0.1 FreeRTOS

In the design of an autonomous vehicle, factors such as precise timing, responsiveness, and predictability are crucial for the software. If the software fails to process inputs from sensors quickly enough, it could lead to an accident. These critical aspects can be addressed by utilizing a real-time operating system (RTOS) like FreeRTOS. With RTOS task scheduling, essential actions such as halting movement when an obstacle blocks the way are executed within specified time constraints. This ensures that the action of stopping the vehicle is not obstructed by other tasks, thus preventing accidents.

0.1.1 The Espressif flavor

One of the reasons we chose the ESP32 MCU was to leverage its capabilities with FreeRTOS. The ESP32 uses a custom flavor of FreeRTOS made by Espressif. A key difference for this custom flavor is its support for Dual-Core processors. This means that tasks can be distributed across two cores, as opposed to the original one core support in FreeRTOS. Another advantage is the presence of a HAL, ensuring that MCU changes in the ESP ecosystem are durable without the need for significant code maneuvers.

Throughout this section some FreeRTOS native function are used. These functions include:

Tasks:

```
1         // Task to be created.Pointer that will be used as the parameter for the task
      being
2         void vTaskCode( void * pvParameters )
3         {
4             for( ;; )
5             {
6                 // Task code goes here.
7             }
8         }
9
```

Listing 1: FreeRTOS task creation

There are a few things to notice that differ from a function in regular C language. Firstly, the function name has a prefix of 'v,' where 'v' stands for void, indicating that the function does not return anything. Another notable aspect is the use of a 'for' loop. In this context, a task is designed to run continuous operations, hence the inclusion of the forever loop. It's important to highlight that, unlike native C, the forever loop won't block other operations. The 'pv' prefix on the function parameter indicates that the return type is a pointer to void. This parameter is used to continuously pass values to the task throughout the program

Creating task:

```
1         xTaskCreate( vTaskCode, "NAME", STACK_SIZE, &ucParameterToPass, tskIDLE_PRIORITY
      , &xHandle );
2
```

Listing 2: Creating a task

When creating a task, several parameters need to be fulfilled. These include:

- **vTestCode:** The task function to call, as seen in listings 1.
- **NAME:** A given name for the task..
- **STACK_SIZE:** The size of the task stack, specified in bytes.
- **ucParameterToPass:** A pointer to pass parameters to the task.
- **tskIDLE_PRIORITY:** The priority at which tasks are run.
- **xHandle:** The task handle by which the task can be referred to.

The 'x' prefix in the function name indicates that the function returns a value. In this case, it signifies the return of the handle, which is used to reference to the task later on.

0.1.2 Development enviroment

A familiar development environment is crucial for ensuring quality and productivity in coding. Learning the ins and outs of a new Integrated Development Environment (IDE) can be time-consuming. Therefore, it is a significant advantage that Espressif has integrated their ESP-IDF into the VS Code IDE. VS Code is an IDE familiar to everyone participating in the project, and its extensive array of tools further facilitates development.