My experience with robots and embedded systems

Portfolio of Parfenov Aleksey

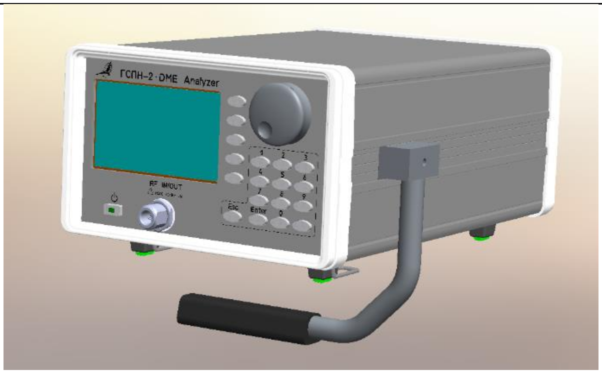
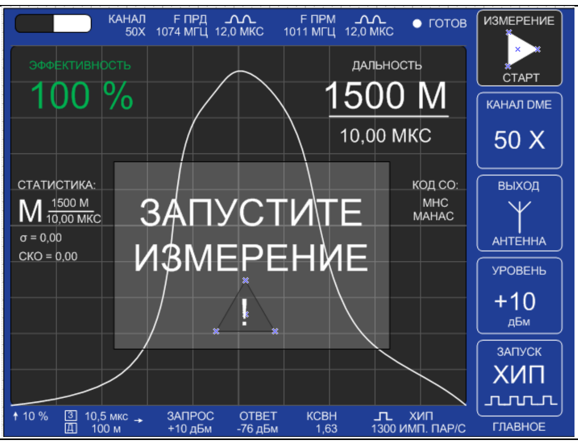
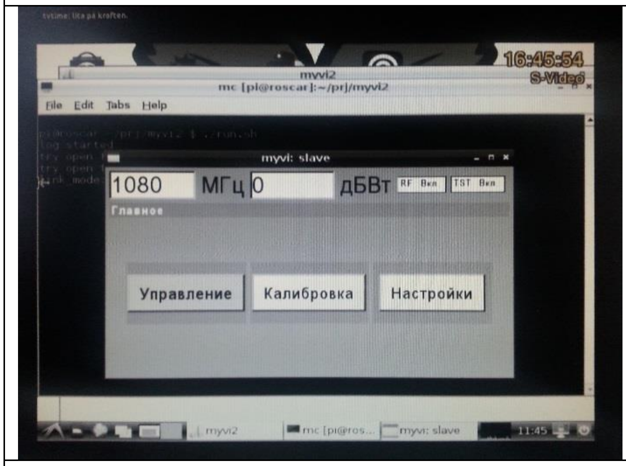
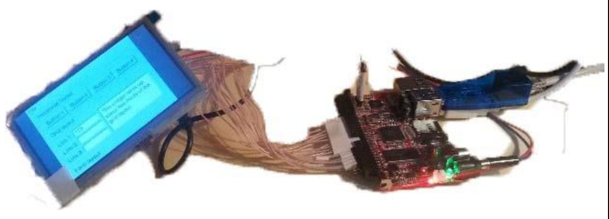
Contents

UI interface for a portable device in area of avionics.

This was a main project that I developed while worked at RTS. The project was about developing user interface unit for a portable measuring device that would allow to measure properties of the telemetry radio signal from an aircraft.

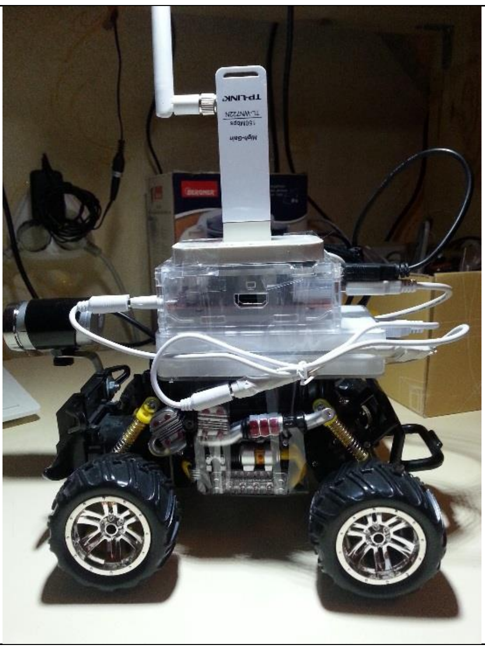
Parts that I developed:

- Software for UI module using bare C++. This includes core routines, UI custom description language, versatile set of UI primitives. While implementing that several hardware platforms were examined including NIOS II, TI TMS320 DSP.
- Software for an Altera FPGA chip using Verilog language.
- As part of an extended project the software was successfully ported to embedded Linux environment, that involved adapting display driver to work with custom hardware interface. Also, the experience of building Linux system from scratch for embedded platform was obtained.

	 КАНАЛ 50X F ПРД 1074 МГЦ 12,0 МКС F ПРМ 1011 МГЦ 12,0 МКС ● ГОТОВ ИЗМЕРЕНИЕ СТАРТ ЭФФЕКТИВНОСТЬ 100 % ДАЛЬНОСТЬ 1500 М 10,00 МКС КАНАЛ DME 50 X СТАТИСТИКА: M 1500 М 10,00 МКС σ = 0,00 СКО = 0,00 ЗАПУСТИТЕ ИЗМЕРЕНИЕ КОД СО: МНС МАНАС ВЫХОД АНТЕННА УРОВЕНЬ +10 дБм ЗАПУСК ХИП 10 % 10,5 мкс 100 м ЗАПРОС +10 дБм ОТВЕТ -76 дБм КСВН 1,63 ХИП 1300 ИМП. ПАР/С ГЛАВНОЕ
General view	Example of the UI
 16:45:54 S-Video myvi2 mc [pi@roscar]:~/prj/myvi2 File Edit Tabs Help myvi: slave 1080 МГц 0 дБВт RF Вкл TST Вкл Главное Управление Калибровка Настройки myvi2 mc [pi@ros... myvi: slave 11:45	
Running on Raspberry Pi	Embedded Linux system

Autonomous car with ROS and video streaming

The idea of the project was to make an autonomous car with video streaming feature, that can be driven remotely over WI-FI. Finally, this working prototype was made.


Autonomous toy car with ROS client-server architecture

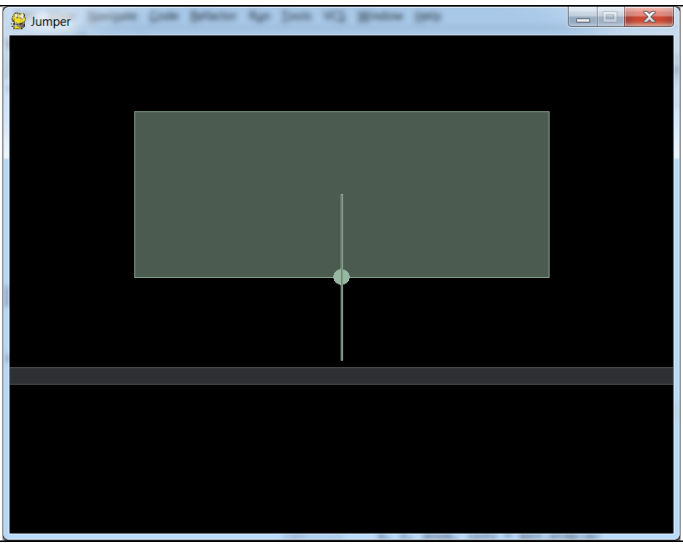
As a mobile platform, a tiny toy car was used. All on-board processing was on Raspberry Pi rev. B board. There is also a WiFi stick attached and usual Web camera. To control motors of the car that was originally driven by a remote-control unit, the receiver chip was removed and the Pi board was connected instead. As for software side, this was a distributed ROS system. There is custom ROS client for motor control and raw video capture client at car side. At the PC side, there is a ROS node that read wheel position from a joystick and send it to the car.

Live video streaming with minimal delays turned out to be a hard problem. The least achieved delay was about 300 ms with the use of Gstreamer that allowed hardware-encoded video on Pi platform.

Simulation of the project was also performed within ROS. That included conversion from URDF to SDF file format with Freemarker and shell scripts, running the model in Gazebo and control it manually with turtlebot_teleop node as well as using move_base that includes use of different SLAM algorithms. Theoretically it is possible to build a robot like this that could build a map with monocular SLAM algorithm and navigate freely using only its camera.

Experiments with Reinforcement Learning

As part of studying reinforcement learning basics and applications of it, several simulations on the PC were performed. The final goal of all of that is to acquire related skills to usefully apply RF approach in real world challenges such as an autonomous control of underactuated systems like a robotic arm or bipedal movement. The frameworks used are Python, Tensorflow, Box2d, and others.

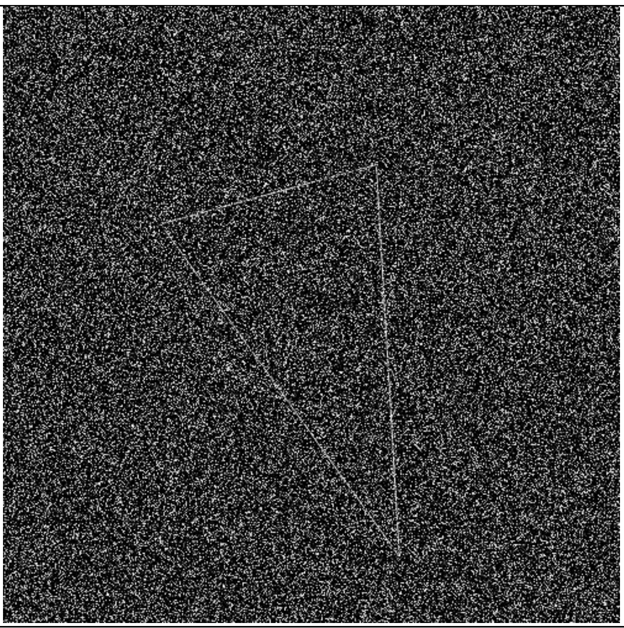


Screenshot of a simulation example

There is also a Java version of it. The online video of the simulation on Java is here: Jumper rl4j + Jbox2d

Test project for a company that develops image recognition software.

The task was to 1) generate a noisy image of a triangle, then 2) recover coordinates of the triangle vertices from the image as precisely as possible. This is what I got:



Generated image of a triangle with noise

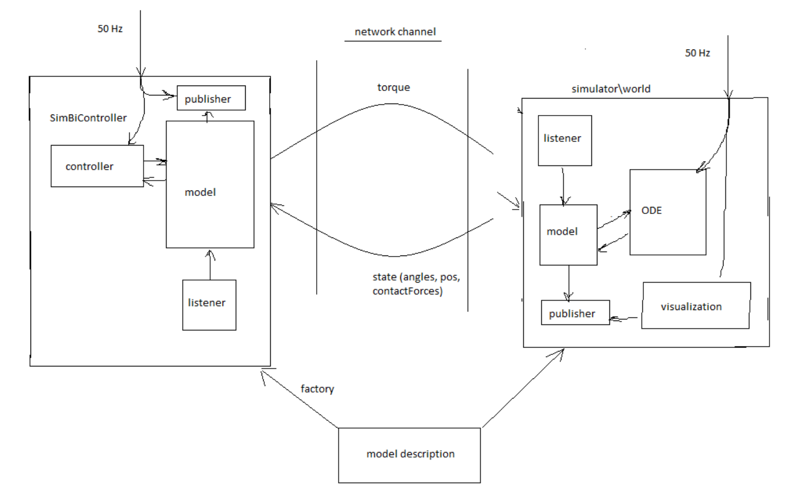
Output:

```
read file: image.pgm
start processing

elapsed time: 0.928

detected points:
212.298 140.886
105.661 334.021
432.056 276.595

original points:
106.151 333.04
431.652 276.186
212.197 140.774

error for pt(106.151 333.04): 1.09633
error for pt(431.652 276.186): 0.575356
error for pt(212.197 140.774): 0.150457

avg err: 0.60738
```

First version of the algorithm was made on Python, and moved to C++ finally because of strict time limitations – 1 second for recognition. OpenCV library was used (Hough Transform), KMeans, some CPU cache optimization techniques were applied.

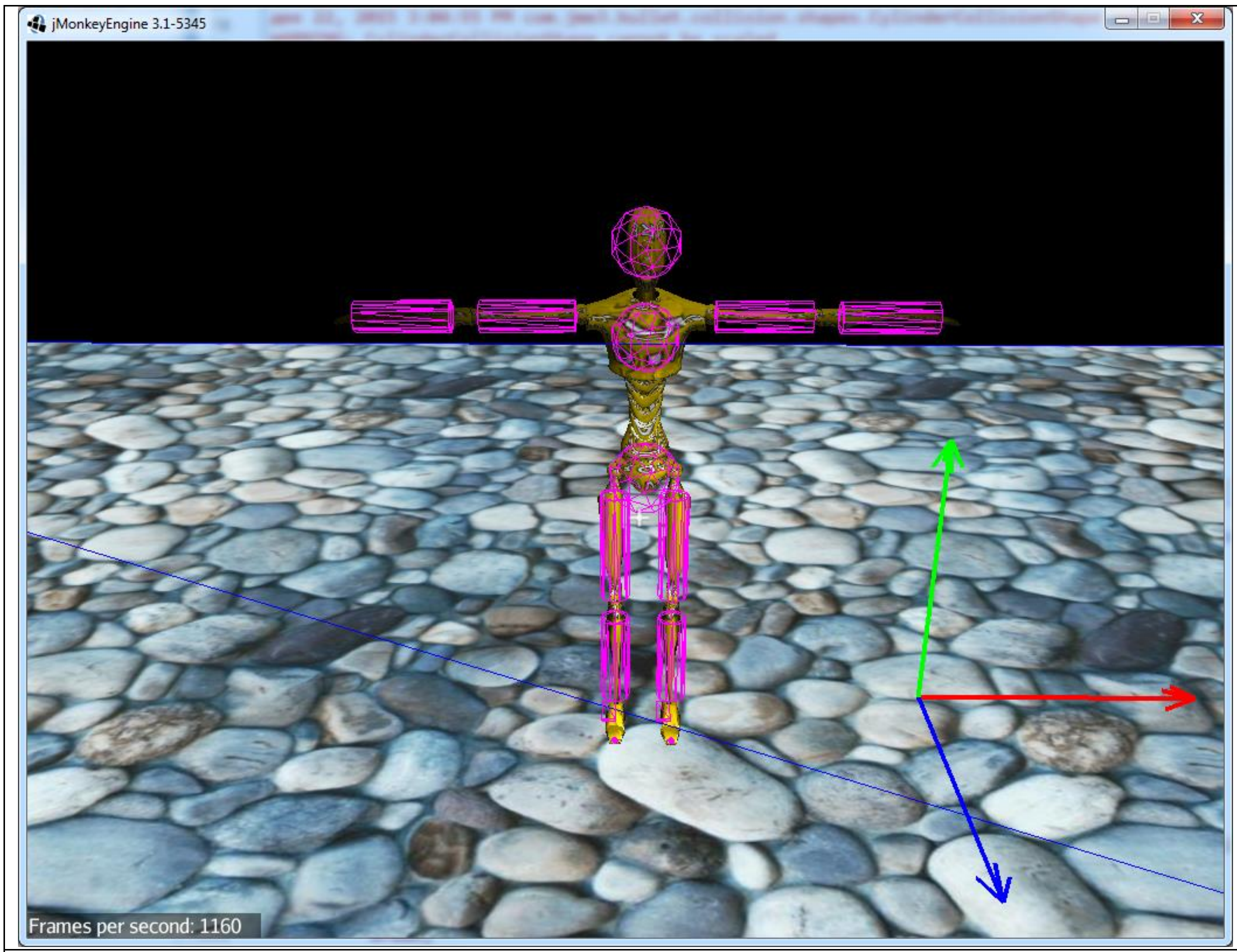
Application of SIMBICON framework to control a virtual robot inside ROS Gazebo simulator.

The idea of this project was to apply an existing technique that is known as SIMBICON to control a virtual model in ROS Gazebosim. SIMBICON is a known approach that allows naturally looking simulation of virtual bipeds. It stands for Simple Biped Locomotion Control and its description is available in the Internet. In their work a custom simulation environment was used. So, the solution was to integrate the existing implementation with Gazebosim and ROS environment. Firstly, the original framework was divided. The one part of it was responsible to run a physical simulation and rendering. The other was dedicated to run a controller. These two parts communicated over network.



Interfacing client and server in 2-tier model of SIMBICON

After successful run in this configuration it was supposed to replace rendering part of it with Gazebosim. But because of the difference between control interfaces it was decided to implement control logic from scratch in Java. Java client for Gazebo was successfully implemented. It also can copy state of a modeled system from Gazebo and apply it to it's own 3D model.



Custom model rendered with JavaFX/jMonkeyEngine

The Java client can read STL files into a mesh, parse SDF (Gazebosim) robot description and display it.

You may see sources at Github.