Project Overview

Kehan Xing

3B Computer Engineering

University of Waterloo

Development of bead-based detection of SARS-CoV-2 IgM/IgG in a multiplex POCT platform

Thinkari Research Inc.

Mississauga, ON

Software Developer *Sep. 2020 – Apr. 2021*

Project Description:

This project focuses on developing a bead-based immunoassay system for the simultaneous and quantitative measurement of human IgM and IgG specific to SARS-CoV-2. It uses two development boards for combining the software and the hardware. One of them is used for sensor detection and motor control, and the other one contains backend applications with Ubuntu 16.04 system.

My role:

- **Participate in the development of bead-counting algorithms** - These algorithms analyzed images based on the OpenCV library and output the density of the desired kind of particle. I optimized the running time of the algorithms as well as ported them from Python to C++.
- **Build a motor-control module for sample preparation -** This module required the motor to move back and forth based on the messages sent from the backend. It took information including the number of movements, the distance of a single movement, the speed, and the motor number, then controlled the motor to move accordingly.
- **Develop UI design using Qt5** and connect it with the backend applications using the API functions provided

Goal Reached:

Optimizing running time

Problem Description: The beads-counting algorithm had an acceptable running time in the virtual machines. However, it needed 10 times the running time on the embedded system.

Steps:

- **Research the reason for the long running time -** It's related to the CPU and the 32-bit Ubuntu system, but it's also related to the OpenCV library. The optimization of OpenCV on the ARM environment was not very ideal, so I started to research other image processing libraries.
- **Research other image processing libraries** including Scipy, Mahotas, Libvips etc. and record their features related to the running time
- **Find the time-consuming functions -** The running time of each line was tested and recorded to find the most time-consuming functions in the original algorithm which are the MORPH_CLOSE function and the bilateralFilter function.
- **Find similar functions in other libraries** and record the required parameters and the examples on the website
- **Test correctness after replacing the functions -** After replacing the functions from other libraries, the results should remain the same.
- **Test running time -** The last step was testing the running time of the algorithm again after replacing functions and recorded the change of the running time.

Porting from Python to C++

Problem Description: In order to better integrate the software, the algorithm need to be ported from Python to C++.

Steps:

- **Find a replacement for Numpy -** Numpy is an important library used in the algorithm that doesn't have a C++ API, so the first step was finding an alternative for it, which was Numcpp.
- **Find alternatives in OpenCV -** The matrix variable type used in OpenCV was Mat, but the variable type used in Numcpp was NdArray. To avoid unnecessary conversions, I decided to use similar functions in OpenCV first, if applicable.
- **Change variable types and functions related to the change of language –** The methods to go through a directory, store values in vectors or writing to a file, etc., are different in the two languages, so I changed them in this step.

- **Change formats of the image processing functions -** The C++ implementation of the image process functions are different too, especially for the requirements of the parameters. The formats are changed according to the documentation on the library websites.

- **Debug for successful compiling and counting -** Some errors related to the conversion from the Numpy array in the Python code to OpenCV matrix in the C++ code were successfully solved.

- **Debug for correctness -** The ported algorithm was tested with groups of images and the results were compared with the ones from the Python version. Some images had different results, but we concluded that the differences were acceptable, as they wouldn't change the density a lot.

Developing sample preparation module

Problem Description: The project requires an extra module to mix the liquids before taking pictures and analyzing. Previously, this step was done manually, so I developed a sample preparation module that used motors to mix the liquids.

Steps:

- **Design communication messages -** To control the motors, command messages are used. For this new module, I designed a message format that contained the number of movements, the distance of a single movement, the speed of the motor and the motor number.

- **Program the motor-controlling module -** The program read the information in the message and used the motor controlling functions to control the movement. The program also sent a message back when the work was done.

- **Design protection mechanism using sensors -** The distance in the message may exceed the physical distance, so a protection mechanism using two sensors on both sides were added to avoid hardware damage.

- **Combine with the main motor controlling project -** The last step was combining the sample preparation module with the main project and testing its functionalities.

Caption Conductor

Evertz Microsystems LTD.

Burlington, ON

Embedded System Developer *Jan. 2020 – Apr. 2020*

Project Description:

This project is a flexible IP-based real-time caption management system. Caption Conductor eliminates the need for ageing captioning technology and connects the caption encoders with live captioners over IP by use of a simple web-based interface. It uses an access server, a facility server, a dashboard and caption encoders to connect live captioners with the videos.

My role:

- **Troubleshoot software problems –** There were some problems caused by special cases in the program, including audio-only source, which was solved by adding a default video, and invalid caption format or characters, which was solved by detecting and deleting invalid components etc.
- **Participate in updating the embedded system due to hardware requirements -** To accommodate hardware requirements, the system was updated from Ubuntu 16.04 to Ubuntu 18.04. At the same time, the code needed to be updated from Python 2.7 to Python 3.6 to make it compatible with both Python 2 and 3, and it was my duty to update the code to Python 3.6

Goal Reached:

Update Caption conductor code

Problem Description: The project's source code was originally written in Python 2.7, but it needed to be compatible with both Python 2 and Python 3 due to hardware requirements.

Steps:

- **Check library availabilities –** There are more than 20 libraries used in the project, so the first step is to check their availabilities of Python 3.6.
- **Set up new virtual machines and check the install package -** In order to have an independent developing environment, two new virtual machines were set up for the facility server and access server to run the code and use the install package to install all the libraries needed.

- **Test caption conductor on the new virtual machines -** Before updating the code, the entire caption conductor was manually tested on the two new virtual machines. A problem with Janus (a WebRTC tool) was found, so I compiled Janus again on the new virtual machines and updated the install package.

- **Debug errors for successfully compiling -** An automated translation tool, Python 2to3, was used to translate simple differences such as strings. Then I compiled the code under Python 3.6 and debugged the errors appeared. The errors were mainly related to library versions and string conflicts. The debugger inside Eclipse was used to solve them.

- **Debug errors for successfully running as a server -** The server with the updated code was set up to be tested in this step. All the functionalities were manually tested and compared to the performance of the original code.

- **Write TU report** – A TU (Technical Uncertainty) report was written to summarize all the problems and solutions.

Real-time Executive

University of Waterloo

Waterloo, ON *May 2021 – Aug. 2021*

Project Description:

This project is a small Real-Time Executive (RTX) for Keil MCB1700 board populated with an NXP LPC1768 microcontroller. The RTX can take tasks and manage them according to their priorities. The project can handle both non-real-time tasks and real-time tasks, and there are also two built-in tasks to check the status of the system.

My role:

- Participate in designing the functions to manage memory and tasks
- Build unit test cases to test the functions

Goal Reached:

- **Create kernel memory management functions -** The memory allocator was developed using a binary buddy system. A bit tree was used to record the status of each bit, and a link list was used to record the free memory blocks.
- **Manage tasks according to priorities -** There were four priorities in total. When a task was created, it was assigned to one of the priorities and stored in the ready queue which was a priority queue. The tasks could change their priorities or other tasks' priorities if they had permission. The system picked the first task in the ready queue with the highest priority when the previous task was done.
- **Create communications between tasks -** The tasks could create a mailbox for themselves and send messages to other tasks as well as receive from other tasks. There were blocked and unblocked send/receives for different purposes.
- **Add real-time (periodic) tasks to the system –** Real-time tasks were added to the system using timers. The system checked the deadline of each real-time task every 500 μs using an interrupt. The real-time tasks had the highest priority, so a preemption occurred if a non-real-time task was running.

Automatic Door Opener

University of Waterloo

Waterloo, ON *May 2021 – Aug. 2021*

Project Description:

This project automates an entry/exit doorway to a customer site that senses customers automatically with programmable trigger levels for proximity on STM32F401RE. The hardware components were designed and tested using the simulator software Proteus.

My role:

- Design and test the hardware components
- Develop software with C

Goal Reached:

- **Pick and test the hardware components -** The hardware components in this project are two ultrasonic sensors, a motor, a motor driver, two distance limit switches, a collision switch, an LCD and LEDs. I read the datasheets of the available components and tested their behaviours to pick the most suitable ones.
- **Connect circuits -** The supply voltage for the MCU was 3.3V, and some of the components required a 5V voltage. Therefore, step-up and step-down voltage level translations were applied. Also, all the components were placed on a PCB to choose the correct PCB size as well as connect all of the components together.
- **Program software -** There were three modes in the project. The locked mode brought the door to fully closed or fully open. The set-up mode allowed users to set the detecting distance and moving speed. The run mode detected customers using ultrasonic sensors. There were also switches used in the run mode to avoid physical damage to the door and customers. The software read the status of the ultrasonic sensors and the switches to decide the motor's movement.
- **Create test cases -** The test cases including mode conversion, parameter setting in the set-up mode and hardware behaviour in each mode were manually tested and written in the report.