці, купівля корпусу робота, оплата інтернету, оплата підтримки робочого середовища), матеріальні (комплектуючі системи), інформаційні (маркетингові кампанії) та інші ресурси.

4. Продукт. Головне питання – яку цінність ми приносимо клієнтам?

Розроблений автономний робот здійснює популяризацію науки та робототехніки, в тому числі серед студентів технічних спеціальностей вищих навчальних закладів. Завдяки таким роботам студенти можуть самостійно вивчати роботу комплектуючих систем, а також розробляти програмне забезпечення з елементами машинного навчання. Окрім простого продажу продукту ми пропонуємо також технічну підтримку даної системи, її постійне вдосконалення, додавання нових функцій та імплементацію нових технічних засобів. Важливою характеристикою робота є мультиплатформність –

управління роботом може здійснюватись на будь-якому пристрої за допомогою Bluetooth технології.

5. Взаємодія з клієнтом. Головне питання – як ми взаємодіємо з клієнтом?

Замовлення робота буде здійснюватись за допомогою різних засобів зв'язку, наприклад, електронне листування, телефонні дзвінки, тощо. Це забезпечуватиме тісну комунікацію з клієнтами та дозволить одночасно отримувати відгуки про автономний автомобіль.

6. Логістика. Головне питання – як клієнти дізнаються про нас?

Розповсюдження інформації про автономного робота відбуватиметься через мережу інтернет, оскільки даний спосіб є найдешевшим та найпопулярнішим серед цільової аудиторії. Необхідно оплатити хостинг та розробити сайт-візитку. Кооперація з маркетинговими компаніями дозволить нам ефективно просувати наш продукт за допомогою реклами, а дистриб'ютори будуть відвідувати вищі навчальні заклади з метою ознайомлення можливих клієнтів з нашою системою.

7. Ключовий клієнт. Головне питання – кому ми допомагаємо?

На жаль, державні вищі навчальні заклади не завжди встигають йти в ногу з сучасними технологіями, а бюрократичні процеси не завжди дозволяють здійснювати закупівлю необхідних систем, зокрема з іноземного ринку. Розроблений продукт придатний одразу для використання без очікувань на доставку комплектуючих.

Тому нашими основними клієнтами стануть вищі навчальні заклади, особливо приватного типу.

8. Структура витрат. Головне питання – що ми вкладаємо?

Варто зазначити, що витрати будуть складатись з трьох частин: витрати на апаратне забезпечення, витрати на розробку програмного забезпечення та витрати на проведення маркетингових кампаній.

До витрат на апаратне забезпечення належать:

- База автомобіля на дистанційному керуванні (3000 грн);
- Raspberry Pi 4+ (1800 грн);

- Камера (500 грн);
- Батарея (900 грн);
- Servo Driver Hat (456 грн);
- Модуль перетворювача блоку живлення купити (150 грн);
- Bluetooth контроллер (300 грн).

Загальні витрати становлять

$$Ззаг.ап = 3000 + 1800 + 500 + 900 + 456 + 150 + 300 = 6\ 656 \text{ гривень.}$$

Тепер розрахуємо витрати на розробку програмного забезпечення. Спочатку визначимо середню годинну оплату програміста. Для цього необхідно спочатку визначити місячне грошове забезпечення програміста. Керуючись даними Держстатслужби, зарплата програміста в регіоні складає приблизно 10000,00 гривень (взято середнє значення грошового забезпечення програміста за 2019 рік). Визначимо число робочих годин у місяці за наступною формулою:

$$n(м) = (N - N(п) - N(в)) \cdot 8$$

де N – загальне число днів у місяці,

N(п) – число святкових днів у місяці,

N(в) – число вихідних днів у місяці.

Число святкових днів у місяці в середньому становить 1 день, а вихідних – 8. Отже, число робочих годин у місяці дорівнює:

$$n(м) = (31 - 1 - 8) \cdot 8 = 184 \text{ [години].}$$

Середня годинна оплата програміста визначається співвідношенням

$$Срозр. = ФЗРсн / n(м) ,$$

де ФЗРсн – місяний фонд грошового забезпечення.

$$Срозр. = 10000/184 = 54{,}34 \text{ [гривень]}$$

Отже, витрати по оплаті праці розробника програми складають:

$$Зрозр. = 368 \cdot 54{,}34 = 19997{,}12 \text{ [гривень]}$$

Витрати на маркетингову кампанію складають 6000,00 гривень

Отже, сумарно витрати на одного робота складатимуть

$$Ззаг = Ззаг.ап + Зрозр + Змарк$$

$$Ззаг = 6656 + 19997{,}12 + 6000 = 32653{,}12 \text{ [гривень]}$$

9. Структура доходів. Головне питання – що ми отримуємо?

Кожен підприємець прагне отримувати в підсумку своєї діяльності якомога більший дохід і прибуток. Власне продаж автономних мобільних роботів буде основним джерелом доходу. Користувачі, які вже купили продукт, отримують також безкоштовну технічну підтримку. За додаткову оплату клієнти можуть замовити нові функції, кількість яких з часом буде збільшуватися. Дохід:

- ціноутворення на унікальність програмного забезпечення – 11000 гривень.

- рентабельність – х3

- термін окупності – 3 місяців

- загальний прибуток за 1 рік роботи – 523837,44 гривень.

- дохід за 1 рік роботи – 132000,00 гривень.

ВИСНОВКИ

Згідно з темою дипломної роботи, мета даної роботи полягає в створенні автономного мобільного робота на основі машинного навчання для участі у міжнародному конкурсі з робототехніки «Sumo Challenge», який проходив у Технологічному Університеті Лодзі.

На першому етапі було проаналізовано предметну область та розглянуто приклади застосування аналогів проекту.

На другому етапі були розглянуті основні визначення та поняття про технології, які були застосовані при розробці.

На третьому етапі дипломної роботи було реалізовано програмну та апаратну частину дипломного проекту. Програмна частина була створена з використанням Python бібліотеки Donkey Car. Апаратна частина була розроблена на базі машинки на дистанційному керуванні Himoto та однопалатному комп’ютері Raspberry PI.

На четвертому етапі було розглянуте економічне обґрунтування проекту та створено його бізнес-модель.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

REFERENCES

1. Автономний робот. URL: https://uk.wikipedia.org/wiki/%D0%90%D0%B2%D1%82%D0%BE%D0%BD%D0%BE%D0%BC%D0%BD%D0%B8%D0%B9_%D1%80%D0%BE%D0%B1%D0%BE%D1%82 (дата звернення 06.12.2019)
2. History of Autonomous robots. URL: https://en.wikipedia.org/wiki/History_of_robots (дата звернення 08.12.2019)
3. World robotic report. URL: https://ifr.org/ifr-press-releases/news/world-robotics-report-2016 (дата звернення: 08.12.2019)
4. Міжнародний конкурс з робототехніки Summo Chalange 2019. URL: https://mif.pnu.edu.ua/2019/12/19/міжнародний-конкурс-з-робототехніки-sumo/ (дата звернення 20.12.2019)
5. It’s 2020. Where are our self-driving cars? URL: https://www.vox.com/future-perfect/2020/2/14/21063487/self-driving-cars-autonomous-vehicles-waymo-cruise-uber (дата звернення 01.03.2020)
6. Future of driving. URL: https://www.tesla.com/autopilot (дата звернення 10.01.2019)
7. Advantages and Disadvantages of Python Programming Language. URL: https://medium.com/@mindfiresolutions.usa/advantages-and-disadvantages-of-python-programming-language-fd0b394f2121 (дата звернення 03.03.2020)
8. What is a Python library and what can I use it for? URL: https://www.quora.com/What-is-a-Python-library-and-what-can-I-use-it-for (дата звернення 05.03.2020)
9. Top 10 Python Libraries You Must Know In 2020ю URL: https://www.edureka.co/blog/python-libraries/ (дата звернення 10.05.2020)
10. Документація до бібліотеки Donkey Car. URL: https://docs.donkeycar.com/ (дата звернення: 01.12.2019)

11. Explained: Neural Networks. URL: http://news.mit.edu/2017/explained-neural-networks-deep-learning-0414 (дата звернення: 03.12.2019)
12. Understanding of Convolutional Neural Network (CNN) – Deep Learning. URL: https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148 (дата звернення: 03.12.2019)
13. Raspberry Pi Control Servo Motor. URL: https://tutorials-raspberrypi.com/raspberry-pi-servo-motor-control/ (дата звернення: 05.12.2019)
14. Autonomous robot. URL: https://en.wikipedia.org/wiki/Autonomous_robot (дата звернення 07.12.2019)
15. Kozlenko, M. I. (2015). The interference immunity of the telemetric information data exchange with autonomous mobile robots. Naukovyi Visnyk Natsionalnoho Hirnychoho Universytetu, (1), 107–113.
16. Документація бібліотеки Keras. URL: https://keras.io/api/ (дата звернення: 06.12.2019)
17. Training a Convolutional Neural Network from scratch. URL: https://towardsdatascience.com/training-a-convolutional-neural-network-from-scratch-2235c2a25754?gi=550b3af8c704 (дата звернення: 07.12.2019)
18. Building a Donkey Car (Part 1 of 2). URL: https://www.jamestharpe.com/donkeycar-build/ (дата звернення: 10.12.2019).
19. M. Kozlenko, A. Bosyi, O. Simkiv, and N. Savchenko, "Artificial intelligence in e-commerce," in Proc. 3rd International Conference "Applied Information Systems and Technologies in the Information Society" (AISTIS), V. Pleskach and V. Mironova, Eds. Taras Shevchenko National University of Kyiv, Kyiv, Ukraine, Sep. 30, 2019, pp. 207-211.
20. The Donkey Car: Part 2 – Build, Calibrate, and Generate Training Data. URL: https://medium.com/@dmccreary/the-donkey-car-part-2-build-calibrate-and-generate-training-data-54265797e8c9 (дата звернення: 10.12.2019)

21. Creating a Neural Network from Scratch in Python. URL: https://stackabuse.com/creating-a-neural-network-from-scratch-in-python/ (дата звернення 15.12.2019).
22. M. Kozlenko and V. Vialkova, "Software Defined Demodulation of Multiple Frequency Shift Keying with Dense Neural Network for Weak Signal Communications," 2020 IEEE 15th International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering (TCSET), Lviv-Slavske, Ukraine, 2020, pp. 590-595, doi: 10.1109/TCSET49122.2020.235501.

ДОДАТОК А

Сирцевий код файлу train.py.

```
#!/usr/bin/env python3
"""
Scripts to train a keras model using tensorflow.
Uses the data written by the donkey v2.2 tub writer,
but faster training with proper sampling of distribution over
tubs.
Has settings for continuous training that will look for new files
as it trains.
Modify on_best_model if you wish continuous training to update
your pi as it builds.
You can drop this in your ~/mycar dir.
Basic usage should feel familiar: python train.py --model
models/mypilot


Usage:
    train.py [--tub=<tub1,tub2,..tubn>] [--file=<file> ...] (--
model=<model>) [--transfer=<model>] [--
type=(linear|latent|categorical|rnn|imu|behavior|3d|look_ahead|ten
sorrt_linear|tflite_linear|coral_tflite_linear)] [--continuous] [-
-aug]

Options:
    -h --help        Show this screen.
    -f --file=<file> A text file containing paths to tub files,
one per line. Option may be used more than once.
"""
import os
import glob
import random
import json
import time
import zlib
from os.path import basename, join, splitext, dirname
import pickle
import datetime

from tensorflow.python import keras
from docopt import docopt
import numpy as np
from PIL import Image

import donkeycar as dk
from donkeycar.parts.datastore import Tub
from donkeycar.parts.keras import KerasLinear, KerasIMU,\
     KerasCategorical, KerasBehavioral, Keras3D_CNN,\
     KerasRNN_LSTM, KerasLatent
```

```
from donkeycar.parts.augment import augment_image
from donkeycar.utils import *


'''
matplotlib can be a pain to setup on a Mac. So handle the case
where it is absent. When present,
use it to generate a plot of training results.
'''
try:
    import matplotlib.pyplot as plt
    do_plot = True
except:
    do_plot = False
    print("matplotlib not installed")


'''
Tub management
'''
def make_key(sample):
    tub_path = sample['tub_path']
    index = sample['index']
    return tub_path + str(index)

def make_next_key(sample, index_offset):
    tub_path = sample['tub_path']
    index = sample['index'] + index_offset
    return tub_path + str(index)


def collate_records(records, gen_records, opts):
    '''
    open all the .json records from records list passed in,
    read their contents,
    add them to a list of gen_records, passed in.
    use the opts dict to specify config choices
    '''

    new_records = {}

    for record_path in records:

        basepath = os.path.dirname(record_path)
        index = get_record_index(record_path)
        sample = { 'tub_path' : basepath, "index" : index }

        key = make_key(sample)

        if key in gen_records:
            continue
```

```
        try:
            with open(record_path, 'r') as fp:
                json_data = json.load(fp)
        except:
            continue

        image_filename = json_data["cam/image_array"]
        image_path = os.path.join(basepath, image_filename)

        sample['record_path'] = record_path
        sample["image_path"] = image_path
        sample["json_data"] = json_data

        angle = float(json_data['user/angle'])
        throttle = float(json_data["user/throttle"])

        if opts['categorical']:
            angle = dk.utils.linear_bin(angle)
            throttle = dk.utils.linear_bin(throttle, N=20,
offset=0, R=opts['cfg'].MODEL_CATEGORICAL_MAX_THROTTLE_RANGE)

        sample['angle'] = angle
        sample['throttle'] = throttle

        try:
            accl_x = float(json_data['imu/acl_x'])
            accl_y = float(json_data['imu/acl_y'])
            accl_z = float(json_data['imu/acl_z'])

            gyro_x = float(json_data['imu/gyr_x'])
            gyro_y = float(json_data['imu/gyr_y'])
            gyro_z = float(json_data['imu/gyr_z'])

            sample['imu_array'] = np.array([accl_x, accl_y,
accl_z, gyro_x, gyro_y, gyro_z])
        except:
            pass

        try:
            behavior_arr =
np.array(json_data['behavior/one_hot_state_array'])
            sample["behavior_arr"] = behavior_arr
        except:
            pass

        sample['img_data'] = None

        # Initialise 'train' to False
        sample['train'] = False
```

```
        # We need to maintain the correct train - validate ratio
across the dataset, even if continous training
        # so don't add this sample to the main records list
(gen_records) yet.
        new_records[key] = sample

    # new_records now contains all our NEW samples
    # - set a random selection to be the training samples based on
the ratio in CFG file
    shufKeys = list(new_records.keys())
    random.shuffle(shufKeys)
    trainCount = 0
    #  Ratio of samples to use as training data, the remaining are
used for evaluation
    targetTrainCount = int(opts['cfg'].TRAIN_TEST_SPLIT *
len(shufKeys))
    for key in shufKeys:
        new_records[key]['train'] = True
        trainCount += 1
        if trainCount >= targetTrainCount:
            break
    # Finally add all the new records to the existing list
    gen_records.update(new_records)

def save_json_and_weights(model, filename):
    '''
    given a keras model and a .h5 filename, save the model file
    in the json format and the weights file in the h5 format
    '''
    if not '.h5' == filename[-3:]:
        raise Exception("Model filename should end with .h5")

    arch = model.to_json()
    json_fnm = filename[:-2] + "json"
    weights_fnm = filename[:-2] + "weights"

    with open(json_fnm, "w") as outfile:
        parsed = json.loads(arch)
        arch_pretty = json.dumps(parsed, indent=4, sort_keys=True)
        outfile.write(arch_pretty)

    model.save_weights(weights_fnm)
    return json_fnm, weights_fnm


class MyCPCallback(keras.callbacks.ModelCheckpoint):
    '''
    custom callback to interact with best val loss during
continuous training
    '''
```

```
    def __init__(self, send_model_cb=None, cfg=None, *args,
**kwargs):
        super(MyCPCallback, self).__init__(*args, **kwargs)
        self.reset_best_end_of_epoch = False
        self.send_model_cb = send_model_cb
        self.last_modified_time = None
        self.cfg = cfg

    def reset_best(self):
        self.reset_best_end_of_epoch = True

    def on_epoch_end(self, epoch, logs=None):
        super(MyCPCallback, self).on_epoch_end(epoch, logs)

        if self.send_model_cb:
            '''
            check whether the file changed and send to the pi
            '''
            filepath = self.filepath.format(epoch=epoch, **logs)
            if os.path.exists(filepath):
                last_modified_time = os.path.getmtime(filepath)
                if self.last_modified_time is None or
self.last_modified_time < last_modified_time:
                    self.last_modified_time = last_modified_time
                    self.send_model_cb(self.cfg, self.model,
filepath)

        '''
        when reset best is set, we want to make sure to run an
entire epoch
        before setting our new best on the new total records
        '''
        if self.reset_best_end_of_epoch:
            self.reset_best_end_of_epoch = False
            self.best = np.Inf


def on_best_model(cfg, model, model_filename):

    model.save(model_filename, include_optimizer=False)

    if not cfg.SEND_BEST_MODEL_TO_PI:
        return

    on_windows = os.name == 'nt'

    #If we wish, send the best model to the pi.
    #On mac or linux we have scp:
    if not on_windows:
        print('sending model to the pi')
```

```
        command = 'scp %s %s@%s:~/%s/models/;' % (model_filename,
cfg.PI_USERNAME, cfg.PI_HOSTNAME, cfg.PI_DONKEY_ROOT)

        print("sending", command)
        res = os.system(command)
        print(res)

    else: #yes, we are on windows machine

        #On windoz no scp. In order to use this you must first
setup
        #an ftp daemon on the pi. ie. sudo apt-get install vsftpd
        #and then make sure you enable write permissions in the
conf
        try:
            import paramiko
        except:
            raise Exception("first install paramiko: pip install
paramiko")

        host = cfg.PI_HOSTNAME
        username = cfg.PI_USERNAME
        password = cfg.PI_PASSWD
        server = host
        files = []

        localpath = model_filename
        remotepath = '/home/%s/%s/%s' %(username,
cfg.PI_DONKEY_ROOT, model_filename.replace('\\', '/'))
        files.append((localpath, remotepath))

        print("sending", files)

        try:
            ssh = paramiko.SSHClient()

ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())

ssh.load_host_keys(os.path.expanduser(os.path.join("~", ".ssh",
"known_hosts")))
            ssh.connect(server, username=username,
password=password)
            sftp = ssh.open_sftp()

            for localpath, remotepath in files:
                sftp.put(localpath, remotepath)

            sftp.close()
            ssh.close()
            print("send succeded")
        except:
```

```
            print("send failed")


def train(cfg, tub_names, model_name, transfer_model, model_type,
continuous, aug):
    '''
    use the specified data in tub_names to train an artifical
neural network
    saves the output trained model as model_name
    '''
    verbose = cfg.VEBOSE_TRAIN

    if model_type is None:
        model_type = cfg.DEFAULT_MODEL_TYPE

    if "tflite" in model_type:
        #even though we are passed the .tflite output file, we
train with an intermediate .h5
        #output and then convert to final .tflite at the end.
        assert(".tflite" in model_name)
        #we only support the linear model type right now for
tflite
        assert("linear" in model_type)
        model_name = model_name.replace(".tflite", ".h5")
    elif "tensorrt" in model_type:
        #even though we are passed the .uff output file, we train
with an intermediate .h5
        #output and then convert to final .uff at the end.
        assert(".uff" in model_name)
        #we only support the linear model type right now for
tensorrt
        assert("linear" in model_type)
        model_name = model_name.replace(".uff", ".h5")

    if model_name and not '.h5' == model_name[-3:]:
        raise Exception("Model filename should end with .h5")

    if continuous:
        print("continuous training")

    gen_records = {}
    opts = { 'cfg' : cfg}

    if "linear" in model_type:
        train_type = "linear"
    else:
        train_type = model_type

    kl = get_model_by_type(train_type, cfg=cfg)
```

```
    opts['categorical'] = type(kl) in [KerasCategorical,
KerasBehavioral]

    print('training with model type', type(kl))

    if transfer_model:
        print('loading weights from model', transfer_model)
        kl.load(transfer_model)

        #when transfering models, should we freeze all but the
last N layers?
        if cfg.FREEZE_LAYERS:
            num_to_freeze = len(kl.model.layers) -
cfg.NUM_LAST_LAYERS_TO_TRAIN
            print('freezing %d layers' % num_to_freeze)
            for i in range(num_to_freeze):
                kl.model.layers[i].trainable = False

    if cfg.OPTIMIZER:
        kl.set_optimizer(cfg.OPTIMIZER, cfg.LEARNING_RATE,
cfg.LEARNING_RATE_DECAY)

    kl.compile()

    if cfg.PRINT_MODEL_SUMMARY:
        print(kl.model.summary())

    opts['keras_pilot'] = kl
    opts['continuous'] = continuous
    opts['model_type'] = model_type

    extract_data_from_pickles(cfg, tub_names)

    records = gather_records(cfg, tub_names, opts, verbose=True)
    print('collating %d records ...' % (len(records)))
    collate_records(records, gen_records, opts)

    def generator(save_best, opts, data, batch_size,
isTrainSet=True, min_records_to_train=1000):

        num_records = len(data)

        while True:

            if isTrainSet and opts['continuous']:
                '''
                When continuous training, we look for new records
after each epoch.
                This will add new records to the train and
validation set.
                '''
```

```
                records = gather_records(cfg, tub_names, opts)
                if len(records) > num_records:
                    collate_records(records, gen_records, opts)
                    new_num_rec = len(data)
                    if new_num_rec > num_records:
                        print('picked up', new_num_rec -
num_records, 'new records!')
                        num_records = new_num_rec
                        save_best.reset_best()
                if num_records < min_records_to_train:
                    print("not enough records to train. need %d,
have %d. waiting..." % (min_records_to_train, num_records))
                    time.sleep(10)
                    continue

            batch_data = []

            keys = list(data.keys())

            random.shuffle(keys)

            kl = opts['keras_pilot']

            if type(kl.model.output) is list:
                model_out_shape = (2, 1)
            else:
                model_out_shape = kl.model.output.shape

            if type(kl.model.input) is list:
                model_in_shape = (2, 1)
            else:
                model_in_shape = kl.model.input.shape

            has_imu = type(kl) is KerasIMU
            has_bvh = type(kl) is KerasBehavioral
            img_out = type(kl) is KerasLatent

            if img_out:
                import cv2

            for key in keys:

                if not key in data:
                    continue

                _record = data[key]

                if _record['train'] != isTrainSet:
                    continue

                if continuous:
```

```
                    #in continuous mode we need to handle files
getting deleted
                    filename = _record['image_path']
                    if not os.path.exists(filename):
                        data.pop(key, None)
                        continue

                batch_data.append(_record)

                if len(batch_data) == batch_size:
                    inputs_img = []
                    inputs_imu = []
                    inputs_bvh = []
                    angles = []
                    throttles = []
                    out_img = []
                    out = []

                    for record in batch_data:
                        #get image data if we don't already have
it
                        if record['img_data'] is None:
                            filename = record['image_path']

                            img_arr =
load_scaled_image_arr(filename, cfg)

                            if img_arr is None:
                                break

                            if aug:
                                img_arr = augment_image(img_arr)

                            if cfg.CACHE_IMAGES:
                                record['img_data'] = img_arr
                        else:
                            img_arr = record['img_data']

                        if img_out:
                            rz_img_arr = cv2.resize(img_arr, (127,
127)) / 255.0

out_img.append(rz_img_arr[:,:,0].reshape((127, 127, 1)))

                        if has_imu:
                            inputs_imu.append(record['imu_array'])

                        if has_bvh:

inputs_bvh.append(record['behavior_arr'])
```

```
                        inputs_img.append(img_arr)
                        angles.append(record['angle'])
                        throttles.append(record['throttle'])
                        out.append([record['angle'],
record['throttle']])

                    if img_arr is None:
                        continue

                    img_arr =
np.array(inputs_img).reshape(batch_size,\
                        cfg.TARGET_H, cfg.TARGET_W, cfg.TARGET_D)

                    if has_imu:
                        X = [img_arr, np.array(inputs_imu)]
                    elif has_bvh:
                        X = [img_arr, np.array(inputs_bvh)]
                    else:
                        X = [img_arr]

                    if img_out:
                        y = [out_img, np.array(angles),
np.array(throttles)]
                    elif model_out_shape[1] == 2:
                        y = [np.array([out]).reshape(batch_size,
2) ]
                    else:
                        y = [np.array(angles),
np.array(throttles)]

                    yield X, y

                    batch_data = []

    model_path = os.path.expanduser(model_name)


    #checkpoint to save model after each epoch and send best to
the pi.
    save_best = MyCPCallback(send_model_cb=on_best_model,
                                    filepath=model_path,
                                    monitor='val_loss',
                                    verbose=verbose,
                                    save_best_only=True,
                                    mode='min',
                                    cfg=cfg)

    train_gen = generator(save_best, opts, gen_records,
cfg.BATCH_SIZE, True)
    val_gen = generator(save_best, opts, gen_records,
cfg.BATCH_SIZE, False)
```

```
    total_records = len(gen_records)

    num_train = 0
    num_val = 0

    for key, _record in gen_records.items():
        if _record['train'] == True:
            num_train += 1
        else:
            num_val += 1

    print("train: %d, val: %d" % (num_train, num_val))
    print('total records: %d' %(total_records))

    if not continuous:
        steps_per_epoch = num_train // cfg.BATCH_SIZE
    else:
        steps_per_epoch = 100

    val_steps = num_val // cfg.BATCH_SIZE
    print('steps_per_epoch', steps_per_epoch)

    cfg.model_type = model_type

    go_train(kl, cfg, train_gen, val_gen, gen_records, model_name,
steps_per_epoch, val_steps, continuous, verbose, save_best)



def go_train(kl, cfg, train_gen, val_gen, gen_records, model_name,
steps_per_epoch, val_steps, continuous, verbose, save_best=None):

    start = time.time()

    model_path = os.path.expanduser(model_name)

    #checkpoint to save model after each epoch and send best to
the pi.
    if save_best is None:
        save_best = MyCPCallback(send_model_cb=on_best_model,
                                 filepath=model_path,
                                 monitor='val_loss',
                                 verbose=verbose,
                                 save_best_only=True,
                                 mode='min',
                                 cfg=cfg)

    #stop training if the validation error stops improving.
    early_stop = keras.callbacks.EarlyStopping(monitor='val_loss',
```

```
min_delta=cfg.MIN_DELTA,

patience=cfg.EARLY_STOP_PATIENCE,
                                                    verbose=verbose,
                                                    mode='auto')

    if steps_per_epoch < 2:
        raise Exception("Too little data to train. Please record
more records.")

    if continuous:
        epochs = 100000
    else:
        epochs = cfg.MAX_EPOCHS

    workers_count = 1
    use_multiprocessing = False

    callbacks_list = [save_best]

    if cfg.USE_EARLY_STOP and not continuous:
        callbacks_list.append(early_stop)

    history = kl.model.fit_generator(
                    train_gen,
                    steps_per_epoch=steps_per_epoch,
                    epochs=epochs,
                    verbose=cfg.VEBOSE_TRAIN,
                    validation_data=val_gen,
                    callbacks=callbacks_list,
                    validation_steps=val_steps,
                    workers=workers_count,
                    use_multiprocessing=use_multiprocessing)

    full_model_val_loss = min(history.history['val_loss'])
    max_val_loss = full_model_val_loss +
cfg.PRUNE_VAL_LOSS_DEGRADATION_LIMIT

    duration_train = time.time() - start
    print("Training completed in %s." %
str(datetime.timedelta(seconds=round(duration_train))) )

    print("\n\n---------- Best Eval Loss :%f ---------" %
save_best.best)

    if cfg.SHOW_PLOT:
        try:
            if do_plot:
                plt.figure(1)
```

```
                # Only do accuracy if we have that data (e.g.
categorical outputs)
                if 'angle_out_acc' in history.history:
                    plt.subplot(121)

                # summarize history for loss
                plt.plot(history.history['loss'])
                plt.plot(history.history['val_loss'])
                plt.title('model loss')
                plt.ylabel('loss')
                plt.xlabel('epoch')
                plt.legend(['train', 'validate'], loc='upper
right')

                # summarize history for acc
                if 'angle_out_acc' in history.history:
                    plt.subplot(122)
                    plt.plot(history.history['angle_out_acc'])
                    plt.plot(history.history['val_angle_out_acc'])
                    plt.title('model angle accuracy')
                    plt.ylabel('acc')
                    plt.xlabel('epoch')
                    #plt.legend(['train', 'validate'], loc='upper
left')

                plt.savefig(model_path + '_loss_acc_%f.png' %
save_best.best)
                plt.show()
            else:
                print("not saving loss graph because matplotlib
not set up.")
        except Exception as ex:
            print("problems with loss graph: {}".format( ex ) )

    #Save tflite, optionally in the int quant format for Coral TPU
    if "tflite" in cfg.model_type:
        print("\n\n--------- Saving TFLite Model ---------")
        tflite_fnm = model_path.replace(".h5", ".tflite")
        assert(".tflite" in tflite_fnm)

        prepare_for_coral = "coral" in cfg.model_type

        if prepare_for_coral:
            #compile a list of records to calibrate the
quantization
            data_list = []
            max_items = 1000
            for key, _record in gen_records.items():
                data_list.append(_record)
                if len(data_list) == max_items:
                    break
```

```
            stride = 1
            num_calibration_steps = len(data_list) // stride

            #a generator function to help train the quantizer with
the expected range of data from inputs
            def representative_dataset_gen():
                start = 0
                end = stride
                for _ in range(num_calibration_steps):
                    batch_data = data_list[start:end]
                    inputs = []

                    for record in batch_data:
                        filename = record['image_path']
                        img_arr = load_scaled_image_arr(filename,
cfg)
                        inputs.append(img_arr)

                    start += stride
                    end += stride

                    # Get sample input data as a numpy array in a
method of your choosing.
                    yield [ np.array(inputs,
dtype=np.float32).reshape(stride, cfg.TARGET_H, cfg.TARGET_W,
cfg.TARGET_D) ]
        else:
            representative_dataset_gen = None

        from donkeycar.parts.tflite import keras_model_to_tflite
        keras_model_to_tflite(model_path, tflite_fnm,
representative_dataset_gen)
        print("Saved TFLite model:", tflite_fnm)
        if prepare_for_coral:
            print("compile for Coral w: edgetpu_compiler",
tflite_fnm)
            os.system("edgetpu_compiler " + tflite_fnm)

    #Save tensorrt
    if "tensorrt" in cfg.model_type:
        print("\n\n--------- Saving TensorRT Model ---------")
        # TODO RAHUL
        # flatten model_path
        # convert to uff
        # print("Saved TensorRT model:", uff_filename)

    if cfg.PRUNE_CNN:
        base_model_path = splitext(model_name)[0]
        cnn_channels = get_total_channels(kl.model)
```

```
        print('original model with {}
channels'.format(cnn_channels))
        prune_gen = SequencePredictionGenerator(gen_records, cfg)
        target_channels = int(cnn_channels * (1 -
(float(cfg.PRUNE_PERCENT_TARGET) / 100.0)))

        print('Target channels of {0} remaining with {1:.00%}
percent removal per iteration'.format(target_channels,
cfg.PRUNE_PERCENT_PER_ITERATION / 100))

        from keras.models import load_model
        prune_loss = 0
        while cnn_channels > target_channels:
            save_best.reset_best()
            model, channels_deleted = prune(kl.model, prune_gen,
1, cfg)
            cnn_channels -= channels_deleted
            kl.model = model
            kl.compile()
            kl.model.summary()

            #stop training if the validation error stops
improving.
            early_stop =
keras.callbacks.EarlyStopping(monitor='val_loss',

min_delta=cfg.MIN_DELTA,

patience=cfg.EARLY_STOP_PATIENCE,

verbose=verbose,

mode='auto')

            history = kl.model.fit_generator(
                        train_gen,
                        steps_per_epoch=steps_per_epoch,
                        epochs=epochs,
                        verbose=cfg.VEBOSE_TRAIN,
                        validation_data=val_gen,
                        validation_steps=val_steps,
                        workers=workers_count,
                        callbacks=[early_stop],
                        use_multiprocessing=use_multiprocessing)

            prune_loss = min(history.history['val_loss'])
            print('prune val_loss this iteration:
{}'.format(prune_loss))

            # If loss breaks the threshhold
            if prune_loss < max_val_loss:
```

```
model.save('{}_prune_{}_filters.h5'.format(base_model_path,
cnn_channels))
            else:
                break

        print('pruning stopped at {} with a target of
{}'.format(cnn_channels, target_channels))


class SequencePredictionGenerator(keras.utils.Sequence):
    """
    Provides a thread safe data generator for the Keras
predict_generator for use with kerasergeon.
    """
    def __init__(self, data, cfg):
        data = list(data.values())
        self.n = int(len(data) *
cfg.PRUNE_EVAL_PERCENT_OF_DATASET)
        self.data = data[:self.n]
        self.batch_size = cfg.BATCH_SIZE
        self.cfg = cfg

    def __len__(self):
        return int(np.ceil(len(self.data) /
float(self.batch_size)))

    def __getitem__(self, idx):
        batch_data = self.data[idx * self.batch_size:(idx + 1) *
self.batch_size]

        images = []
        for data in batch_data:
            path = data['image_path']
            img_arr = load_scaled_image_arr(path, self.cfg)
            images.append(img_arr)

        return np.array(images), np.array([])

def sequence_train(cfg, tub_names, model_name, transfer_model,
model_type, continuous, aug):
    '''
    use the specified data in tub_names to train an artifical
neural network
    saves the output trained model as model_name
    trains models which take sequence of images
    '''
    assert(not continuous)

    print("sequence of images training")
```

```
    kl = dk.utils.get_model_by_type(model_type=model_type,
cfg=cfg)

    tubs = gather_tubs(cfg, tub_names)

    verbose = cfg.VEBOSE_TRAIN

    records = []

    for tub in tubs:
        record_paths = glob.glob(os.path.join(tub.path,
'record_*.json'))
        print("Tub:", tub.path, "has", len(record_paths),
'records')

        record_paths.sort(key=get_record_index)
        records += record_paths


    print('collating records')
    gen_records = {}

    for record_path in records:

        with open(record_path, 'r') as fp:
            json_data = json.load(fp)

        basepath = os.path.dirname(record_path)
        image_filename = json_data["cam/image_array"]
        image_path = os.path.join(basepath, image_filename)
        sample = { 'record_path' : record_path, "image_path" :
image_path, "json_data" : json_data }

        sample["tub_path"] = basepath
        sample["index"] = get_image_index(image_filename)

        angle = float(json_data['user/angle'])
        throttle = float(json_data["user/throttle"])

        sample['target_output'] = np.array([angle, throttle])
        sample['angle'] = angle
        sample['throttle'] = throttle

        sample['img_data'] = None

        key = make_key(sample)

        gen_records[key] = sample
```

```
    print('collating sequences')

    sequences = []

    target_len = cfg.SEQUENCE_LENGTH
    look_ahead = False

    if model_type == "look_ahead":
        target_len = cfg.SEQUENCE_LENGTH * 2
        look_ahead = True

    for k, sample in gen_records.items():

        seq = []

        for i in range(target_len):
            key = make_next_key(sample, i)
            if key in gen_records:
                seq.append(gen_records[key])
            else:
                continue

        if len(seq) != target_len:
            continue

        sequences.append(seq)

    print("collated", len(sequences), "sequences of length",
target_len)

    #shuffle and split the data
    train_data, val_data  = train_test_split(sequences,
test_size=(1 - cfg.TRAIN_TEST_SPLIT))


    def generator(data, opt, batch_size=cfg.BATCH_SIZE):
        num_records = len(data)

        while True:
            #shuffle again for good measure
            random.shuffle(data)

            for offset in range(0, num_records, batch_size):
                batch_data = data[offset:offset+batch_size]

                if len(batch_data) != batch_size:
                    break

                b_inputs_img = []
                b_vec_in = []
                b_labels = []
```

```
                b_vec_out = []

                for seq in batch_data:
                    inputs_img = []
                    vec_in = []
                    labels = []
                    vec_out = []
                    num_images_target = len(seq)
                    iTargetOutput = -1
                    if opt['look_ahead']:
                        num_images_target = cfg.SEQUENCE_LENGTH
                        iTargetOutput = cfg.SEQUENCE_LENGTH - 1

                    for iRec, record in enumerate(seq):
                        #get image data if we don't already have
it
                        if len(inputs_img) < num_images_target:
                            if record['img_data'] is None:
                                img_arr =
load_scaled_image_arr(record['image_path'], cfg)
                                if img_arr is None:
                                    break
                                if aug:
                                    img_arr =
augment_image(img_arr)

                                if cfg.CACHE_IMAGES:
                                    record['img_data'] = img_arr
                            else:
                                img_arr = record['img_data']

                            inputs_img.append(img_arr)

                        if iRec >= iTargetOutput:
                            vec_out.append(record['angle'])
                            vec_out.append(record['throttle'])
                        else:
                            vec_in.append(0.0) #record['angle'])
                            vec_in.append(0.0)
#record['throttle'])

                    label_vec =
seq[iTargetOutput]['target_output']

                    if look_ahead:
                        label_vec = np.array(vec_out)

                    labels.append(label_vec)

                    b_inputs_img.append(inputs_img)
                    b_vec_in.append(vec_in)
```

```
                    b_labels.append(labels)


                if look_ahead:
                    X =
[np.array(b_inputs_img).reshape(batch_size,\
                        cfg.TARGET_H, cfg.TARGET_W,
cfg.SEQUENCE_LENGTH)]
                    X.append(np.array(b_vec_in))
                    y = np.array(b_labels).reshape(batch_size,
(cfg.SEQUENCE_LENGTH + 1) * 2)
                else:
                    X =
[np.array(b_inputs_img).reshape(batch_size,\
                        cfg.SEQUENCE_LENGTH, cfg.TARGET_H,
cfg.TARGET_W, cfg.TARGET_D)]
                    y = np.array(b_labels).reshape(batch_size, 2)

                yield X, y

    opt = { 'look_ahead' : look_ahead, 'cfg' : cfg }

    train_gen = generator(train_data, opt)
    val_gen = generator(val_data, opt)

    model_path = os.path.expanduser(model_name)

    total_records = len(sequences)
    total_train = len(train_data)
    total_val = len(val_data)

    print('train: %d, validation: %d' %(total_train, total_val))
    steps_per_epoch = total_train // cfg.BATCH_SIZE
    val_steps = total_val // cfg.BATCH_SIZE
    print('steps_per_epoch', steps_per_epoch)

    if steps_per_epoch < 2:
        raise Exception("Too little data to train. Please record
more records.")

    cfg.model_type = model_type

    go_train(kl, cfg, train_gen, val_gen, gen_records, model_name,
steps_per_epoch, val_steps, continuous, verbose)

    '''
    kl.train(train_gen,
        val_gen,
        saved_model_path=model_path,
        steps=steps_per_epoch,
```

```
        train_split=cfg.TRAIN_TEST_SPLIT,
        use_early_stop = cfg.USE_EARLY_STOP)
    '''


def multi_train(cfg, tub, model, transfer, model_type, continuous,
aug):
    '''
    choose the right regime for the given model type
    '''
    train_fn = train
    if model_type in ("rnn",'3d','look_ahead'):
        train_fn = sequence_train

    train_fn(cfg, tub, model, transfer, model_type, continuous,
aug)


def prune(model, validation_generator, val_steps, cfg):
    percent_pruning = float(cfg.PRUNE_PERCENT_PER_ITERATION)
    total_channels = get_total_channels(model)
    n_channels_delete = int(math.floor(percent_pruning / 100 *
total_channels))

    apoz_df = get_model_apoz(model, validation_generator)

    model = prune_model(model, apoz_df, n_channels_delete)

    name = '{}/model_pruned_{}_percent.h5'.format(cfg.MODELS_PATH,
percent_pruning)

    model.save(name)

    return model, n_channels_delete


def extract_data_from_pickles(cfg, tubs):
    """
    Extracts record_{id}.json and image from a pickle with the
same id if exists in the tub.
    Then writes extracted json/jpg along side the source pickle
that tub.
    This assumes the format {id}.pickle in the tub directory.
    :param cfg: config with data location configuration. Generally
the global config object.
    :param tubs: The list of tubs involved in training.
    :return: implicit None.
    """
    t_paths = gather_tub_paths(cfg, tubs)
    for tub_path in t_paths:
        file_paths = glob.glob(join(tub_path, '*.pickle'))
```

```
        print('found {} pickles writing json records and images in
tub {}'.format(len(file_paths), tub_path))
        for file_path in file_paths:
            # print('loading data from {}'.format(file_paths))
            with open(file_path, 'rb') as f:
                p = zlib.decompress(f.read())
            data = pickle.loads(p)

            base_path = dirname(file_path)
            filename = splitext(basename(file_path))[0]
            image_path = join(base_path, filename + '.jpg')
            img =
Image.fromarray(np.uint8(data['val']['cam/image_array']))
            img.save(image_path)

            data['val']['cam/image_array'] = filename + '.jpg'

            with open(join(base_path,
'record_{}.json'.format(filename)), 'w') as f:
                json.dump(data['val'], f)


def prune_model(model, apoz_df, n_channels_delete):
    from kerassurgeon import Surgeon
    import pandas as pd

    # Identify 5% of channels with the highest APoZ in model
    sorted_apoz_df = apoz_df.sort_values('apoz', ascending=False)
    high_apoz_index = sorted_apoz_df.iloc[0:n_channels_delete, :]

    # Create the Surgeon and add a 'delete_channels' job for each
layer
    # whose channels are to be deleted.
    surgeon = Surgeon(model, copy=True)
    for name in high_apoz_index.index.unique().values:
        channels = list(pd.Series(high_apoz_index.loc[name,
'index'],
                                  dtype=np.int64).values)
        surgeon.add_job('delete_channels', model.get_layer(name),
                        channels=channels)
    # Delete channels
    return surgeon.operate()


def get_total_channels(model):
    start = None
    end = None
    channels = 0
    for layer in model.layers[start:end]:
        if layer.__class__.__name__ == 'Conv2D':
            channels += layer.filters
```

```
    return channels


def get_model_apoz(model, generator):
    from kerassurgeon.identify import get_apoz
    import pandas as pd

    # Get APoZ
    start = None
    end = None
    apoz = []
    for layer in model.layers[start:end]:
        if layer.__class__.__name__ == 'Conv2D':
            print(layer.name)
            apoz.extend([(layer.name, i, value) for (i, value)
                         in enumerate(get_apoz(model, layer,
generator))])

    layer_name, index, apoz_value = zip(*apoz)
    apoz_df = pd.DataFrame({'layer': layer_name, 'index': index,
                            'apoz': apoz_value})
    apoz_df = apoz_df.set_index('layer')
    return apoz_df


def removeComments( dir_list ):
    for i in reversed(range(len(dir_list))):
        if dir_list[i].startswith("#"):
            del dir_list[i]
        elif len(dir_list[i]) == 0:
            del dir_list[i]

def preprocessFileList( filelist ):
    dirs = []
    if filelist is not None:
        for afile in filelist:
            with open(afile, "r") as f:
                tmp_dirs = f.read().split('\n')
                dirs.extend(tmp_dirs)

    removeComments( dirs )
    return dirs

if __name__ == "__main__":
    args = docopt(__doc__)
    cfg = dk.load_config()
    tub = args['--tub']
    model = args['--model']
    transfer = args['--transfer']
    model_type = args['--type']
    continuous = args['--continuous']
```

```
    aug = args['--aug']

    dirs = preprocessFileList( args['--file'] )
    if tub is not None:
        tub_paths = [os.path.expanduser(n) for n in
tub.split(',')]
        dirs.extend( tub_paths )

    multi_train(cfg, dirs, model, transfer, model_type,
continuous, aug)
```

ДОДАТОК Б

Сирцевий код файлу manage.py

```
#!/usr/bin/env python3
"""
Scripts to drive a donkey 2 car

Usage:
    manage.py (drive) [--model=<model>] [--js]
        [--
type=(linear|categorical|rnn|imu|behavior|3d|localizer|latent)]
        [--camera=(single|stereo)] [--meta=<key:value> ...]
    manage.py (train) [--tub=<tub1,tub2,..tubn>] [--file=<file>
...]
        (--model=<model>) [--transfer=<model>]
        [--
type=(linear|categorical|rnn|imu|behavior|3d|localizer)]
        [--continuous] [--aug]


Options:
    -h --help          Show this screen.
    --js               Use physical joystick.
    -f --file=<file>   A text file containing paths to tub files,
one per line.
                       Option may be used more than once.
    --meta=<key:value> Key/Value strings describing describing a
piece of meta
                       data about this drive. Option may be used
more than
                       once.
"""

from docopt import docopt

import donkeycar as dk

from donkeycar.parts.transform import TriggeredCallback,
DelayedTrigger
from donkeycar.parts.datastore import TubHandler
from donkeycar.parts.controller import LocalWebController, \
    JoystickController, get_js_controller, JoyStickSub
from donkeycar.parts.throttle_filter import ThrottleFilter
from donkeycar.parts.file_watcher import FileWatcher
from donkeycar.parts.launch import AiLaunch
from donkeycar.utils import *
from tensorflow.python import keras

from car.camera import CameraConfiguration
from car.controller import ControllerConfiguration
```

```
def drive(cfg, model_path: str = None, use_joystick=False,
model_type=None,
          camera_type='single', meta="linear"):
    """
    Construct a working robotic vehicle from many parts. Each part
runs as a
    job in the Vehicle loop, calling either it's run or
run_threaded method
    depending on the constructor flag `threaded`. All parts are
updated one
    after another at the framerate given in cfg.DRIVE_LOOP_HZ
assuming each
    part finishes processing in a timely manner. Parts may have
named
    outputs and inputs. The framework handles passing named
outputs to parts
    requesting the same named input.
    """
    print("[*] Started configure vehicle...")

    # Initialize car
    vehicle = dk.vehicle.Vehicle()

    print("[*] Setup camera...")
    CameraConfiguration(vehicle, camera_type).configure_camera()
    print("[*] Camera is ready...")

    print("[*] Setup joystick...")
    controller = ControllerConfiguration(vehicle)
    print("[*] Joystick is ready...")

    # this throttle filter will allow one tap back for esc reverse
    th_filter = ThrottleFilter()
    vehicle.add(th_filter, inputs=['user/throttle'],
outputs=['user/throttle'])

    # See if we should even run the pilot module.
    # This is only needed because the part run_condition only
accepts boolean
    class PilotCondition:
        def run(self, mode):
            return mode != "user"

    vehicle.add(PilotCondition(), inputs=['user/mode'],
outputs=['run_pilot'])

    class RecordTracker:
        def __init__(self):
            self.last_num_rec_print = 0
```

```
            self.force_alert = 0

        def run(self, num_records):
            if num_records is None:
                return 0

            if self.last_num_rec_print != num_records or
self.force_alert:
                self.last_num_rec_print = num_records

                if num_records % 10 == 0:
                    print(f"[I] Recorded {num_records} records")

            return 0

    rec_tracker_part = RecordTracker()
    vehicle.add(rec_tracker_part, inputs=["tub/num_records"],
                outputs=['records/alert'])

    if cfg.AUTO_RECORD_ON_THROTTLE and isinstance(ctr,
JoystickController):
        # then we are not using the circle button. hijack that to
force a
        # record count indication
        def show_record_amount_status():
            rec_tracker_part.last_num_rec_print = 0
            rec_tracker_part.force_alert = 1

        # Todo: move to controller
        ctr.set_button_down_trigger('circle',
show_record_amount_status)

    class ImgPreProcess:
        """
        preprocess camera image for inference.
        normalize and crop if needed.
        """

        def __init__(self, cfg):
            self.cfg = cfg

        def run(self, img_arr):
            return normalize_and_crop(img_arr, self.cfg)

    inf_input = 'cam/normalized/cropped'
    vehicle.add(ImgPreProcess(cfg),
                inputs=['cam/image_array'],
                outputs=[inf_input],
                run_condition='run_pilot')
    inputs = [inf_input]
```

```
    def load_model(kl, model_path):
        print(f"[*] Starting loading model from {model_path}")
        start = time.time()
        kl.load(model_path)
        print(f"[*] Model loaded in {time.time() - start} sec.")

    def load_weights(kl, weights_path):
        try:
            print(f"[*] Starting loading model weights from
{weights_path}")
            start = time.time()
            kl.model.load_weights(weights_path)
            print(f"[*] Model weights loaded in {time.time() -
start} sec.")
        except Exception as e:
            print(e)
            print(f"[!] Problems loading weights from
{weights_path}")

    def load_json_model(kl, json_filename):
        print(f"[*] Starting loading model json from
{json_filename}")
        try:
            start = time.time()
            with open(json_filename, 'r') as handle:
                kl.model =
keras.models.model_from_json(handle.read())
            print(f"[*] Model loaded in {time.time() - start}
sec.")
        except Exception as e:
            print(e)
            print(f"[!] Problems loading model json from
{json_filename}")

    if model_path:
        # When we have a model, first create an appropriate Keras
part
        keras_model = dk.utils.get_model_by_type(model_type, cfg)
        if not model_path.endswith(".h5"):
            raise Exception(
                "[!] Unknown extension type on model file! Use
`.h5`"
            )

        # when we have a .h5 extension
        # load everything from the model file
        load_model(keras_model, model_path)

        def reload_model(filename):
            load_model(keras_model, filename)
```

```
        model_reload_cb = reload_model

        # this part will signal visual LED, if connected
        vehicle.add(FileWatcher(model_path, verbose=True),
                    outputs=['modelfile/modified'])

        # these parts will reload the model file, but only when ai
is
        # running so we don't interrupt user driving
        vehicle.add(FileWatcher(model_path),
outputs=['modelfile/dirty'],
                    run_condition="ai_running")
        vehicle.add(DelayedTrigger(100),
inputs=['modelfile/dirty'],
                    outputs=['modelfile/reload'],
run_condition="ai_running")
        vehicle.add(TriggeredCallback(model_path,
model_reload_cb),
                    inputs=["modelfile/reload"],
run_condition="ai_running")

        outputs = ['pilot/angle', 'pilot/throttle']

        if cfg.TRAIN_LOCALIZER:
            outputs.append("pilot/loc")

        vehicle.add(keras_model, inputs=inputs,
                    outputs=outputs,
                    run_condition='run_pilot')

        # Choose what inputs should change the car.

    class DriveMode:
        def run(self, mode,
                user_angle, user_throttle,
                pilot_angle, pilot_throttle):
            if mode == 'user':
                return user_angle, user_throttle

            elif mode == 'local_angle':
                return pilot_angle, user_throttle

            else:
                return pilot_angle, pilot_throttle *
cfg.AI_THROTTLE_MULT

    vehicle.add(DriveMode(),
                inputs=['user/mode', 'user/angle',
'user/throttle',
                        'pilot/angle', 'pilot/throttle'],
                outputs=['angle', 'throttle'])
```

```
    # to give the car a boost when starting ai mode in a race.
    aiLauncher = AiLaunch(cfg.AI_LAUNCH_DURATION,
cfg.AI_LAUNCH_THROTTLE,
                          cfg.AI_LAUNCH_KEEP_ENABLED)

    vehicle.add(aiLauncher,
                inputs=['user/mode', 'throttle'],
                outputs=['throttle'])

    if isinstance(ctr, JoystickController):
        ctr.set_button_down_trigger(cfg.AI_LAUNCH_ENABLE_BUTTON,
                                    aiLauncher.enable_ai_launch)

    class AiRunCondition:
        """
        A bool part to let us know when ai is running.
        """

        def run(self, mode):
            return mode != "user"

    vehicle.add(AiRunCondition(), inputs=['user/mode'],
outputs=['ai_running'])

    # Ai Recording
    class AiRecordingCondition:
        """
        return True when ai mode, otherwize respect user mode
recording flag
        """

        def run(self, mode, recording):
            return recording if mode == 'user' else True

    if cfg.RECORD_DURING_AI:
        vehicle.add(
            AiRecordingCondition(),
            inputs=['user/mode', 'recording'],
            outputs=['recording']
        )

    # Drive train setup
    if cfg.DONKEY_GYM:
        pass

    elif cfg.DRIVE_TRAIN_TYPE == "SERVO_ESC":
        from donkeycar.parts.actuator import PCA9685, PWMSteering,
PWMThrottle

        steering_controller = PCA9685(cfg.STEERING_CHANNEL,
```

```
                                        cfg.PCA9685_I2C_ADDR,

busnum=cfg.PCA9685_I2C_BUSNUM)
        steering = PWMSteering(controller=steering_controller,
                               left_pulse=cfg.STEERING_LEFT_PWM,
                               right_pulse=cfg.STEERING_RIGHT_PWM)

        throttle_controller = PCA9685(cfg.THROTTLE_CHANNEL,
                                      cfg.PCA9685_I2C_ADDR,

busnum=cfg.PCA9685_I2C_BUSNUM)
        throttle = PWMThrottle(controller=throttle_controller,
                               max_pulse=cfg.THROTTLE_FORWARD_PWM,

zero_pulse=cfg.THROTTLE_STOPPED_PWM,
                               min_pulse=cfg.THROTTLE_REVERSE_PWM)

        vehicle.add(steering, inputs=['angle'])
        vehicle.add(throttle, inputs=['throttle'])


    elif cfg.DRIVE_TRAIN_TYPE == "DC_STEER_THROTTLE":
        from donkeycar.parts.actuator import
Mini_HBridge_DC_Motor_PWM

        steering = Mini_HBridge_DC_Motor_PWM(cfg.HBRIDGE_PIN_LEFT,

cfg.HBRIDGE_PIN_RIGHT)
        throttle = Mini_HBridge_DC_Motor_PWM(cfg.HBRIDGE_PIN_FWD,
                                             cfg.HBRIDGE_PIN_BWD)

        vehicle.add(steering, inputs=['angle'])
        vehicle.add(throttle, inputs=['throttle'])


    elif cfg.DRIVE_TRAIN_TYPE == "DC_TWO_WHEEL":
        from donkeycar.parts.actuator import
TwoWheelSteeringThrottle, \
            Mini_HBridge_DC_Motor_PWM

        left_motor =
Mini_HBridge_DC_Motor_PWM(cfg.HBRIDGE_PIN_LEFT_FWD,

cfg.HBRIDGE_PIN_LEFT_BWD)
        right_motor =
Mini_HBridge_DC_Motor_PWM(cfg.HBRIDGE_PIN_RIGHT_FWD,

cfg.HBRIDGE_PIN_RIGHT_BWD)
        two_wheel_control = TwoWheelSteeringThrottle()

        vehicle.add(two_wheel_control,
```

```
                    inputs=['throttle', 'angle'],
                    outputs=['left_motor_speed',
'right_motor_speed'])

        vehicle.add(left_motor, inputs=['left_motor_speed'])
        vehicle.add(right_motor, inputs=['right_motor_speed'])

    elif cfg.DRIVE_TRAIN_TYPE == "SERVO_HBRIDGE_PWM":
        from donkeycar.parts.actuator import ServoBlaster,
PWMSteering
        steering_controller = ServoBlaster(cfg.STEERING_CHANNEL)
# really pin
        # PWM pulse values should be in the range of 100 to 200
        assert (cfg.STEERING_LEFT_PWM <= 200)
        assert (cfg.STEERING_RIGHT_PWM <= 200)
        steering = PWMSteering(controller=steering_controller,
                               left_pulse=cfg.STEERING_LEFT_PWM,
                               right_pulse=cfg.STEERING_RIGHT_PWM)

        from donkeycar.parts.actuator import
Mini_HBridge_DC_Motor_PWM
        motor = Mini_HBridge_DC_Motor_PWM(cfg.HBRIDGE_PIN_FWD,
                                          cfg.HBRIDGE_PIN_BWD)

        vehicle.add(steering, inputs=['angle'])
        vehicle.add(motor, inputs=["throttle"])

    # add tub to save data

    inputs = ['cam/image_array',
              'user/angle', 'user/throttle',
              'user/mode']

    types = ['image_array',
             'float', 'float',
             'str']

    if cfg.TRAIN_BEHAVIORS:
        inputs += ['behavior/state', 'behavior/label',
                   "behavior/one_hot_state_array"]
        types += ['int', 'str', 'vector']

    if cfg.HAVE_IMU:
        inputs += ['imu/acl_x', 'imu/acl_y', 'imu/acl_z',
                   'imu/gyr_x', 'imu/gyr_y', 'imu/gyr_z']

        types += ['float', 'float', 'float',
                  'float', 'float', 'float']

    if cfg.RECORD_DURING_AI:
        inputs += ['pilot/angle', 'pilot/throttle']
```

```
        types += ['float', 'float']

    th = TubHandler(path=cfg.DATA_PATH)
    tub = th.new_tub_writer(inputs=inputs, types=types,
user_meta=meta)
    vehicle.add(tub, inputs=inputs, outputs=["tub/num_records"],
                run_condition='recording')

    if type(ctr) is LocalWebController:
        print("You can now go to <your pi ip address>:8887 to
drive your car.")
    elif isinstance(ctr, JoystickController):
        print("You can now move your joystick to drive your car.")
        # tell the controller about the tub
        ctr.set_tub(tub)

        if cfg.BUTTON_PRESS_NEW_TUB:
            def new_tub_dir():
                vehicle.parts.pop()
                tub = th.new_tub_writer(inputs=inputs,
types=types,
                                        user_meta=meta)
                vehicle.add(tub, inputs=inputs,
outputs=["tub/num_records"],
                            run_condition='recording')
                ctr.set_tub(tub)

            ctr.set_button_down_trigger('cross', new_tub_dir)
        ctr.print_controls()

    # run the vehicle for 20 seconds
    vehicle.start(rate_hz=cfg.DRIVE_LOOP_HZ,
                  max_loop_count=cfg.MAX_LOOPS)


if __name__ == '__main__':
    args = docopt(__doc__)
    cfg = dk.load_config()

    if args['drive']:
        model_type = args['--type']
        camera_type = args['--camera']
        drive(cfg, model_path=args['--model'],
use_joystick=args['--js'],
              model_type=model_type, camera_type=camera_type,
              meta=args['--meta'])

    if args['train']:
        from train import multi_train, preprocessFileList

        tub = args['--tub']
```

```
        model = args['--model']
        transfer = args['--transfer']
        model_type = args['--type']
        continuous = args['--continuous']
        aug = args['--aug']

        dirs = preprocessFileList(args['--file'])
        if tub is not None:
            tub_paths = [os.path.expanduser(n) for n in
tub.split(',')]
            dirs.extend(tub_paths)

        multi_train(cfg, dirs, model, transfer, model_type,
continuous, aug)
```