РОБОТОТЕХНІЧНА СИСТЕМА ДЛЯ СЛІДУВАННЯ ЗА ОБ’ЄКТОМ

Поліський університет

ЗМІСТ

ВСТУП 3

1. ТЕХНОЛОГІЇ У СФЕРІ КОМП'ЮТЕРНОГО ЗОРУ ТА РОБОТОТЕХНІКИ .. 4

2. РОЗРОБЛЕННЯ РОБОТОТЕХНІЧНОЇ СИСТЕМИ ДЛЯ СЛІДУВАННЯ ЗА ОБ'ЄКТОМ 6

2.1 Програмна реалізація відстеження заданого об'єкта, виявлення кольору та обличчя 6

2.2 Реалізація робототехнічної системи для слідування за об'єктом 10

ВИСНОВКИ 14

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 15

ДОДАТКИ 17

ВСТУП

В багатьох сферах діяльності людини попит на автоматизовані та автономні системи постійно зростає. Залучення робототехнічних систем у промисловість, медицину та оборонний сектор сприяє їх стабільності та розвитку. На початку XXI століття робототехнічні системи обмежувались датчиками відстані, руху та температури. Завдяки розвитку таких технологій як комп'ютерний зір, штучний інтелект та машинне навчання, можливості робототехнічних системи стали значно ширшими.

Комп'ютерний зір – це область штучного інтелекту, яка вивчає можливості комп'ютерів бачити, розуміти та аналізувати зображення та відео. Комп'ютерний зір знаходить застосування у сфері медицини, автоматизації виробництва, транспорту, безпеки та інших. Однією з галузей, у якій технології комп'ютерного зору стали невід'ємною частиною стала робототехніка.

Метою даного дослідження є автоматизація процесу слідування за об'єктом за допомогою розробленої робототехнічної системи на основі технології комп'ютерного зору.

Предметом дослідження є методи, засоби та інструменти розроблення системи слідування за об'єктом на основі штучного інтелекту.

Об'єктом дослідження визначено процес розробки робототехнічної системи, яка включає робоплатформу та відеодатчик і застосовує технології комп'ютерного зору. Ідея розробки робототехнічної системи, полягає в тому, що робоплатформа, в залежності від положення та розміру об'єкта, за яким ведеться слідування за допомогою відеодатчика, може залучати відповідні потужності лівого, правого, або обох моторів аби втримувати заданий об'єкт в полі зору.

1. ТЕХНОЛОГІЇ У СФЕРІ КОМП’ЮТЕРНОГО ЗОРУ ТА РОБОТОТЕХНІКИ

Комп’ютерний зір – надзвичайно широка область, яка включає в себе багато різнопланових задач, таких яких сегментація, фільтрація, класифікація, реконструкція сцени, оцінка положення об’єкта, виявлення об’єктів, відеоспостереження та багато інших [1]. Робототехнічні системи, оснащені комп'ютерним зором, стають дедалі більш автономними, здатними виконувати складні завдання в динамічних середовищах, таких як самостійне водіння, доставка та дослідження.

Автопілот – технологія в який перетинаються комп’ютерний зір та штучний інтелект. Різноманітні датчики зчитують стан навколишнього середовища після чого ШІ розпізнає дорожню розмітку та знаки, розраховує оптимальну траєкторію руху враховуючи виявлені перешкоди та правила дорожнього руху. Для прикладу така технологія використовується в автомобілях Tesla і пропонує наступний спектр функцій [2]:

- Автоматичне керування – керує автомобілем без втручання водія;
- Автоматична зміна смуги – водій, використовуючи сигнали повороту, може інструктувати автопілот змінити смугу руху;
- Розгін та гальмування – автопілот може автоматично регулювати швидкість та підтримувати безпечну відстань від автомобілів навколо;
- Розумний виклик – перебуваючи на стоянці, власник може активувати дану функцію і автомобіль знайде до нього шлях.

Однією з галузей, де активно використовується комп’ютерний зір є аграрний сектор. За допомогою комп'ютерного зору можна аналізувати зображення посівів, зроблені дронами або іншими робототехнічними системами, щоб виявити ознаки проблем, таких як дефіцит поживних речовин, хвороби, шкідники та бур'яни шкідники та бур'яни. Це дозволяє фермерам оперативно вживати заходів, що може допомогти їм збільшити врожайність і зменшити втрати [3].

В основі створення більшості робототехнічних систем лежить використання одноплатних комп’ютерів та мікроконтролерів. Одними з провідних розробок в сфері одноплатних комп’ютерів є NVIDIA Jetson Nano та Raspberry Pi 5 (рис. 1.1).

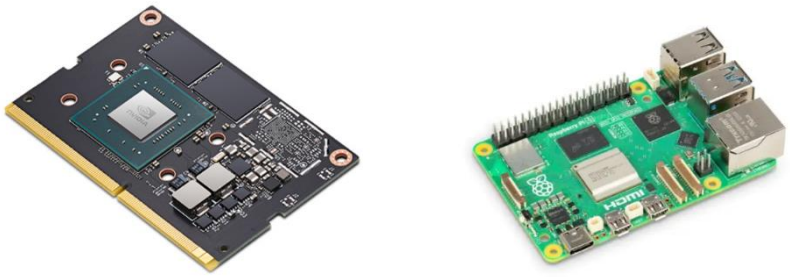

Рис. 1.1 – NVIDIA Jetson Nano та Raspberry Pi 5

NVIDIA Jetson Nano – компактний комп’ютер (110 мм x 79 мм x 30,21 мм), на базі чотириядерного процесора ARM Cortex-A57 MPCore та графічного процесора з 128-ядерною архітектурою NVIDIA Maxwell, призначений для запуску нейронних мереж [4].

Raspberry Pi 5 – одноплатний комп’ютер, що базується на процесорі ARM Cortex-A56 та графічному процесорі VideoCore VII GPU. Даний пристрій здатний працювати під управлінням повноцінної операційної системи, такої як Linux Ubuntu або Windows 10 IoT Core [5].

Реалізація комп’ютерного зору за допомогою одноплатного комп’ютера передбачає використання відеодатчика та розгорнення нейронної мережі. Розробки в сфері систем спостереження пропонують камери з вбудованим штучним інтелектом. Перевагою використання таких відеодатчиків є зменшення навантаження на комп’ютер або мікроконтролер. Прикладами таких камер є Pixy2 CMUcam5 та Huskylens (рис. 1.2).

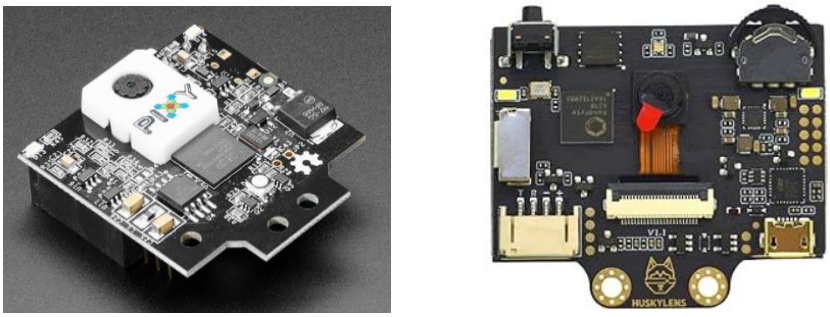

Рис. 1.2 – Pixy2 CMUcam5 та Huskylens

2. РОЗРОБЛЕННЯ РОБОТОТЕХНІЧНОЇ СИСТЕМИ ДЛЯ СЛІДУВАННЯ ЗА ОБ’ЄКТОМ

2.1 Програмна реалізація відстеження заданого об’єкта, виявлення кольору та обличчя

2.1.1 Вибір бібліотеки

Одними з найпоширеніших бібліотек комп’ютерного зору є OpenCV, SimpleCV, SimpleITK, Mahotas, Scikit-Image, PCL та VXL [6]. Для вирішення завдань програмного відстежування об’єкта на відео та виявлення кольору або обличчя на зображенні було використано OpenCV. Перевага даній бібліотеці була надана через простоту реалізації застосунків в реальному часі та наявність Python інтерфейсу.

OpenCV (Open Source Computer Vision Library) – це бібліотека комп’ютерного зору і машинного навчання з відкритим вихідним кодом. Бібліотека містить понад 2500 оптимізованих алгоритмів, які включають повний набір як класичних алгоритмів, так і алгоритмів комп’ютерного зору і машинного навчання. Дана бібліотека має інтерфейси C++, C, Python та Java. OpenCV розроблена з акцентом на додатках реального часу [7].

2.1.2 Відстеження заданого об’єкта на зображенні

OpenCV пропонує широкий спектр алгоритмів для відстеження об’єктів. Найвживанішими є CSRT (Channel State Representative Tracking), KCF (Kernel Correlation Filters), TLD (Tracking, Learning, Detection), DeepSORT (Simple Online and Realtime Tracking with a Deep Association Metric), SiamRPN++ (Siamese Region Proposal Network) [8]. Орієнтуючись на простоту ініціалізації та швидкість роботу алгоритму використано CSRT. Даний алгоритм використовує канальне представлення зображення для ефективного відстеження об’єктів в реальному часі.

Програма являє собою таку послідовність операцій: зчитування заданого відеоканалу, представлення користувачу першого кадру, на якому він обирає

об’єкт (прямокутну зону), який буде відстежувати трекер, після чого відтворює наступні кадри заданого відеоканалу, виділяючи прямокутником вибраний об’єкт. Ключові рядки Python коду представлені на рис 2.1 (повний код в додатку А).

```
tracker = cv2.legacy.TrackerCSRT_create() #створити об'єкт трекера

v = cv2.VideoCapture(0) # захопити відео-канал з індексом 0
# v = cv2.VideoCapture(r'vid.mp4') # Аргументом також може бути готовий відеофайл

ret, frame = v.read() # v.read поверне True та об'єкт кадру, якщо операція пройшла успішно
frame = imutils.resize(frame, width = 600)
cv2.imshow('Frame', frame)
tracked_obj = cv2.selectROI('Frame', frame) #cv.selectROI повертає прямокутник, виділений користувачем
tracker.init(frame, tracked_obj)

while True:
     ret, frame = v.read()
     success, box = tracker.update(frame)
     if success:
          (x, y, w, h) = [int(a) for a in box]
          cv2.rectangle(frame ,(x, y),(x + w, y + h), (100, 255, 0), 2)
     cv2.imshow('Frame', frame)
```

Рис. 2.1 – Відслідковування заданого об’єкта

2.1.2 Виявлення обличчя на зображенні

Для виявлення обличь на Python було використано Face Recognition Library в основі якої лежить dlib (C++). Дана бібліотека пропонує функцію «face_locations» [9], яка на вхід приймає зображення, а повертає масив кортежів, що містять крайні точки захопленого обличчя. Ключові рядки Python коду представлені на рисунку 2.2 (повний код в додатку Б).

```
v = cv2.VideoCapture(0)

while True:
     ret, frame = v.read()

     face_location = face_locations(frame)

     for top, right, bottom, left in face_location:
          cv2.rectangle(frame, (left, top), (right, bottom), (0,0,255), 2)
```

Рис. 2.2 – Виявлення обличчя на зображенні

2.1.3 Виявлення заданого кольору на зображенні

Виявлення кольору реалізовано на основі функції OpenCV «inRange». На вхід дана функція приймає зображення і колір, а повертає чорно-білу маску (рис. 2.3), на якій всі інші кольори відсіюються (білі зони – збіг із заданим кольором). Якщо задати один колір, функція буде неефективною, відсіюючи відтінки заданого кольору. Основним завданням аналізу кольору є порахувати верхню та нижню границю кольору. Таку задачу легше виконати, перевівши колір з формату BGR (Blue, Green, Red) в HSV (Hue – відтінок, Saturation – насиченість, Value – значення) (рис. 2.4) [10].

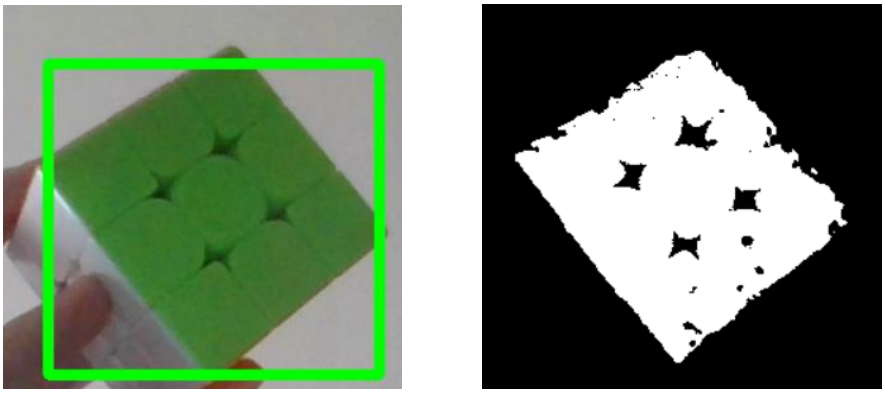

Рис. 2.3 – Захоплення зеленого кольору (оригінальне зображення та маска)

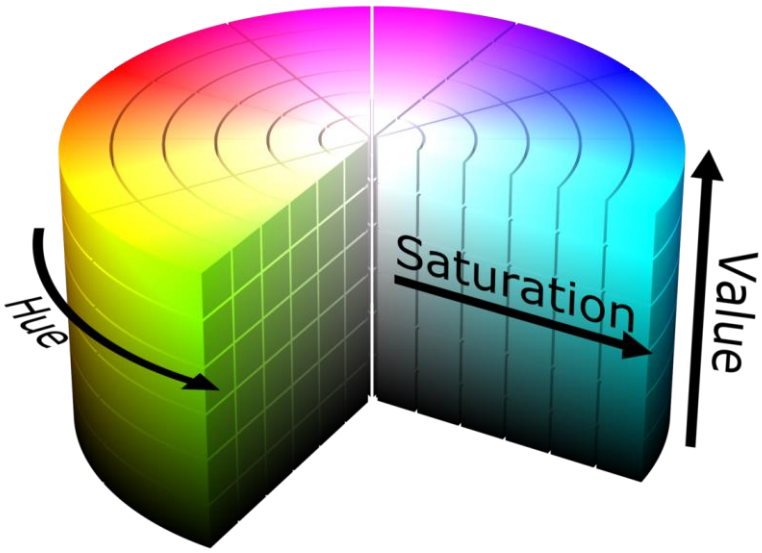

Рис.2.4 – HVS формат кольору

У Python-коді розрахунок границь буде виглядати наступним чином (рис. 2.5).

```
lowerLimit = np.array([hue - 10, 100, 100], dtype=np.uint8)
upperLimit = np.array([hue + 10, 255, 255], dtype=np.uint8)
#фрагмент функції get_limits
```

Рис 2.5 – Розрахунок границь кольору

В даному випадку «hue – 10» та «hue + 10» є незначними відхиленнями відтінку кольору. Наступним кроком є використання «inRange» та «Image.getbbox» бібліотеки PIL для отримання границь об’єкта. Ключові рядки Python коду представлені на рис. 2.6 (повний код в додатку В).

```
color_to_capture = (0, 255, 0)

video_capture = cv2.VideoCapture(0)

while True:
    ret, frame = video_capture.read()

    hsvImage = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    lowerLimit, upperLimit = get_limits(color=color_to_capture)

    mask = cv2.inRange(hsvImage, lowerLimit, upperLimit)

    mask_ = Image.fromarray(mask)

    bbox = mask_.getbbox()
    if bbox is not None:
        x1, y1, x2, y2 = bbox
        frame = cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 5)
```

Рис 2.6 – Виявлення заданого кольору

2.2 Реалізація робототехнічної системи для слідування за об’єктом

2.2.1 Компоненти робототехнічної системи

Основними компонентами робототехнічної системи є мікроконтролер Micro Bit, робоплатформа DFRobot Maqueen Plus та відеодатчик Huskylens. Програмна логіка, закладена розробником в мікроконтролер, може керувати робоплатформою та зчитувати дані, які надає відеодатчик.

Micro Bit v1.38 – плата для розробки, призначена для освоєння програмування мікроконтролерів (рис. 2.7). Підтримує програмування блоками, а також на Python та JavaScript. В якості середовища розробки можна використовувати MakeCode або Mind+ [11].

Рис 2.7 – Micro Bit

DFRobot Maqueen Plus v2.1 – робоплатформа, яка працює на основі контролера Micro Bit (рисунок 2.8). Платформа має 2 двигуни, ультразвуковий датчик відстані, ІЧ-приймач, датчик відстеження лінії [12].

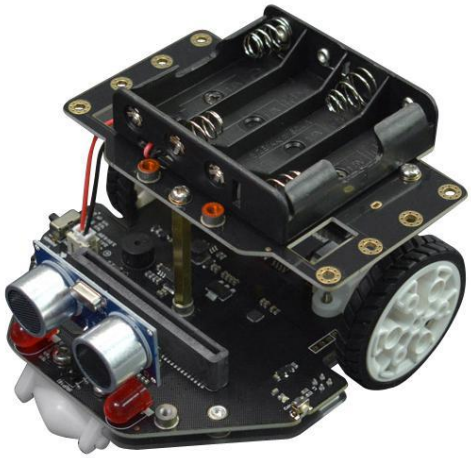

Рис 2.8 – DFRobot Maqueen Plus

Huskylens – відеодатчик зі штучним інтелектом, який може виконувати наступні функції: розпізнавання обличчя, відстеження об’єктів, розпізнавання об’єктів, відстеження лінії, виявлення кольорів та тегів (рисунок 2.9) [13]. HuskyLens працює з мікроконтролерами Arduino, Micro Bit, а також з одноплатними комп’ютерами Raspberry Pi, Orange Pi, Jetson Nano.

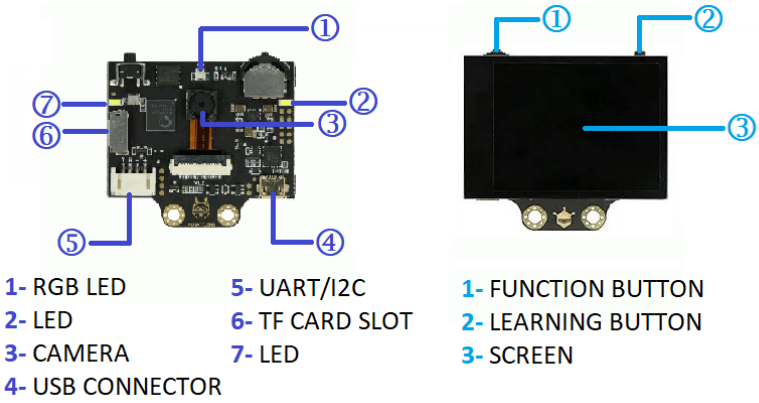

Рис. 2.9 – Huskylens

2.2.2 Програмування мікроконтролера

Суть розробленого алгоритму слідування робоплатформи за об’єктом наступна: якщо об’єкт перебуває у правій частині зображення, яке захоплює відеодатчик, то робоплатформа залучає лівий мотор, якщо ж у лівій частині зображення – правий мотор. Якщо розмір об’єкта на зображенні більший від базового, то платформа підключає два мотори для руху назад, якщо розмір менший – два мотори для руху вперед. Така логіка має сприяти втриманню заданого об’єкта в полі зору відеодатчика.

Обрані для побудови робототехнічної системи компоненти мають високу сумісність. Середовище розробки MakeCode [14], призначене для програмування мікроконтролера Micro Bit, має інтерфейси для роботи як з Maqueen Micro Bit, так і з Huskylens.

Програмний інтерфейс maqueenPlusV2 дає можливість взаємодіяти з моторами. Для реалізації функцій руху клас maqueenPlusV2 надає функції control_motor та control_motor_stop. Приклад виклику функції control_motor показано на рисунку 2.10.

```
def MoveForward():
    maqueenPlusV2.control_motor(maqueenPlusV2.MyEnumMotor.ALL_MOTOR,
        maqueenPlusV2.MyEnumDir.FORWARD,
        speed)
```

Рис. 2.10 – Рух вперед

Дисплей Huskylens має роздільну здатність 320 на 240 пікселів. За допомогою програмного інтерфейсу HuskyLens можна отримати стан захоплених об’єктів відеодатчиком. Захоплені об’єкти мають наступні атрибути: ID; x, y – горизонтальна та вертикальна координата в пікселях центру об’єкта; width, height – висота та ширина в пікселях об’єкта на зображенні, learned – чи вивчений даний об’єкт.

Функція відстеження передбачає постійний процес оновлення стану зображення. Клас basic пропонує функцію forever, яка аргументом приймає іншу функцію, що буде постійно викликатись (on_forever). Таким чином, на початку функції «on forever» необхідно отримати координату x та висоту захопленого об’єкта (рис. 2.11).

```
    x_box_position = huskylens.reade_box(1, Content1.X_CENTER)
    box_height = huskylens.reade_box(1, Content1.HEIGHT)
```

Рис 2.11 – Отримання атрибутів захопленого об’єкта

Маючи атрибути захопленого об’єкта (x_box_position, box_height), розмір дисплея (box_width, box_height) можна описати виклик відповідних функцій руху в залежності від виконаних умов (рис. 2.11) (загальний код програми в додатку Г).

```
    if box_height > box_height_default + 10:
        MoveBackward()
    elif box_height < box_height_default - 10 and box_height > -1:
        MoveForward()
    elif x_box_position > width_center + free_gap:
        MoveRight()
    elif width_center - free_gap > x_box_position and x_box_position > -1:
        MoveLeft()
    else: Stop()
```

Рис. 2.11 – Умови для виклику функцій руху

2.2.3 Тестування робототехнічної системи

Розроблена робототехнічна система працює відповідно до закладеного у мікроконтролер алгоритму (рис. 2.12). Розроблена програмна логіка не є досконалою і може бути допрацьована наступними функціями: динамічний розрахунок швидкості, з якою необхідно рухатись до об’єкта; пошук об’єкта, якщо він зник з поля зору.

Доступ до відеофайлу з демонстрацією процесу тестування робототехнічної системи розміщено на Google Drive [15].

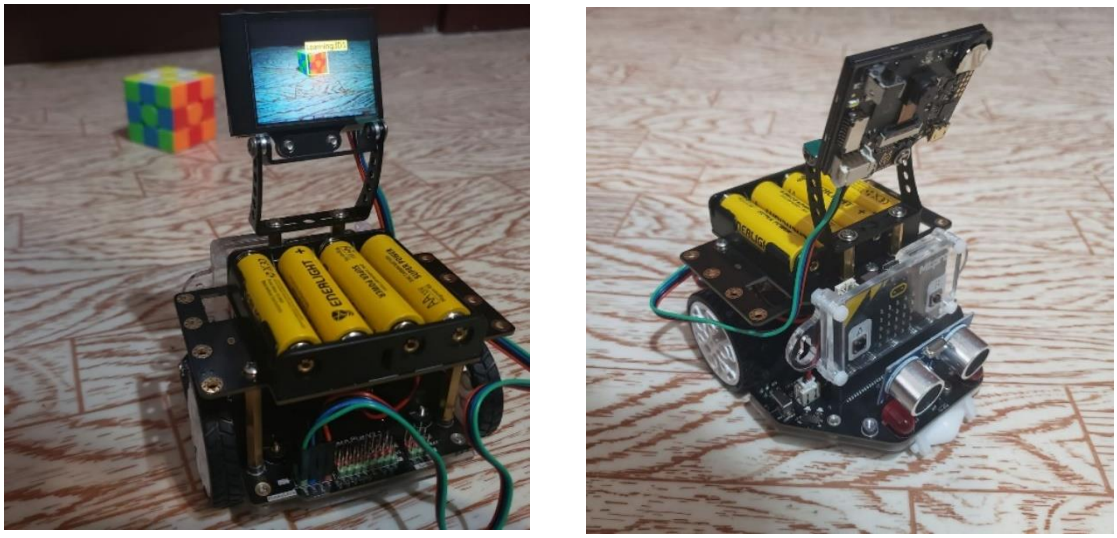

Рис. 2.12 – Робототехнічна система

ВИСНОВКИ

Результатом даного дослідження є програмна реалізація функцій відслідковування об’єкта, виявлення кольору та обличчя, а також реалізація робототехнічної системи, в основі якої лежить відеодатчик з ШІ Huskylens, плата Micro Bit та робоплатформа DFRobot Maqueen Plus. Використовуючи середовище розробки MakeCode, було програмно описано логіку взаємодії відеодатчика та робоплатформи. В даній роботі було продемонстровано простоту роботи з Micro Bit та Huskylens.

Розробка проектів на основі Micro Bit несе в собі в першу чергу навчальну функцію, але такі розробки можуть стати прототипами для більш серйозних рішень. DFRobot та Micro пропонують пропонують додаткові модулі, які можна використати в робототехнічних системах, на основі Maqueen Plus та Micro Bit (навантажувач, штовхач та інші).

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Р. М. Тимчишин, О. Є. Волков, О. Ю. Господарчук, Ю. П. Богачук. СУЧАСНІ ПІДХОДИ ДО РОЗВ'ЯЗАННЯ ЗАДАЧ КОМП'ЮТЕРНОГО ЗОРУ, Київ, 2018. 28 с.

2. Autopilot and Full Self-Driving Capability. URL: https://www.tesla.com/support/autopilot (дата звернення: 10.04.2024).

3. Лапи з інтелектом. URL: https://ifarming.ua/itehnologii/lapy-z-intelektom (дата звернення: 10.04.2024).

4. Jetson Nano. URL: https://www.nvidia.com/ru-ru/autonomous-machines/embedded-systems/jetson-nano/product-development/ (дата звернення: 14.04.2024).

5. Raspberry Pi. URL: https://www.raspberrypi.com/products/raspberry-pi-5/ (дата звернення: 14.04.2024).

6. Top 16 computer vision libraries . URL: https://www.superannotate.com/blog/computer-vision-libraries (дата звернення 11.04.2024).

7. К. О. Мартиненко. Порівняльний аналіз бібліотек комп'ютерного зору. Вінниця. 2016. 3 с.

8. OpenCV. Tracking API. URL: https://docs.opencv.org/3.4/d9/df8/group__tracking.html (дата звернення 10.04.2024).

9. face_recognition package. URL: https://face-recognition.readthedocs.io/en/latest/face_recognition.html (дата звернення 11.04.2024).

10. OpenCV. Thresholding Operation using inRange. URL: https://docs.opencv.org/3.4/da/d97/tutorial_threshold_inRange.html (дата звернення 12.04.2024).

11. 1.3x micro:bit revision URL: https://tech.microbit.org/hardware/1-3-revision/ (дата звернення 12.04.2024).

12. micro:Maqueen Plus V2. URL: https://www.dfrobot.com/product-2487.html (дата звернення 12.04.2024).
13. Easy-to use machine vision camera from DFRobot . URL: https://wiki.microblocks.fun/en/extension_libraries/huskylens (дата звернення 12.04.2024).
14. MakeCode. URL: https://makecode.microbit.org/ (дата звернення 12.04.2024).13.
15. Тестування. Google Drive. URL: https://drive.google.com/drive/folders/1o-awEfJjF7CgBUREyn3JMyPhiIkbyqzI?usp=sharing (дата звернення: 15.04.2024).

ДОДАТКИ

ДОДАТОК А

```
import cv2
import imutils

tracker = cv2.legacy.TrackerCSRT_create() #створити об'єкт трекера

v = cv2.VideoCapture(0) # захопити відео-канал з індексом 0
# v = cv2.VideoCapture(r'vid.mp4') # Аргументом також може бути готовий відеофайл

ret, frame = v.read() # v.read поверне для ret True та об'єкт кадру для frame, якщо
операція пройшла успішно
frame = imutils.resize(frame, width = 600)
cv2.imshow('Frame', frame)
tracked_obj = cv2.selectROI('Frame', frame) #cv.selectROI повертає прямокутник,
виділений користувачем
tracker.init(frame, tracked_obj)

while True:
    ret, frame = v.read()
    if not ret:
        break
    frame = imutils.resize(frame, width = 600)
    success, box = tracker.update(frame)
    if success:
        (x, y, w, h) = [int(a) for a in box]
        cv2.rectangle(frame ,(x, y),(x + w, y + h), (100, 255, 0), 2)

    if cv2.waitKey(5) & 0xFF == ord('q'):
        break

    cv2.imshow('Frame', frame)

v.release()
cv2.destroyAllWindows()
```

Рис А.1 – Відслідковування об’єкта

ДОДАТОК Б

```python
import cv2
from face_recognition import face_locations

v = cv2.VideoCapture(0)

while True:
    ret, frame = v.read()

    face_location = face_locations(frame)

    for top, right, bottom, left in face_location:
        cv2.rectangle(frame, (left, top), (right, bottom), (0,0,255), 2)

    cv2.imshow("Frame", frame)

    if cv2.waitKey(5) & 0xFF == ord('q'):
        break

v.release()
cv2.destroyAllWindows()
```

Рис. Б.1 – Розпізнавання обличчя

ДОДАТОК В

```python
import cv2
from PIL import Image
import numpy as np

def get_limits(color):
    c = np.uint8([[color]])
    hsvC = cv2.cvtColor(c, cv2.COLOR_BGR2HSV)

    hue = hsvC[0][0][0]

    if hue >= 165:
        lowerLimit = np.array([hue - 10, 100, 100], dtype=np.uint8)
        upperLimit = np.array([180, 255, 255], dtype=np.uint8)
    elif hue <= 15:
        lowerLimit = np.array([0, 100, 100], dtype=np.uint8)
        upperLimit = np.array([hue + 10, 255, 255], dtype=np.uint8)
    else:
        lowerLimit = np.array([hue - 10, 100, 100], dtype=np.uint8)
        upperLimit = np.array([hue + 10, 255, 255], dtype=np.uint8)

    return lowerLimit, upperLimit

color_to_capture = (255, 0, 0)
video_capture = cv2.VideoCapture(0)

while True:
    ret, frame = video_capture.read()

    hsvImage = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    lowerLimit, upperLimit = get_limits(color=color_to_capture)

    mask = cv2.inRange(hsvImage, lowerLimit, upperLimit)
    cv2.imshow("Mask", mask)

    mask_ = Image.fromarray(mask)

    bbox = mask_.getbbox()
    if bbox is not None:
        x1, y1, x2, y2 = bbox
        frame = cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 5)
    cv2.imshow("Frame", frame)

    if cv2.waitKey(5) & 0xFF == ord('q'):
        break

video_capture.release()
cv2.destroyAllWindows()
```

Рис. В.1 – Виявлення кольору

ДОДАТОК Г

```
y_box_position = 0
x_box_position = 0
box_height_default = 50
speed = 0
screen_width = 320
screen_hight = 240

width_center = screen_width / 2
hight_center = screen_hight / 2

free_gap = 30
speed = 55

def MoveRight():
    maqueenPlusV2.control_motor(maqueenPlusV2.MyEnumMotor.LEFT_MOTOR,
        maqueenPlusV2.MyEnumDir.FORWARD,
        speed)
    maqueenPlusV2.control_motor_stop(maqueenPlusV2.MyEnumMotor.RIGHT_MOTOR)

def MoveLeft():
    maqueenPlusV2.control_motor(maqueenPlusV2.MyEnumMotor.RIGHT_MOTOR,
        maqueenPlusV2.MyEnumDir.FORWARD,
        speed)
    maqueenPlusV2.control_motor_stop(maqueenPlusV2.MyEnumMotor.LEFT_MOTOR)

def MoveBackward():
    maqueenPlusV2.control_motor(maqueenPlusV2.MyEnumMotor.ALL_MOTOR,
        maqueenPlusV2.MyEnumDir.BACKWARD,
        speed)

def MoveForward():
    maqueenPlusV2.control_motor(maqueenPlusV2.MyEnumMotor.ALL_MOTOR,
        maqueenPlusV2.MyEnumDir.FORWARD,
        speed)

def Stop():
```

```
    maqueenPlusV2.control_motor_stop(maqueenPlusV2.MyEnumMotor.ALL_MOTOR)
def on_forever():
    huskylens.request()

    x_box_position = huskylens.reade_box(1, Content1.X_CENTER)
    box_height = huskylens.reade_box(1, Content1.HEIGHT)

    if box_height > box_height_default - 10:
        MoveBackward()
    elif box_height < box_height_default - 10 and box_height > -1:
        MoveForward()
    elif x_box_position > width_center + free_gap:
        MoveRight()
    elif width_center - free_gap > x_box_position and x_box_position > -1:
        MoveLeft()
    else:
        Stop()

maqueenPlusV2.i2c_init()
huskylens.init_i2c()
huskylens.init_mode(protocolAlgorithm.ALGORITHM_OBJECT_TRACKING)

basic.forever(on_forever)
```

Рис. Г.1 – Логіка мікроконтролера