Real-time Robot Camera Control in Erlang

Supervisors:

Phil Trinder, Natalia Chechina, Gerardo Aragon Camarasa

**Aim:**

We propose to investigate an implementation of a real-time robot camera control system based on the Erlang VM and compare this implementation with possible standard approaches based on the Robot Operating System.

**Overview:**

Advanced robotics systems are typically implemented within the Robot Operating System (ROS, http://www.ros.org/), originally developed by Willow Garage (https://www.willowgarage.com/pages/software/ros-platform). ROS, an emergent de-facto standard, supports concurrent asynchronous processes that are capable of communicating by means of a message passing protocol. Services required by a robot, such as image capture, camera control and motor control gripper actuation are implemented as independent ROS “nodes” (processes) which publish information collected, or processed, e.g. images, which are then made available to consumer nodes, such as an object recognition subsystem, or face detector and tracker. ROS has grown organically, by means of contributions from the Open Source robotics community (mainly academic groups) and supports all major subsystems required by a robot. Therefore, it is possible to endow any new robot with fully operational control software by writing the appropriate drivers to allow ROS to interface to the specific hardware facilities implemented by the new robot.

The reason we want to investigate Erlang as a potential alternative core control system software architecture to ROS, is that ROS exhibits a number of limitations in practice:

* scalability issues posed by all-to-all connections between the ROS nodes
* inter-process synchronisation is based on time-stamps which can get out-of-order causing processes to fail

whereas Erlang is well known for its concurrency, distribution, ability to handle a large number of processes (~2M), and reliability mechanisms. One Erlang VM can work on 64 cores supporting millions of concurrent processes, and in experiments run within the RELEASE project, up to 400 Erlang VMs have been demonstrated (this depends on the number of global operations and intensity of data flow between the VMs). The RELEASE project aims to enable Erlang applications to operate on 100,000 cores. Therefore, Erlang has the potential to offer a solid and fault tolerant underlying architecture on which to build ROS-like processes and inter-communications mechanisms.

In order to investigate the idea of Erlang implementing a ROS-like systems architecture, we propose in this project to construct a “model” system comprising an active robot camera head (Logitek QuickCam® Sphere™, http://www.logitech.com/en-gb/support/quickcam-sphere-af), i.e. a camera that can pan and tilt under computer control. The task will be to implement Erlang processes that undertake the functions required to implement a face detector and tracker:

* Image acquisition (a C++ image acquisition driver is available)
* Face detection and localisation (using the Open CV (http://opencv.org/) C++ face detector module)
* Tracking control to drive the pan-tilt gaze direction of the camera based on the output from the face detector module (a C++ camera control driver is available)
* User interface to report the state of the system and present a view of the images currently being captured, annotated with any detected faces and their track direction.

We shall install the Erlang VM, and different Erlang processes will be responsible for various aspects of the robot camera's actions. The camera will provide input data via a C++ API to an Erlang process. Depending on the input data the Erlang process will do a corresponding action. For example, we want the camera to identify a face, and then move the head following the face movement. So, as long as the face\_recognition process gets messages from the camera that a face is identified, it will pass position of the face to the movement\_controll process that will move the head. We may also add reliability by introducing a supervision mechanism to restart failed processes.

**Objectives:**

* Install Erlang VM on the computer used for the project (possibly) under Ubuntu Linux.
* Develop codes required to interface the camera, pan-tilt drive, and Erlang VM using C++ API (Erlang has a way to speak to C++)

<http://www.drxyzzy.org/euc2003/otps2.pdf>

<http://stackoverflow.com/questions/1811516/integrating-erlang-with-c>

* Implement the control part in Erlang by receiving messages from the camera, and sending corresponding messages to the head.
* Validate the performance of the system and analyse the developed architecture in comparison to the standard ROS approach.
* Consider how the devised Erang-based implementation might be extended to exploit existing ROS nodes, or services, or how ROS might be re-designed to be based on Erlang.

**Prerequisites:**

No specific prerequisites: knowledge of Erlang and ROS desirable but not essential. Attendance on the Computer Vision Methods and Applications course CVMA4 recommended.