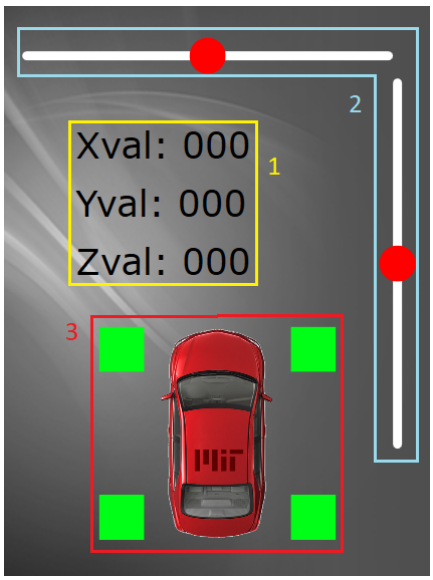
The application uses the touchGFX framework and the gyroscope inside the microcontroller. The interface aims to illustrate the forces to which the four support points of a car are subjected. Obviously the model is very simplified, having only one gyroscope and the microcontroller being a completely rigid body, so it is not possible to consider the damping effects of wheels and suspensions.

The graphic part consists of 3 main elements:



1. Gyroscope text indicator. The values are indicated in degrees/second [°/s].
2. Gyroscope visual indicator. The full scale is set to ±180°/s.
3. Application of measured values of the gyroscope. The squares at the corners can take 4 colors:
    - Green if there are no significant oscillations (less than 30 ° / s);
    - Blue if it detects an inverse oscillation with respect to the rotation of the device;
    - Yellow if it detects a significant oscillation on one or more axes (between 30 ° / s - on single axis - and 180 ° / s - considering the sum of x and y axes);
    - Red if it detects a greater oscillation (sum of oscillations on x and y axes greater than 180 ° / s)

    Since the microcontroller is a rigid body, we consider the four points specular between them (i.e. if there is at least a significant oscillation (yellow or red) on one point, on the opposite one there is an inverse oscillation (blue)).

The application consists of three tasks:

1. `Task1` is responsible for managing the display and the graphical interface. It has minimal priority, as it only needs to update the display when the CPU is idle.
2. `Task2` manages the gyro data, then gets the values and queues the updated data. This way, when `Task1` runs, it updates the graphical interface accordingly. The graphical indicator (2) is updated at each reading, the textual indicator (1) every four readings.
3. `Task3` computes some useful values for updating the graphics application (3) and puts the calculated values in the queue.

Since no method for exchanging messages between tasks is implemented in ERIKA3, I defined the `MessageQueueue` class to contain the updated data; this is not a real message queue, but only implements the functionalities that were needed for the purpose of this application.

`Task2` and `Task3` perform some operations on the same variables (the first read and write, while the second read only) and, to avoid inconsistencies in reading data, a resource lock is used.

An interrupt for the `USER` button is also implemented. Through this it is possible to invert the representation of the x and y axes of the gyroscope indicator (2).

NOTE: I had to include the `.c` file that define the gyroscope component in `main.cpp`, as by including its header file there were an "undefined reference" error during compilation, possibly due to linking problems. Not being able to change the compilation command, I therefore had to opt for this choice.