Real-time Robot Camera Control in Erlang

Project Goals

The aim of this project is to develop a comparative analysis of the scalability and reliability of two platforms relying on the message-passing model (ROS and Erlang), in the context of robotic component control.

ROS is an open-source framework frequently used to provide a wide array of control and monitoring service to both industrial and research-oriented robotic systems. However, it is precisely due to this support for varied robotic capabilities that ROS is theorised not to perform optimally in cases when node numbers rise sharply and the connection graph nears completion.

For such purposes, Erlang, a functional programming language designed to ease concurrent programming, could potentially provide the performance and fault tolerance needed to effectively perform such distributed robotic tasks.

The project will start out with the development of a script to interface with the Logitech Orbit MP camera via Linux. The camera is intended to act as a substitute for an integrated robotic imaging device or any such similar component.

Once this script (likely written in Python) is able to capture an image feed and perform basic face tracking and/or image processing operations, it should be integrated with the ROS environment, running on a single node. Following this, computation can be split across several nodes: one node for different processing operations (capturing images, performing face detection, pan and tilting camera lens in the 4 directions, etc.).

Subsequently, this script can be ported to Erlang, where a multitude of Erlang processes can be started to perform the same distributed computations as in ROS.

The final step of the project, after ensuring the two systems provide the required functionality, is to conduct non-functional performance testing and analyse its results. The two main metrics being explored are:

* Scalability, meaning the variation in computation and camera response time when the number of ROS nodes/Erlang processes is gradually increased.
* Reliability, which tests the two systems’ resilience when faced with partial failures (e.g. one node or process hanging or terminating abruptly); the behaviour of the remaining nodes/processes is a crucial factor to observe

Lastly, some investigation will be conducted into scaling the project to fully integrate ROS and Erlang via a supervisor model, as described in fig. 2.

Project Timeline

|  |  |  |
| --- | --- | --- |
| Preliminaries | Week 1  (21st – 27th September 2015) | * Initial meetings * Start of Literature Review * Erlang tutorials |
| Week 2  (28th – 4th October 2015) | * Obtain camera * Investigate ROS implementation for guideline * Erlang tutorials (cont.) |
| Initial Development | Week 3  (5th – 11th October 2015) | * Investigate and resolve Linux compatibility with camera, building on Logitech’s C++ files * Document Erlang fault tolerance |
| Week 4  (12th – 18th October 2015) | * Develop image processing script for ROS nodes and integrate with camera * Investigate camera control through Erlang |
| Week 5  (19th – 25th October 2015) | * Finalize initial development of camera to ROS interfaces * Document progress made in dissertation, finalize Literature Review |
| Development + Parallel Dissertation Work | Week 6  (26th – 1st November 2015) | * Commence development of camera control nodes in Erlang |
| Week 7  (2nd – 8th November 2015) | * Development work (maximum predicted time) |
| Week 8  (9th – 15th November 2015) |
| Week 9  (16th – 22nd November 2015) |
| Week 10  (23rd – 29th November 2015) | * Review and testing |
| Week 11  (30th – 6th December 2015) | * Refactoring, debugging, testing |
| Week 12  (7th – 13th December 2015) | * Document progress made in dissertation |
| Week 13  (14th – 20th December 2015) | * Prepare to start analysis of the two programs (ROS/Erlang) |
| Week 14  (21st – 27th December 2015) | * Prepare to start analysis of the two programs (ROS/Erlang) |
| Week 15  (28th – 3rd January 2016) | * Perform scalability and reliability tests on both platforms * Devise programmatic tests * Fix last minute bugs preventing testing |
| Week 16  (4th – 10th January 2016) |
| Week 17  (11th – 17th January 2016) |
| Week 18  (18th – 24th January 2016) |
| ROS vs. Erlang Analysis | Week 19  (25th – 31st January 2016) | * Refactoring * Start considering wider goals of project (figure 2) |
| Week 20  (1st – 7th February 2016) | * Research into the implementation of wider goals, to fit into a Future Work section of the dissertation |
| Week 21  (8th – 14th February 2016) |
| Week 22  (15th – 21th February 2016) |
| Week 23  (22nd – 28th February 2016) |
| Finalization | Week 24  (29th – 6th March 2016) | * Dissertation work |
| Week 25  (7th – 13th March 2016) |
| Week 26  (14th – 20th March 2016) |
| Week 27  (21st – 27th March 2016) |

Concept Diagram

**Camera control**

**Camera control**

**Image feed**

**Erlang VM**

**ROS Environment**

**Erlang processes performing image processing**

**ROS nodes performing image processing**

**erl**

**erl**

**erl**

**erl**

**erl**

**R**

**R**

**R**

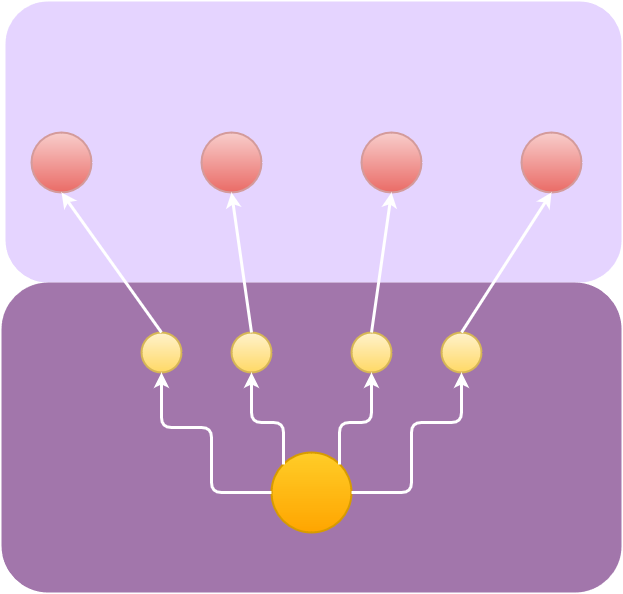
**R**

**ROS**

**OS nodes**

**Reliability testing is done by abruptly terminating nodes**

**Fig. 1**



**ROS Environment**

* **ROS Nodes perform image processing via OpenCV.**

**ROS Node**

**ROS Node**

**ROS Node**

**ROS Node**

**Erlang VM**

**erl**

**erl**

**erl**

**erl**

* **Erlang processes**

**supervise ROS nodes**

**and are in turn**

**supervised by another process**

**erl**



**Camera control**

**Image feed**

**[3]**

**[2]**

**[1]**

**Fig. 2**