Project 1: Real-Time Services on Linux GPOS

Introduction

The aim of this project is to apply the Linux Real-Time Services and the Real-Time Model to the the development of a real-life inspired real-time application.

Following the typical structure of embedded software, the project encompasses a set of cooperating tasks, involving synchronization, shared resources, access to a real-time database, etc.

Preparation

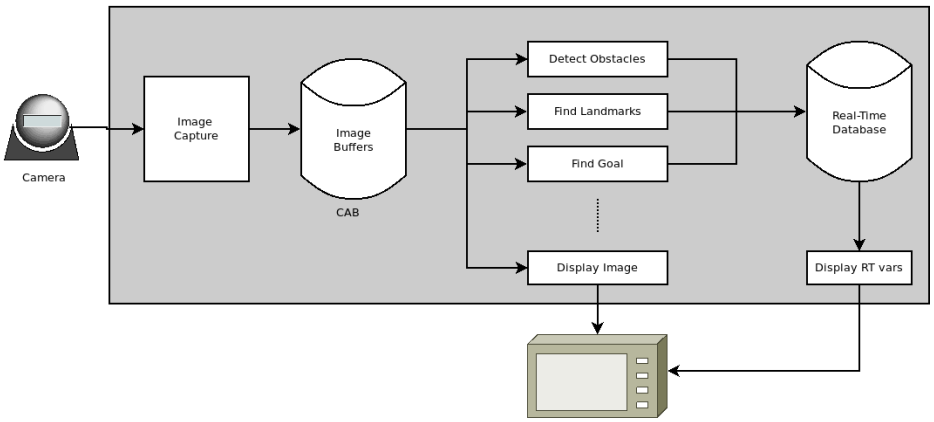
Students should first get comfortable with Linux real-time services addressed in the lab classes, including implementation of tasks as Linux threads, real-time scheduling policies, synchronization primitives, access to shared resources, etc.

Furthermore the students are required to acquire some domain-specific knowledge, in the case related to image acquisition and processing. The source code provided includes information and links that should be studied in detail.

General description

The application to be implemented is representative of the vision subsystem of a mobile robot, with substantial simplifications made to make it compatible with the effort set for the project: two students per group, 6 ECTS, duration of four weeks.

The system architecture is as follows:



Live images are captured from a camera by a dedicated camera task. The activation of this task is autonomous (sporadic task), as it originates from the camera. After some optional processing (e.g. denoising) images are placed in a buffer (CAB type) to be processed by dedicated tasks.

These dedicated tasks perform, as the name implies, specific functions. Examples include:

- Detection of an object on given areas (obstacle detection).
- Detection of the position of objects such as people, other robots, etc.
- Detection of landmarks that allow the robot to localize itself.

The activation of these tasks is explicit and configurable. E.g. object detection should be executed in all image frames (as it involves safety issues), while localization can be executed, if needed, only every 30 frames, as the robot does not move instantly.

Data generated by theses tasks is placed in an Real-Time DataBase (RTDB). A task prints the contents of the RTDB with a user-defined periodicity.

Moreover, the images may also be displayed, something useful for debugging purposes.

(Some) Requirements

- It must be implemented a CAB buffer system, or equivalent. Note that in this kind of applications there are tasks that can take a very long time (e.g. finding objects) while others are very quick and urgent (e.g. detect obstacles). For this reason it is important that tasks can keep a buffer for arbitrary long time durations, so CABs are a good solution.
- Task priorities are left to you, but must be properly justified
- You can choose the frequency of activation of the tasks, noting that the system must be schedulable. This, of course, depends on the capacity of your computer and on the complexity of the algorithms you implement. It is not required to attain a given rate for any task, but the rate selected by you should be properly justified in the report.
- Image processing is a very challenging topic. It is not expected that students implement complex algorithms. Simple (or even simplistic) approaches are fine, as this topic is out of the scope of the course. Use of libraries such as OpenCV should be avoided. It is important that you understand the task code to reason about the execution time of tasks, and the use of libraries such as OpenCV (that implement very powerful but complex algorithms, hidden under a lot of abstractions and modules) makes the analysis of the code very (or better, extremely) difficult.

Note that this specification is very open and allows for different approaches (as usual in real life). You are welcome to discuss with me any aspects of the work that raise doubts.

You only need to implement 3 or 4 processing tasks. It is suggested e.g.:

- Obstacle detection on a given area (assuming the floor is of one color and obstacles are of a different color, for simplicity)
- Detecting an object of a given color (see sample code)
- Detecting a pole with two colors (commonly used for landmarks)

You are advised to leave the image acquisition module as a separate process, and implement everything else in another process. This way, the code becomes independent of the image acquisition library used (ffmpeg in the case, but there are many others), provided that the output format is respected (Packed RGB888+filler in the case).

Deliverables

- A report, up to 8 pages, pdf format, submitted via eLearning, with:
  - Cover page: identification of the course and assignment, identification of the group members (first name, last name and ID number))
  - Description of the software architecture:
    - Which tasks were implemented and their role
    - Which data structures were used and how they are accessed
    - Which synchronization methods have been used
  - Discussion of the task priorities, activation rates and system schedulability
    - A brief description (diagrams and text) of the **task execution sequence and relevant events** when the system is working normally is welcome.
    - The diagram should contain the start/finish times of tasks, operations on the buffers, signals exchanged, etc. A reader should be able to understand which task does what and when.
  - Tests, results and analysis
- Zip file with the full project folder and files.
  - **The project code MUST have a Makefile and be buildable just by typing make.**
  - The use of any libraries besides ffmpeg and SDL and V4Linux must be specified in the report.

Bibliography:

- For CABs, chapter “10.6.1 CYCLIC ASYNCHRONOUS BUFFERS” of the reference book
  - Giorgio Buttazzo (2011). HARD REAL-TIME COMPUTING SYSTEMS: Predictable Scheduling Algorithms and Applications, Third Edition, Springer, 2011
- ffmpeg base documentation (online)

- https://ffmpeg.org/doxygen/trunk/index.html

- Additional ffmpeg documentation (online)
  - https://github.com/leandromoreira/ffmpeg-libav-tutorial
  - https://github.com/leixiaohua1020/simplest_ffmpeg_device
  - https://stackoverflow.com/questions/66384258/sws-scale-yuv-to-rgb-conversion
  - http://www.dranger.com/ffmpeg/ffmpeg.html
- SDL2 (online)
  - https://benedicthenshaw.com/soft_render_sdl2.html
  - https://stackoverflow.com/questions/69259974/pixel-manipulation-with-sdl-surface
  - https://metacpan.org/pod/SDL2::surface