PROJECT INFORMATION

Project Topic: Physical Implementation of Reinforcement Learning and Path-Planning Algorithms on an RC Car
Team Members: Alexander Facey, Carson Page
Supervisor: Hazim Alzorgan

PROJECT DESCRIPTION

This project focuses on applying artificial intelligence (AI) algorithms, such as Q-Learning and A* Path Planning, to a real-world robotic system. The main objective is to program an RC car equipped with a Raspberry Pi and sensors to autonomously navigate an obstacle course. By moving from a simulated environment to physical hardware, this project bridges theoretical AI knowledge with practical engineering skills.
The motivation behind this work is to gain hands-on experience in implementing reinforcement learning algorithms on embedded systems and to understand the challenges of real-world robotic control. The primary resources used include a Raspberry Pi, ultrasonic sensors, motor driver modules, and Python programming.

TOOLS AND METHODS

- **Hardware**: Raspberry Pi, RC car framework, ultrasonic sensors, motor driver (L298N), power supply
- **Software**: Python, Anaconda 3 + Anaconda Navigator, Relevant Python libraries, self-made application for testing object-tracking implementation.
- **Methods**:
    - Implementation of AI path-planning algorithms (Q-Learning, A*)
    - Object tracking via Python script, dependent on OpenCV and MobileNet-SSD
    - Sensor data acquisition for obstacle detection and distance measurement
    - Motor control and steering via GPIO pins
    - Testing and turning of control parameters for accurate navigation

PROJECT ACTIVITIES (TASK 1)

- Designed and implemented a MobileNet-SSD object detection system using OpenCV and Python.
- Configured and bundled Python scripts into a standalone .exe using PyInstaller, including handling external model files (.caffemodel and .prototxt).
- Implemented a workaround for PyInstaller file paths, ensuring the executable can load model files reliably in --onefile mode.
- Developed a live webcam feed pipeline with real-time object detection, including bounding box visualization.
- Tested the system thoroughly on a desktop environment for accuracy and performance.

- Documented the workflow for bundling and distributing the application, including cleaning previous builds and configuring PyInstaller flags.

RESULTS

- Successfully created a working executable that performs real-time object detection on webcam input.
- Demonstrated reliable detection of objects with bounding boxes drawn correctly at >50% confidence.
- Verified that the application handles PyInstaller's _MEIPASS folder correctly using a temporary safe folder workaround.
- Achieved cross-platform readiness in the sense that the script can be bundled and run without Python installed on the target machine (Windows .exe).
- Logged and monitored program execution through redirected stdout/stderr (log.txt) for debugging purposes.

CHALLENGES AND ISSUES

- OpenCV DNN requires model files to be accessible via filesystem, creating complications when bundling into a --onefile executable.
- PyInstaller --onefile mode can cause path resolution issues, requiring a workaround to copy files to a temporary folder.
- Real-time object detection performance may be limited by hardware, especially when transitioning to Raspberry Pi.
- Handling camera access and frame rate consistently for real-time detection was non-trivial.
- Debugging in PyInstaller's environment is harder because print statements don't show in GUI mode, necessitating logging to files.

PLANS FOR NEXT STEPS

Short Term (Next 2 Weeks)

- Port the object detection system to a Raspberry Pi and test with Pi Camera or USB webcam.
- Verify that the PyInstaller workaround and model loading works on the Raspberry Pi environment.
- Begin frame rate and inference optimization for real-time detection on Raspberry Pi.

Medium Term (Weeks 3–4)

- Integrate object detection with RC car control, enabling the car to respond to obstacles.
- Implement basic autonomous navigation algorithms, such as A* pathfinding, for simple obstacle courses.
- Conduct small-scale testing of car navigation in a controlled maze environment.

Long Term (Weeks 5–6)

- Refine navigation system with reinforcement learning (Q-learning) for more complex obstacle avoidance.
- Optimize detection thresholds, frame rate, and model performance for robust real-world operation.
- Conduct full system tests, evaluating the car's ability to navigate autonomously through a complete maze/obstacle course.