Projekt zespołowy

Zintegrowany system wizyjny dla robota społecznego z wykorzystaniem narzędzia Yocto

GolemOS

8th June 2021

Wrocław University of Science and Technology

Skład grupy:
Emilia Szymańska, 248975
Igor Zieliński, 248944
Jacek Multan, 248964
Krystian Matusiak, 249460

Prowadząca:
dr hab. inż. Elżbieta Roszkowska

Contents

1 Project description

1.1 Project objective

The object of interest of the project is *Golem* - a humanoid robot equipped with numerous modules combining mechanics, electronics and cybernetics. Golem's originators are a part of Smart Control Labs (SCLabs). Currently, the robot is in the form of a torso with an attached arm, whose movement is provided by a custom system of hydraulic muscles as in [1].

The task of the project team was to design and build the foundation of the visual perception module. With the help of a depth camera, the robot will recognize objects and react accordingly to changes in the environment. Operating system image with installed necessary drivers and programs was created with the help of the Yocto project, then tested on a virtual machine (QEMU), Raspberry Pi platform and finally Jetson Nano. It was necessary to find adequate camera drivers and processes supporting data transfer between GPU and a camera.

The expected result was a working, personalized operating system on the NVIDIA's Jetson Nano platform. Jetson Nano is widely used in AI applications as in [2], while creating an own OS allows developers to have more control over the vision system. The synchronization of later implemented robot's tasks will be based on the ROS platform, therefore the group's task was also to make a choice between ROS1 and ROS2 (and justify it).

The results (alongside with the step by step tutorial for every used machine) are presented on the website: `https://emilia-szymanska.gitlab.io/golemos/index.html`.

1.2 Budget

Hardware resources:

- microcomputer Raspberry Pi 4 - 270 PLN,
- NVIDIA Jetson Nano development kit- 380 PLN,
- Intel D435i depth camera - 890 PLN.
- server - 3000 PLN.

Personal resources:

- four persons spending six hours weekly during 15 weeks throughout the academic semester (summing up to 360 man-hours); assuming that the hourly rate is 20 PLN brutto, the total cost is equal to 7200 PLN.

1.3 Project management

The project is carried out in cooperation with the Smart Control Labs (SCLabs) group, which imposes a certain organizational structure of the project group. The project leader is directly accountable to the responsible person appointed by SCLabs. The form of the used communication was a text channel on a personalized Mattermost platform. This web-based, configurable chat service allowed - in addition to exchanging messages between users - to share, search and integrate files.

Team meetings (with SCLabs representant) were held weekly (late Wednesdays) to discuss the work division for the forthcoming week. These meetings were in a form of a remote video conference on the Discord platform. In the event of difficulties and the need of brainstorming, a greater frequency of meetings is allowed. Minor doubts are clarified on an ongoing basis via chat. Documentation is kept on the website on an "open source" basis.

The team leader is responsible for monitoring the work, ensuring the flow of information on the SCLabs-team channel, contacting the SCLabs representant on behalf of the group, and contributing to each team member receiving a fair and commensurate final evaluation of the project. Potential conflicts are solved together, reaching a solution during a meeting attended by all group members. The decision is made on the basis of a majority of votes, and in the event of an equal number of votes, the vote of the team leader is decisive.

1.4 Team members

The team member list with short description and photos is presented below.

1. **Emilia Szymańska** [rys. 1a] (248975@student.pwr.edu.pl) - **team leader**. Due to her experience in organizing work and project management, she is responsible for coordinating the work of the team. Knowledge of ROS, web development and meticulousness in terms of documentation are big advantages in term of this team project.
2. **Igor Zieliński** [rys. 1b] - a Raspberry Pi platform expert. Strong programming (especially in the field of C++/C/Python) and analytical skills contribute to the quick localization of algorithmic and system problems, thus results in efficient learning.
3. **Jacek Multan** [rys. 1c] - interested in vision systems and methods related to image processing. He intends to dive into the world of artificial intelligence using embedded systems.
4. **Krystian Matusiak** [rys. 1d] - the Yocto project and embedded systems enthusiast. Targeting the success and constant pursuit of achieving the goal make him quickly acquire new skills and ambitiously approach the assigned tasks.

(a) Emilia Szymańska

(b) Igor Zieliński

(c) Jacek Multan

(d) Krystian Matusiak

Figure 1: Członkowie zespołu.

Due to the specifics of the project, most of the tasks were performed by all members, because it is not possible to create a system image for a virtual machine, RPi4 and Jetson in parallel, it had to be done subsequently. Each team member had to become familiar with the principle of using the Yocto project to build a Poky in subsequent stages, and parallel testing of various options in the system configuration file helped to find the optimal solution. For these reasons, a different convention of division of tasks was adopted - individual members of the team were responsible for the corresponding stages of work. This responsibility included analyzing the Yocto documentation for a given platform, presenting the acquired knowledge to other team members and shaping the work for the near future (indicating who is to test which functions, recording successes and failures, drawing conclusions and adequately modifying the plan). There is a breakdown of the platforms with their future experts below:

- Krystian Matusiak - virtual machine,
- Igor Zieliński - Raspberry Pi 4,
- Jacek Multan - Jetson Nano.

Emilia Szymańska as the leader was responsible for supporting the "temporary experts" in analyzing the documentation, organizing training meetings with SCLabs, and in addition created and managed the project website, chose between ROS and ROS2, learnt about the depth camera and prepared application finally launched on Jetson Nano.

2 Used tools and resources

To configure and build operating systems the following tools and documents were used:

- Yocto Project - Poky repository [3] - open source full-platform build tool based on Linux;
- Yocto Project - Mega-Manual [4] - step by step instructions.

The Poky repository itself uses the following open source tools:

- OpenEmbedded [5] - environment for automating Linux compilation on embedded systems;
- BitBake [6] - tool for building and distributing Linux packets;
- QEMU [7] - virtual machine (Quick EMUlator).

Throughout the project, different open-source meta-layers were used for building the OS for specific platforms:

- **meta-openembedded** - meta-layer containing recipes for basic functionalities e.g. Python, Wifi, ssh;
- **meta-raspberrypi** - meta-layer for Raspberry Pi platform;
- **meta-ros** - meta-layer with ROS configuration;
- **meta-intel-realsense** - meta-layer with libraries for operating with Intel Realsense cameras;
- **meta-tegra** - meta-layer for Jetson Nano OS.

We have also tried out a variety of Poky branches (as some meta-layers were not compatible with specific configurations):

- sumo,
- zeus,
- dunfell,
- gatesgarth,
- hardknott.

3 General steps to build and run the OS

Setting up Poky follows some specific schema:

1. Make sure that your computer is able to run the bitbake building. That is why the following tools must be downloaded:

```
$ sudo apt-get install gawk wget git-core diffstat unzip texinfo
    gcc-multilib build-essential chrpath socat libsdl1.2-dev xterm bmap-tools
    make xsltproc docbook-utils fop dblatex xmlto cpio python python3
    python3-pip python3- pexpect xz-utils debianutils iputils-ping python-git
    bmap-tools python3-git curl parted dosfstools mtools gnupg autoconf
    automake libtool libglib2.0- dev python-gtk2 bsdmainutils screen
    libstdc++-5-dev libx11-dev
```

2. Clone Poky distribution, replacing *{branch}* with the relevant branch name:

```
$ git clone --single-branch -b {branch} git://git.yoctoproject.org/poky.git
```

3. Clone relevant meta-layers (structures that contain classes, recipes and conf files). The example for cloning a meta-layer is presented below:

```
$ git clone --single-branch -b {branch} git://git.yoctoproject.org/meta-layer
```

4. Run the oe-init-build-env script, that defines OpenEmbedded build environment settings:

```
$ source ./poky/oe-init-build-env
```

5. Modify local.conf accordingly to your needs. Depending on what device you have, option „MACHINE" ought to be related to it.

6. Modify conf/bblayes.conf - include meta-layers you want to use.

7. Build an OS image through bitbake command. To have a graphical interface, we ran core-image-sato:

```
$ bitbake core-image-sato
```

Unfortunately it may take a long time (unless you have a computer with a lot of cores). After successfully building the image, it ought to appear at ./tmp/deploy/images/ directory with *wic.bz2 or *sdimg extensions. Save created image on SD card (be cautious which partition is associated with the card):

```
$ sudo bmaptool ${OSimage} /dev/${your_device_name}
```

4 QEMU

Configuring QEMU did not demand plenty of modifications. The main change was in machine selection in *local.conf* file (*MACHINE ?= "qemux86-64"*). The result of this first phase is a working system on x86-64 architecture. We booted the system image with QEMU and this process is presented in Fig. 2.

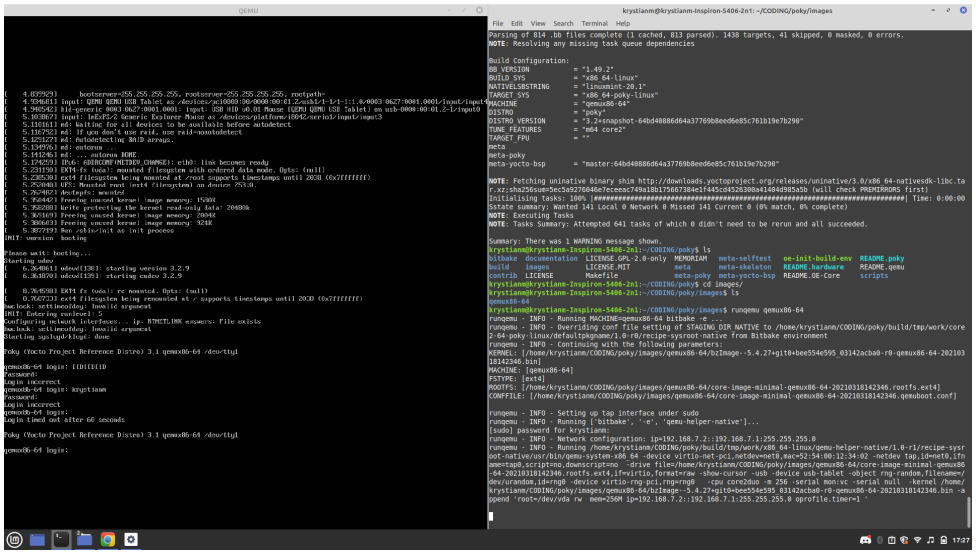

Figure 2: Booting the system on QEMU.

Executing basic bash commands can be seen in Fig. 4.

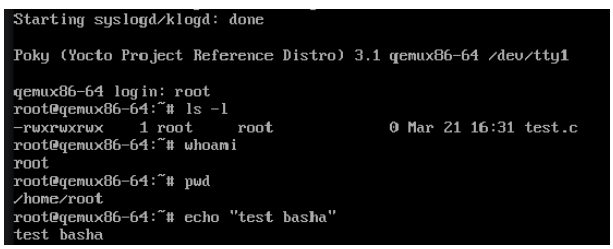

Figure 3: Using bash commands in the command line of the QEMU system.

5 Raspberry Pi

This part of the project required using meta-raspberrypi and meta-openembedded. With basic operating system the configuration and build was not difficult - difficulties appeared when adding ROS and Intel camera drivers. First of all, camera drivers compatible with our camera were added in Hardknott branch, whose current version was from May 2021. Second of all, adding meta-intel-realsense and meta-ros caused problems connected with duplicating librealsense libraries - we had to exclude librealsense2 recipe from meta-ros.

Finally, a systemd script was added to enable our own custom application to be launched right after booting.

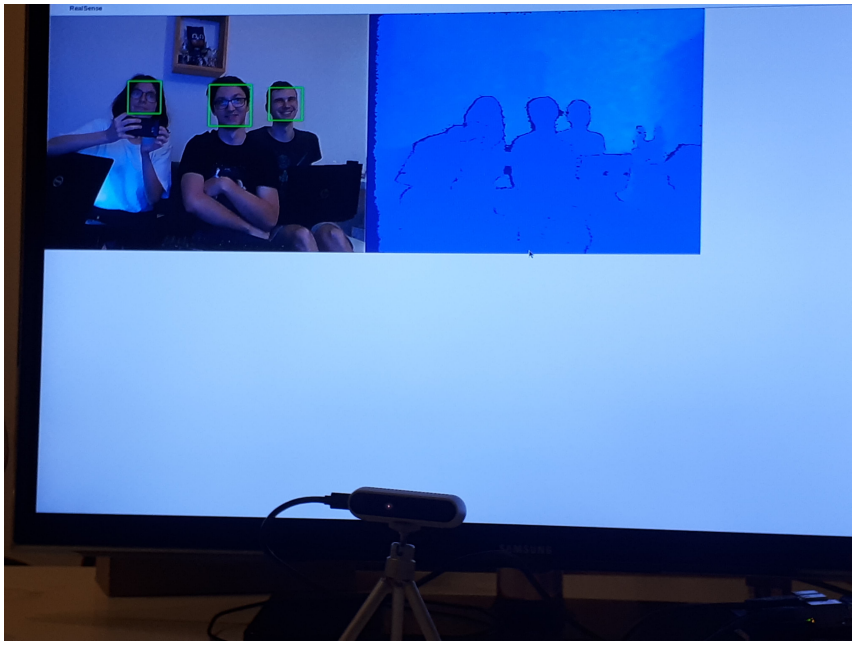

Figure 4: Raspberry Pi with GUI and a program automatically launched while booting.

5.1 Comparison between ROS and ROS2

Because of its numerous advantages and only few drawbags, we have decided to continue our work with ROS2. The comparison of the two versions of ROS are presented in table 1.

ROS	**ROS2**
widely known for a long time	relatively new
a lot of tools adjusted for ROS	tools still being transferred from ROS to ROS2
API not necessarily the same between roscpp and rospy	one base library (rcl) with similar API between rclcpp and rclpy (easy to develop rcljava, rclnodejs etc.)
targeting C++98	targeting C++11 and C++14
freedom in node implementation	modular structure for writing a node
launch files in XML	launch files in Python
need to run the ROS master	nodes are independent and not tight to a global master
parameters handled by the parameter server	node declares and manages its own parameters
synchronous services	asynchronous services
catkin build system	ament build system with colcon tool
Ubuntu as a main OS target	Ubuntu, MacOS and Windows 10 as OS targets

Tabela 1: Comparison between ROS and ROS2.

6 Jetson Nano

Building an OS image for Jetson Nano required much more attention than the previous platforms, mainly because of its complexity. To use functionalities available on Jetson, an SDK tool was needed to configure CUDA and other extra options. This tool requires GUI, therefore it could not be used on our remote server. After installing necessary tools, it is possible to continue with bitbake process. Modifying local.conf and bblayers.conf requires some more sophistication as described in our step-by-step tutorial.

After inserting an SD card, plugging HDMI cable and power supply to Jetson Nano board it is possible to test the system as presented in Figures 5.

(a) Jetson Nano while booting.

(b) Jetson Nano with booted system.

Figure 5: Jetson Nano results.

7 Application

An application with a neural network was prepared for the project. Trained with YOLOv5 architecture, the network detects human faces and classifies them into three classes: with mask, without a mask and with a mask worn improperly. The example of usage of the network is presented in Fig. 6.

Unfortunately, it turned out that PyTorch library (used for YOLO processing) is not built for ARM architecture - one has to build the whole library on their own. As PyTorch is not provided in an accessible way for Yocto development, we were forced to create a simple script with OpenCV to present the functionalities of our OS.

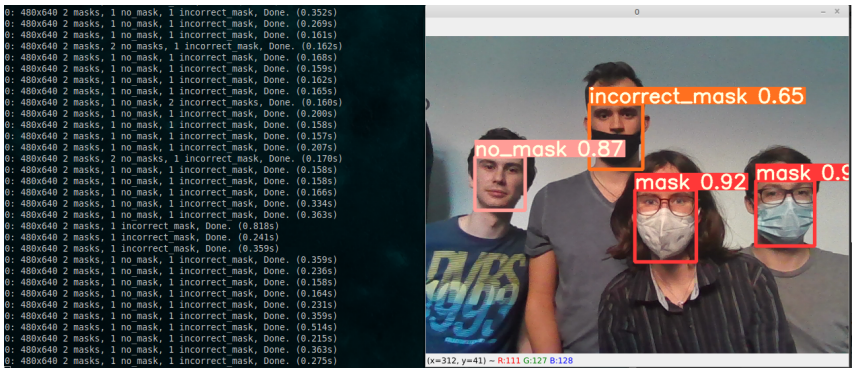

Figure 6: Running the network processing the image from a web camera.

8 Conclusions

It turned out that the project was more complex than we expected it to be. We encountered a lot of difficulties during the project:

- server unavailability for three weeks (need to build systems locally which took up to 8 hours each),
- libxml failing at fetching (need to download it and put into directiories manually),
- different architecture on the server (later changed to the proper one, but we got random errors at the end of bitbake),
- SDK NVIDIA manager compatible only with Ubuntu 16.04 and 18.04, requires GUI, which our server didn't have,
- need to buy an SD card with more memory (the OS didn't want to boot with 16GB),
- proper camera drivers on the newly created Poky branch (Hardknott), which has been released only recently,
- need to switch between different Poky branches because of meta-layers incompatibility,
- incompatible meta-ros and meta-intel-realsense (need to manually remove librealsense2 recipes in meta-ros),
- PyTorch not built for ARM architecture (no option for simple adding the recipe to Poky).

All of these factors contributed to only partly finished project. We have an application with neural network image processing, a Raspberry Pi system image with ROS2 and an automatically launched script using the camera as well as a basic Jetson Nano system image with configured neural network tools. All in all, we have created a tutorial for building three OS images. Detailed documentation is available under `https://emilia-szymanska.gitlab.io/golemos/index.html`. The Jupyter Notebook with the neural network detecting masks can be found here: `https://github.com/emilia-szymanska/mask_detection_and_classification`.

Although there were quite a lot of obstacles, we managed to create a basic vision system - it will be easier to develop the system in the future having all the experience we got during the project.

References

[1] B. Tondu, V. Boitier, and P. Lopez, "Naturally compliant robot-arms actuated by mckibben artificial muscles," in *Proceedings of IEEE International Conference on Systems, Man and Cybernetics*, vol. 3, pp. 2635–2640 vol. 3, 1994.

[2] S. Cass, "Quickly Embed AI Into Your Projects With Nvidia's Jetson Nano," June 2020.

[3] Y. Project, "Poky documentation," `https://git.yoctoproject.org/cgit/cgit.cgi/poky/`.

[4] Y. Project, "Mega Manual," `https://www.yoctoproject.org/docs/current/mega-manual/mega-manual.html`.

[5] "OpenEmbedded main page," `https://www.openembedded.org/wiki/Main_Page`.

[6] C. L. Richard Purdie and P. Blundell, "BitBake User Manual," `https://www.yoctoproject.org/docs/1.6/bitbake-user-manual/bitbake-user-manual.html`.

[7] "QEMU main page," `https://www.qemu.org/`.