AUTONOMOUS EMBEDDED CONTROL

RECRUITMENT TASK

Cairo University Eco-Racing Team

2024

Cairo University Eco-Racing team applies every year for the Shell Eco-Racing 2023 Marathon competition, so, it aims to create an autonomous car, by creating an embedded control system that controls the ICE (Internal combustion engine) vehicle, and this system contains roughly the following:

1. One DC motor to control the speed of wheels.
2. One stepper motor to move the wheels angle right and left.
3. A decision-maker unit (out of scope) which sends its decisions to our micro-controller.
4. A dashboard to show out the current state and the car parameters, that are always up to date.

Kindly take into account the subsequent instructions

Completion of the entire task is not mandatory for delivery; perform to the extent of your capabilities. While online resources are permissible, it is mandatory that you understand the content you are including into your project. Employing code without a comprehensive understanding poses a significant concern.

Communicating and Controlling an Autonomous Car Prototype Task

Objective

Your ability to write efficient, clear, and reliable embedded-C code as well as your capacity to integrate multiple blocks that cooperate harmoniously into a single design will be the focus of this task.

Task Overview

Design a communication and control system that interfaces with an autonomous car prototype. The system should allow a PC to send control commands to the car and receive real-time data updates from the car. The goal is to control the car's speed, direction (forward or backward), and wheel angle through a UART communication interface.

Statement

Implement a simplified version of said system, that receives a data frame, through a UART-PC communication.

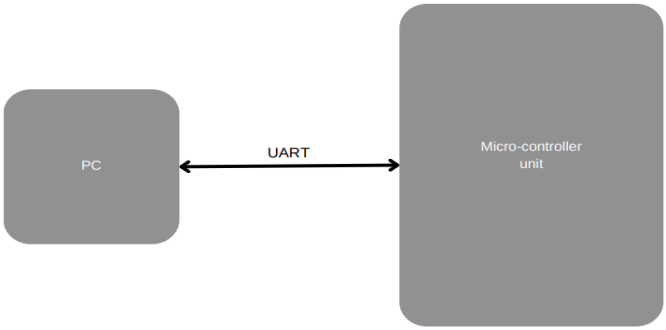


Figure 1

Your PC endpoint should be modeled as a virtual terminal that takes the communication frame and maps its values into speed, direction (forward or backward) and orientation of the autonomous car.

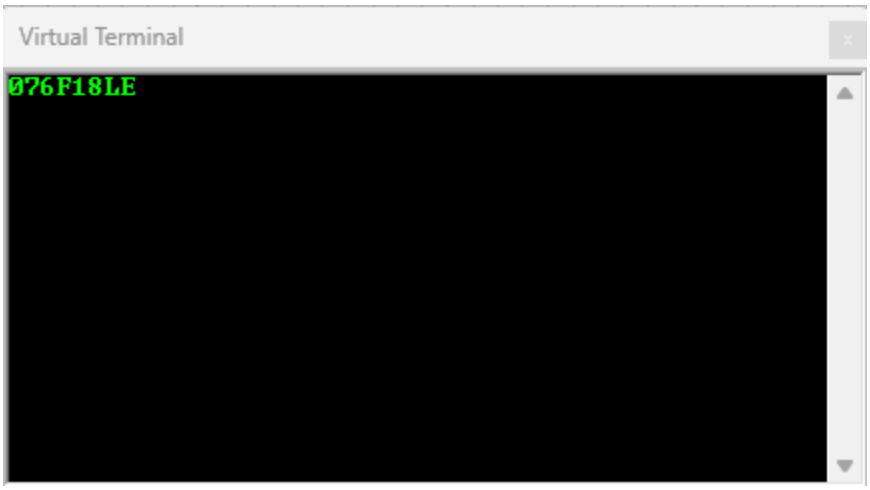


Figure 2

The UART-PC communication frame should be as following:

XXXAYYBE

1. XXX -> for the speed of the motor, controlling the speed of the wheels. (0 -> 100)
2. A -> for the direction of the motor, controlling the speed of the wheels. (F or B)
3. YY -> for the angle of the second motor, controlling the angle of the wheels (0 -> 45)

4. B -> for the direction of the second motor. (Right or left) (with the north of the car being the reference).
5. E -> is an indicator of the end of the frame.

If the user didn't write E at the end of the frame or violated the frame specified length (8 bytes) then it's an un-valid frame and it should be ignored

Examples:

1. 076F18LE -> means the car should change its speed to 76% of the maximum speed, the direction of movement is forward, the wheels should tilt to the left from its axis by 18 degrees.
2. 76F18RE -> This frame looks fine but it violated the standard length we agreed on.
3. F100R09E -> This frame is wrong because it's not matching our standard.
4. 076F18R -> This frame is very correct, but unfortunately, When E is not included, we should consider the frame as noise only should be ignored.

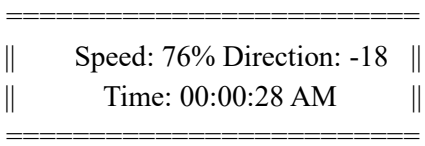
General Assumptions:

1. In case of a DC motor, assume: (ACW -> forward movement, CW -> backward movement).
2. In case of stepper motor, assume: (each step = one degree for simplicity).

Required:

You should enter the speed, direction and orientation, from your PC/laptop keyboard through the virtual terminal by using the above-mentioned communication frame, and the motors should adjust to the entered speeds and directions. In order to implement a simplified version of the dashboard, show on LCD the speed percentage and the angle of wheels (+ve angle means at the right of the car's north, -ve angle means at the left of the car's north), and the time, the time should start from 00:00:00 AM (following the hour, minutes, seconds format).

Example:



Deliverables

Zip file or GitHub repo (preferred) containing the following:

1. A folder named “Code”, containing the following:
    a. All .c and .h files only. (With no compilation/project/IDE files).
    b. One text file for any assumptions you made. (optional).

2. A file outside the folder named “Video”, that contains a screen recording showing the system working and running on Proteus, while using the virtual terminal feature and testing the system using different data frames. (And explain your work by your voice). [Maximum 15 min, and minimum 5 min]
3. In case you implemented the drivers by yourself (It’s not required but preferable), Create the following folders:
    a. MCAL folder
        i. Inc folder
            1. All .h files.
        ii. All .c files
    b. HAL folder
        i. Inc folder
            1. All .h files.
        ii. All .c files
    c. Services folder
        i. Inc folder
            1. All .h files.
        ii. All .c files
    d. App folder
        i. Inc folder
            1. All .h files.
        ii. All .c files
    e. Libraries folder
        i. Inc folder
            1. All .h files.
        ii. All .c files