Tallinn University of Technology

Faculty of Information Technology

Department of Computer Control

Igor Petrov

Raspberry Pi based System for Visual Detection of Fluid Level

Bachelor's Thesis

Supervisor(s): Eduard Petlenkov,

Aleksei Tepljakov,

Department of Computer Control,

Tallinn University of Technology

Tallinn 2014

Declaration: I hereby declare that this Bachelor's thesis, my original investigation and achievement, submitted for the Bachelor's degree at Tallinn University of Technology, has not been submitted for any degree or examination.

Deklareerin, et käesolev bakalaureusetöö, mis on minu iseseisva töö tulemus, on esitatud Tallinna Tehnikaülikooli bakalaureusekraadi taotlemiseks ja selle alusel ei ole varem taotletud akadeemilist kraadi.

Igor Petrov

Date .....

Signature .....

Contents

List of Figures

Nomenclature

API	Application Programming Interface
CPU	Central processing unit
CSI	Camera Serial Interface
FPS	Frames Per Second
GUI	Graphical User Interface
IDE	Integrated development environment
MMAL	Multi-Media Abstraction Layer
RDP	Ramer– Douglas– Peucker algorithm.
ROI	Region of interest
SoC	System on a Chip
UDP	User Datagram Protocol

Abstract

Raspberry Pi based System for Visual Detection of Fluid Level.

The main task of this work was an attempt to develop a system based on an embedded system or single-board computer equipped with digital camera which is capable of visually tracking a real time process and provide the required data to the device controlling the process.

The work was concentrated on developing a software written in C/C++ for GNU/Linux operating system with a purpose of tracking of water level in a cuboid tank, while communicating with a PC running MATLAB.

The thesis is in English and contains 27 pages of text, 4 chapters, 1 table, and 12 figures.

Annotatsioon

Raspberry Pi-l põhinev vedeliku nivoo visuaalse tuvastamise süsteem.

Selle praktilis-rakendusliku töö peamiseks ülesandeks oli välja töötada selline sardsüsteem, mis oleks võimeline reaalajas visuaalselt jälgima vedeliku nivood paagis reaalajas ning saatma sobivad andmed juhtseadmele. Sardsüsteemi platvormiks oli valitud Raspberry Pi—ühest trükkplaadist koosnev arvuti, mille külge on ühendatud ka sobiv digitaalne kaamera.

Lõputöö tulemusena on GNU/Linux operatsioonsüsteemile loodud C/C++ programm, mis on võimeline mõõtma vedeliku nivood läbipaistvas paagis ning saatma edasi saadud informatsiooni traadita arvutivõrgu kaudu. Välja töötatud lahendusel on otsesed rakendused: paagi andurite kalibreerimine ja vedeliku nivoo juhtimine.

Lõputöö on kirjutatud inglise keeles ja sisaldab 27 lehekülge teksti, 4 peatükki, 1 tabel ja 12 joonist.

Acknowledgments

I would like to thank Department of Computer Control for the laboratory equipment they provide to me in order to make this thesis possible.

I would also like to thank my supervisors Eduard Petlenkov and Aleksei Tepljakov, as they both provided me with many points to include, and OpenCV community for great documentation and tutorials.

Finally, I wish to thank my close friend Aleksandr Nikandrov for helping with translation this thesis into English.

Chapter 1

Introduction

1.1 Short Description of The Work

In the past years computer vision have founded itself in a wide scope of application, starting from smartphone games using augmented reality, finishing with critical-safe traffic control systems. But as of current, they have not found much use in controlling real-time systems, mainly because of computational complexity and all-in-all low system reliability. On the other hand, visual feedback loop is efficient for distantly observing objects, and gather information about them, using a simple digital camera.

The task of this thesis was to demonstrate that modern day embedded systems have enough computing power to control real-time processes, using visual feedback loop, and to find an application for computer vision within control theory. From the available in laboratory real-time systems it was decided to use the Multitank, since its process is easy to observe visually.

In Chapter 2 the reader is presented with used equipment and software solutions.

In Chapter 3 the steps of water level detection algorithms are described.

Chapter 4 is focused on developed application and real-time control applications.

Chapter 2

System description

2.1 INTECO Multitank System

The Multitank System comprises a number of separate tanks fitted with drain valves. Two of the tanks have varying cross sections. These introduce nonlinearities into the system. A variable speed pump is used to fill the upper tank. The liquid outflows the tanks due to gravity. The tank valves act as flow resistors. The area ratio of the valves is controlled and is used to vary the outflow characteristics. Each tank is equipped with a level sensor.

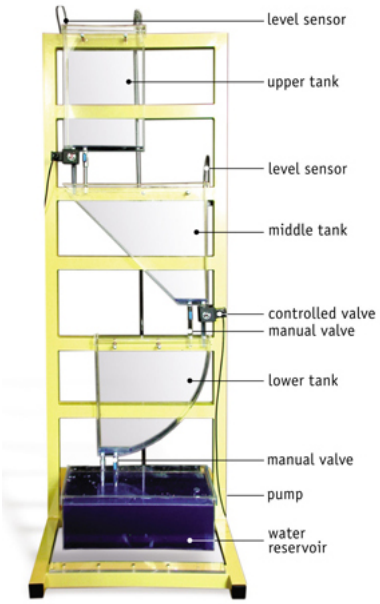

Figure 2.1: Multitank system

The Multitank System is designed to operate with an external PC-based digital con-

troller. The computer communicates with the level sensors, valves and pump by a dedicated I/O board and the power interface. The I/O board is controlled by the real-time software which operates in the MATLAB/Simulink environment [1].

2.2 Raspberry Pi

From the available laboratory equipment, single- board computer Raspberry Pi was the best platform. It is the ARM processor based size of a credit card and has the hardware specifications of mobile phone.

SoC	Broadcom BCM2835
CPU	700 MHz ARM11 ARM1176JZF-S core
GPU	Broadcom VideoCore IV
Memory	256 MiB
USB 2.0 ports	2
Video outputs	Composite RCA, HDMI
Video inputs	CSI
Audio outputs	3.5 mm jack, HDMI
Onboard Network	10/100 wired Ethernet RJ45
Low-level peripherals	GPIO pins, SPI, I²C, I²S, UART
Power ratings	700 mA
Size	85.0 mm x 56.0 mm x 17mm
Weight	40g

Table 2.1: Raspberry Pi hardware specification

Figure 2.2: Raspberry Pi

2.3 Raspberry Pi CSI camera

Due to Raspberry Pi having a CSI connector it can be used with a Omnivision 5647 digital camera comparable to cameras used in smartphones.

The camera module is capable of taking photos up to 5 megapixels (2592×1944 pixels) and can record video at resolutions up to 1920x1080 30fps [2].

Figure 2.3: Raspberry Pi Camera

Camera functions those are important for this work:

- Automatic exposure control.
- Automatic white balance.
- Automatic band filter.
- Automatic 50/60 Hz luminance detection.
- Automatic black level calibration.
- Programmable frame rate.
- Programmable cropping.

2.4 Assembled system

The assembled system can be seen in Figure 2.4.

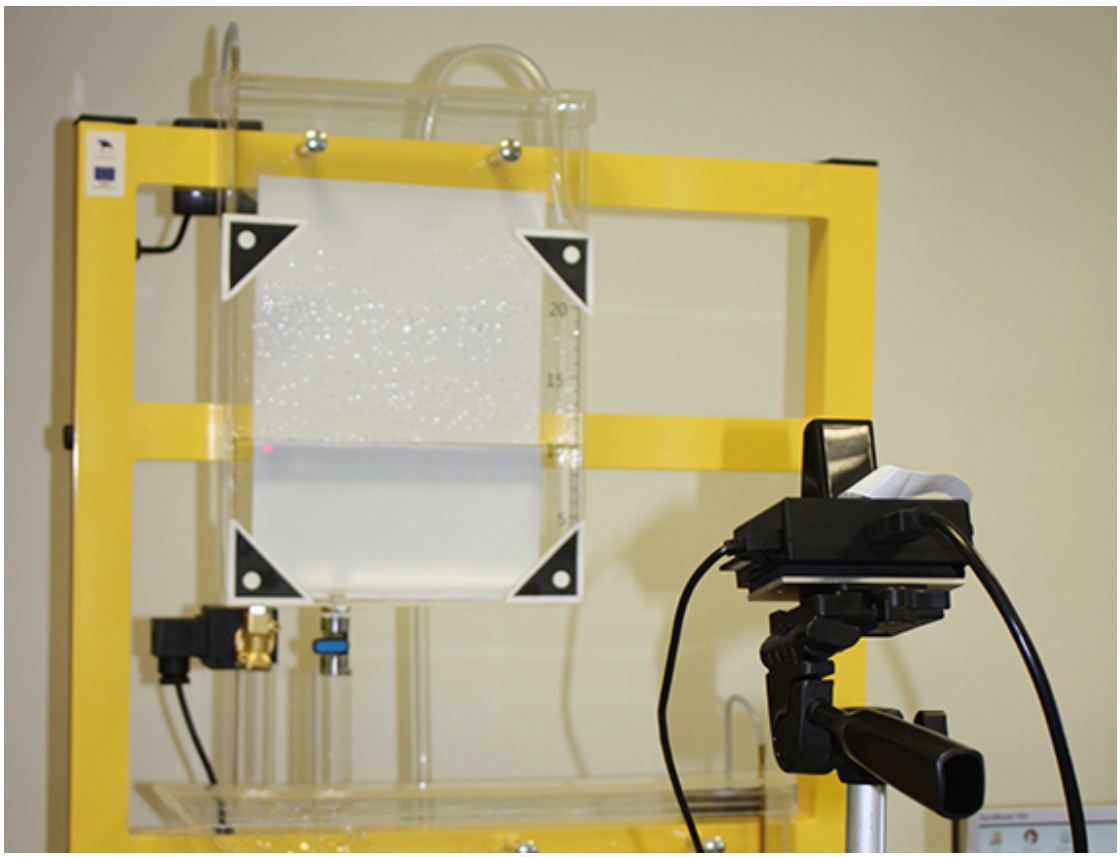

Figure 2.4: Assembled system

2.5 Development environment

Because Raspberry Pi can run GNU/Linux distribution Debian 7.4, it was decided to develop the software under GNU/Linux also. To minimize operating system installation and configuration time Ubuntu 13.10 distributive was used. Initially the software was developed under Ubuntu, compiled for ARM11 architecture and launched on the target platform. Minor differences in packet managers and systems structures, using the both systems was easy. Eclipse IDE for C/C++ Developers Version: Kepler was used for programming tasks. Eclipse main advantage is user- friendly graphical build settings and the ability to configure personal cross-compilation toolchain.

2.6 Computer vision library

Computer vision applications can be developed using software solutions like MATLAB, OpenCV, GNU Octave and many others. MATLAB presents a wide variety of instruments for computer vision [3], but in fact cannot be installed on Raspberry Pi. Furthermore, numerical computing environments, like MATLAB and Octave, cannot

be adequate solutions for the given task, since by design the will waste limited computing resources of target platform. Using this logic it was decided to use OpenCV library which is written in optimized C/C++ and can offer a satisfying performance. OpenCV is an open source computer programming library mainly aimed at real-time computer vision. It has C++, C, Python and Java interfaces and supports Windows, Linux, Mac OS, iOS and Android. The latest version can be downloaded on OpenCV Foundation website [4].

2.7 Camera driver

By default *Video4Linux* interface does not have a driver for Raspberry Pi camera. Raspberry Pi Foundation provides a several applications accessing directly to MMAL API [5]. That means camera will not be recognized by the system as standard video capture device. There are the following ways to solve the problem:

1. Modify source code of provided applications. Use buffer memory of the camera to feed OpenCV image objects [6].
2. Use in project already existing *RaspiCam* library [7].
3. Use 3th party user space *raspicam* driver [8].

We have chosen the third option because it is the least time-consuming on the early stage of development and provide a good abstraction. CSI Camera can be easily replaced by USB camera or other video device. Installation guide can be found on Linux Projects website [8].

Unfortunately, performance about two times lower than guaranteed by the author of *RaspiCam* library [7]. During tests, we have achieved a maximum of 15 FPS in 640x480 video capture resolution instead of the expected 30 FPS.

Before starting the developed application *Video4linux* module must be configured using following command:

```
uv4l --driver raspicam --video_nr 0 --encoding yuv420
--width 640 --height 480 --nopreview --framerate 15
```

Chapter 3

Development

3.1 Tank space detection

The main challenge of this thesis was determining the tank location in the picture. As it can be seen in Fig 2.4, there are a lot of foreign objects, such as background, rack and other multitank equipment. The complexity of identification is also dependent on the camera location and thus field of view may vary. To accurately determent the positioning and geometry of the tank it was decided to add 4 black triangular markers with white circles in the middle. A few standard identification methods for these kinds of tasks were tested. Ultimately, only one of them yield a satisfactory result.

3.1.1 Template matching

The basic idea of this method is to take a previously saved image of a marker and using it, identify the marker position on captured frame. The template is compare with the source by sliding it. That means moving the marker image one pixel at a time in both directions (horizontally and vertically). At each location, specified match method calculates value that represents how good or bad the match at that location is. The drawback of this method is the dependency on camera location and overall lighting. If the camera is set slightly off the markers perpendicular axis or the lighting is dimmed, determination becomes highly unlikely. Furthermore, because it is impossible to know the marker size in the picture, it is required to scale the template multiple times, before the match will occur. This in itself slows the application execution speed.

3.1.2 Hough transform

Was made an attempt to use the Hough transform based line detection algorithm [9] to locate the tank. Unfortunately the source image contains too many straight lines, which complicates the analysis. However, the resulting experience was useful during the next stages.

3.1.3 Contours detection

A viable solution to detect markers was implemented in OpenCV via contours search method, based on an algorithm created by Satoshi Suzuki [10]. Binary-converted image (black and white) is analyzed, resulting in a vector containing points in a 2D space, forming contours and their hierarchy metadata. Every contour is approximated using RDP algorithm [11, 12], to find a similar curve with minimized number of points. Approximated contours can be easily analyzed to find triangles with the circles inside.

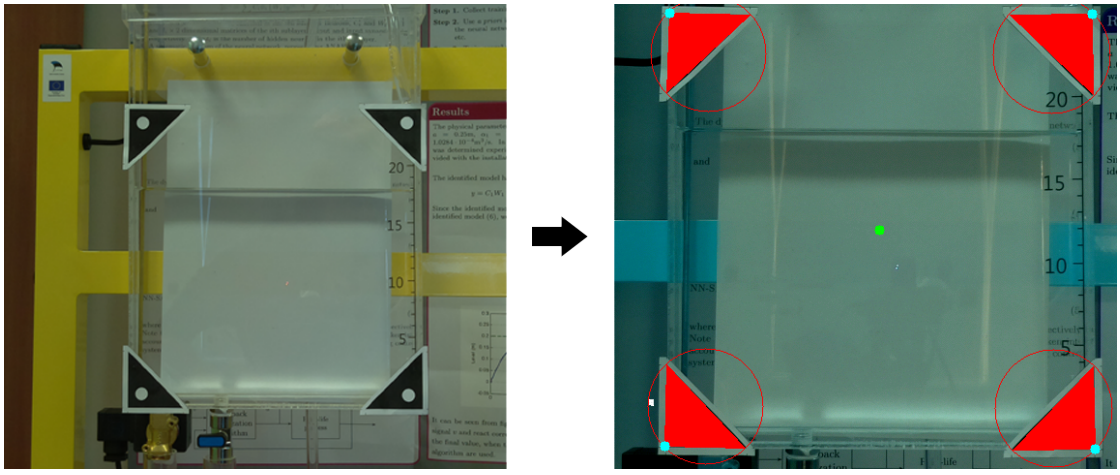

Figure 3.1: Marker detection

In the image on the right, markers are filled with red for demonstration and debug purposes. Green dot represents center of the tank and cyan points are ROI corners needed for the next step of the algorithm.

It should be noted that the colors on the left image differ from the colors on the right, this by no means affect the result of the calculations, since the all of the operations are made on a black and white image. And the color distortion is a result of encoding error.

3.2 Perspective correction

Before detecting the tank water level it, is required to link tank coordinates, acquired from the image, with the real world dimension counterparts. And to further minimize the impact of the camera angle relative to the tank. OpenCV perspective correction tools were used for this task. Corrected image size is a scaled physical tank with a specified factor.

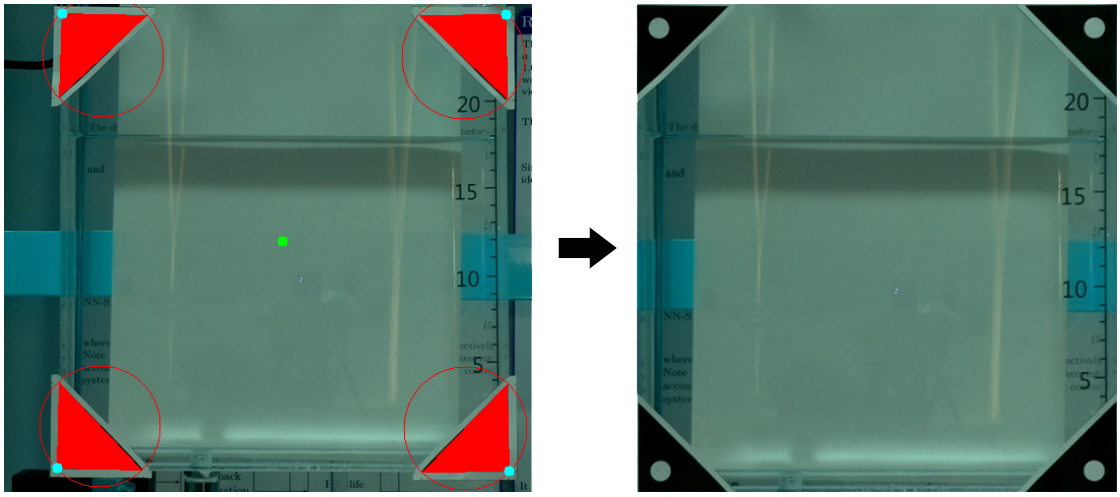

Figure 3.2: Perspective correction

This stage is a performance bottleneck. More details on this can be found in Chapter 4.

3.3 Water level detection

Water level detection consists of 3 stages. First, a ROI needs to be chosen, preferably with a clearly visible water level. Because markers overlap the corners view for further processing the middle image part is used. Next, Canny [13] operator is applied to detect edges. In Fig 3.3 water level can be seen clearly.

On this picture, for demonstration purposes, Canny operator is applied to the whole image. Region in which water could be detected is highlighted with a rectangle. Bottom margin is a result of the tank bottom being in the picture. Lastly, Hough transformation [9] is applied to find straight lines in the image. To isolate false positives and to improve accuracy overall, lines length and pitch are verified. Topmost and the longest horizontal line is taken as current water level.

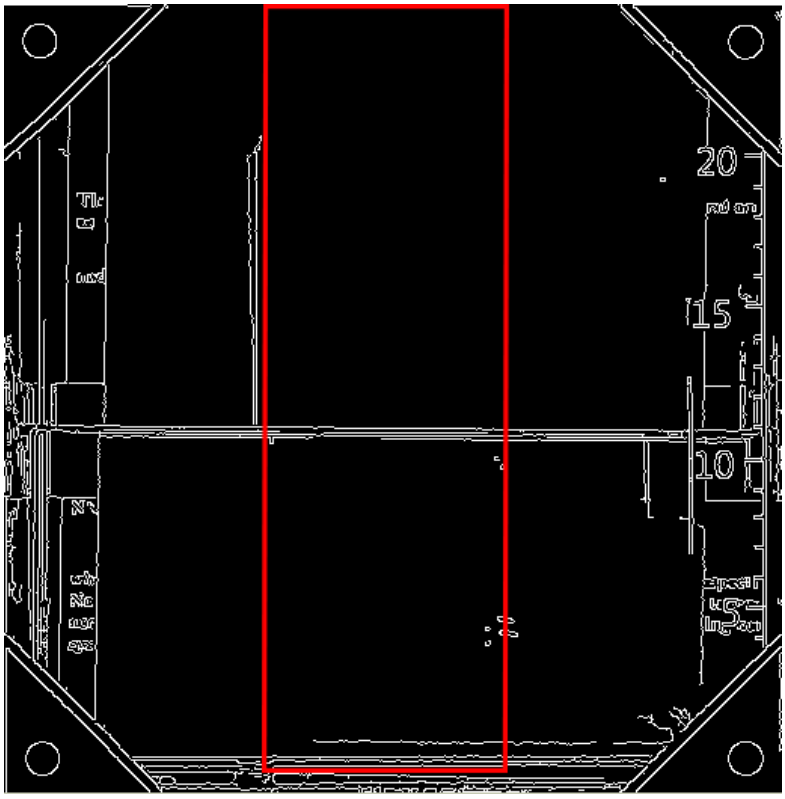

Figure 3.3: Canny edge detector

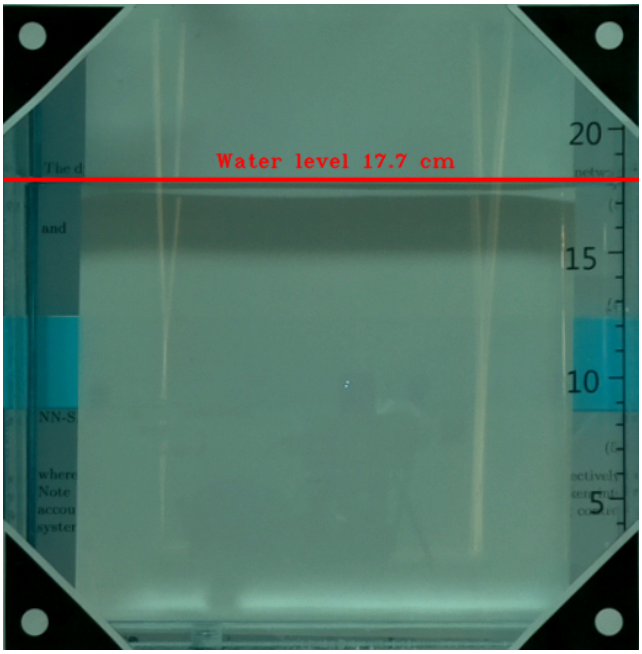

Figure 3.4: Resulting image

Chapter 4

Application

Application is a computer vision program that can detect liquid level in INTECO Multitank. Program is mostly cross platform (except UDP implementation) but for this particular work it was compiled only for GNU/Linux i686 and ARM11 architectures.

The program has 3 interfaces, command line, UDP and GUI (can be disabled at compile time).

4.1 Dependencies

- OpenCV version $\geq$ 2.4 [14].
- sendip program for UDP communication with MATLAB.

4.2 Usage

Before using the application, the user must place the camera in front of the tank and make sure that all markers are visible in the frame. For this purpose provided an argument "-dir". Captured frames will be displayed on the monitor without any processing.

```
WaterLevel -dir
```

To stop the application focus the window and press "q" or Ctrl + C in terminal.

Next step is settings regulation. Execute application in real time water level detection mode "-cam" with argument "-settings". By default parameters are configured to run

application in laboratory using the Omnivision 5647 camera. So if environment or hardware changes, settings should be adjusted. First of all ensure that all 4 markers are detected, slide the "Threshold" trackbar until tank geometry will be recognized. If the level detected incorrectly, regulate “Canny Threshold” parameters to increase or decrease sensitivity.

Running the application in settings mode is not recommended because of increasing CPU load.

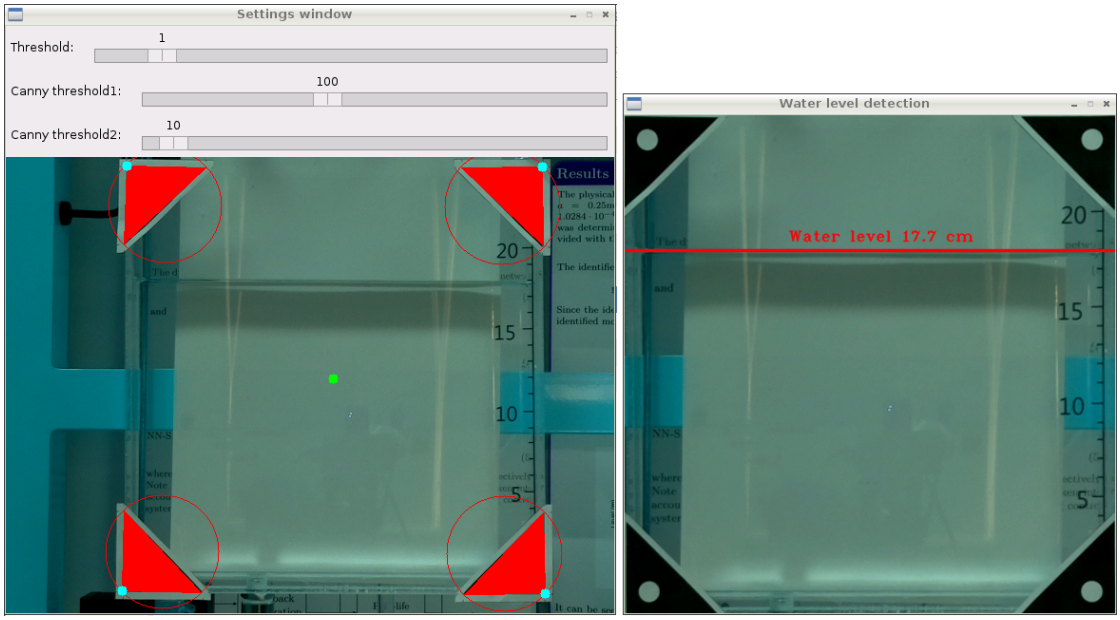

Figure 4.1: User Interface

Program can be started with custom settings using arguments “-tr “ and “-canny”. Example:

```
WaterLevel -cam -tr 2 -canny 10-125
```

4.3 Performance

Performance of the application was measured several times during development and testing.

Application can capture and display approximately 15 raw frames per second with resolution 640x480 in 3 channel RGB mode. Because during the experiment was not made any image processing, we assume that is the maximum possible performance of the user space *raspicam* driver.

The final version of application can process about 5 frames per second. Significant FPS drop is caused by contour identification and perspective correction.

OpenCV frame buffer supply algorithm with the little delay about 2–5 frames, with 5 FPS the latency is 0.4 – 1 seconds.

Performance can be improved using low level commutation between the camera and the application. Further speed gain can be achieved by offloading time consuming tasks to GPU.

4.4 Known issues

Software is very sensitive to image noise. For example, water drops on tank walls, or sun glare complicate the application execution, compared to a human, who does not have these kinds of difficulties. Furthermore, detection accuracy is very dependent on pitch of the camera and correct placement of markers on the tank surface. Observation error is approximately $\pm$ 3 millimeters.

4.5 MATLAB/Simulink Applications

To demonstrate how developed program works together with MATLAB/Simulink was developed 2 real time applications.

4.5.1 Calibration

Visual liquid level detector can be used for calibration of build-in Multitank piezo-resistive pressure transducers. Tank slowly fills up with water and then slowly drains. During this process readings are taken from the camera and tank pressure transducer. In Fig. 4.2 you can see the linear approximation of gathered data.

4.5.2 Real time control

Developed application determines the water level of sufficiently accurate and fast to control the process in real time. For that purpose was build Simulink block diagram

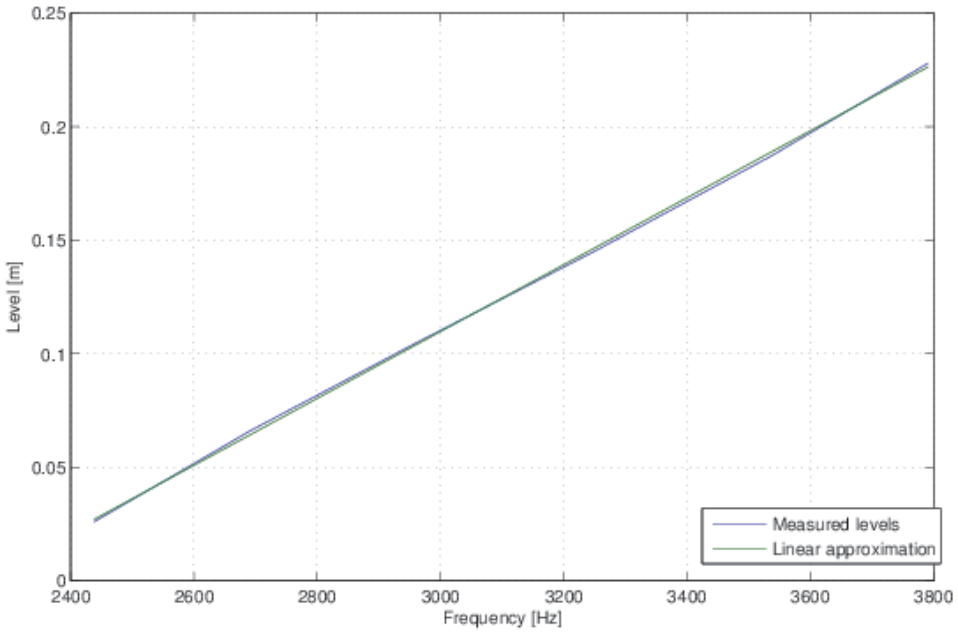

Figure 4.2: Build-in sensor calibration diagram

(Fig. 4.3) with a fractional-order, gain and order scheduled PID controller [15], where visual feedback is used to stabilize the system.

In Fig. 4.4 you can see the gaps on Visual feedback curve. They are caused by the program temporarily losing the water level line if the tank is filling up too quickly.

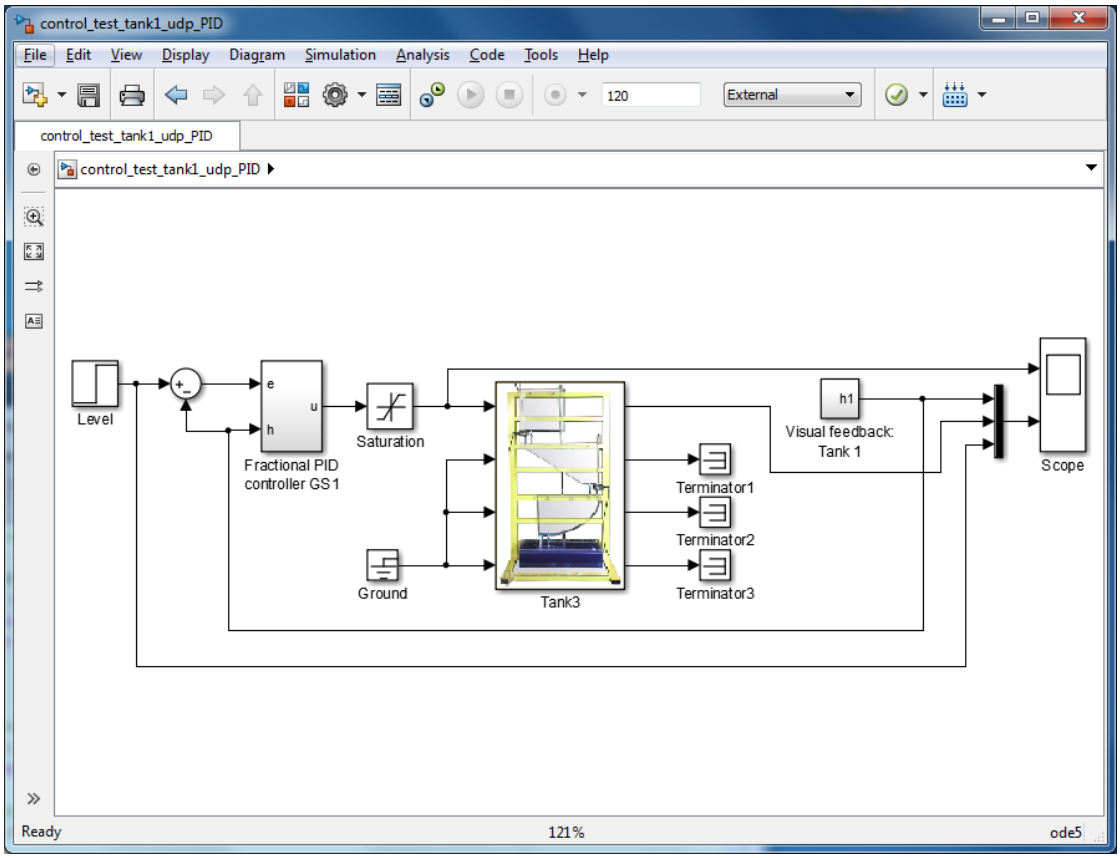

Figure 4.3: Simulink block diagram

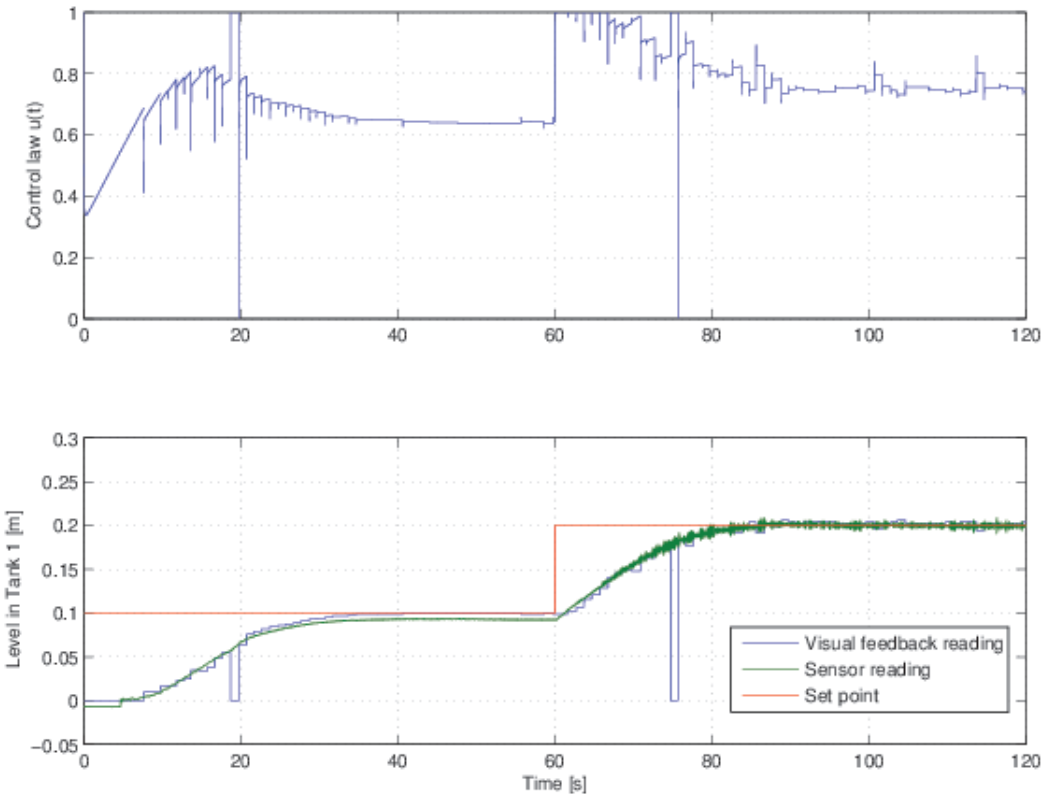

Figure 4.4: Real-time control diagram

Conclusions

This thesis can be viewed as a starting point for implementing a complex control system for real-time processes, using s visual feedback loop. The main focus was made on developing an application for an embedded system that detects fluid levels on an imaged received from a digital camera.

A few months back, when the development has started, only the final goal was set, while having no experience image processing. Because of this, a lot of time was spent to familiarize and practice with basic algorithms of computer vision.

In the end a fully working application was developed that successfully handles the given task. And thanks to MATLAB's communication interface, the software can be used to calibrate and experiment with the Multitank system.

Bibliography

[1] INTECO. Inteco multitank overview. [accessed 30.05.2014]. [Online]. Available: http://www.inteco.com.pl/products/multi-tank/

[2] OV5647 color CMOS QSXGA (5-megapixel) image sensor. [Accessed 31.05.2014]. [Online]. Available: http://ovt.com/products/sensor.php?id=66&limit=171

[3] Matlab computer vision system toolbox. [accessed 01.06.2014]. [Online]. Available: http://www.mathworks.se/products/computer-vision/

[4] OpenCV website. [accessed 01.06.2014]. [Online]. Available: http://opencv.org/

[5] Raspberry Pi camera documentation. [accessed 02.06.2014]. [Online]. Available: http://www.raspberrypi.org/wp-content/uploads/2013/07/RaspiCam-Documentation.pdf

[6] OpenCV and Pi camera board. [accessed 02.06.2014]. [Online]. Available: http://thinkrpi.wordpress.com/2013/05/22/opencv-and-camera-board-csi/

[7] Raspicam: C++ API for using raspberry camera with/without openCV. [accessed 02.06.2014]. [Online]. Available: http://www.uco.es/investiga/grupos/ava/node/40

[8] Userspace Video4Linux driver module. [accessed 02.06.2014]. [Online]. Available: http://www.linux-projects.org/modules/sections/index.php?op=viewarticle&artid=16

[9] R. O. Duda and P. E. Hart, "Use of the Hough transformation to detect lines and curves in pictures," *Commun. ACM*, vol. 15, no. 1, pp. 11–15, January 1972.

[10] S. Suzuki and K. Abe, “Topological structural analysis of digitized binary images by border following,” *Computer Vision, Graphics, and Image Processing*, vol. 30, no. 1, pp. 32–46, April 1985.

[11] U. Ramer, “An iterative procedure for the polygonal approximation of plane curves,” *Computer Graphics and Image Processing*, vol. Volume 1, Issue 3, pp. 244–256, November 1972.

[12] D. Douglas and T. Peucker, “Algorithms for the reduction of the number of points required to represent a digitized line or its caricature,” *Cartographica: The International Journal for Geographic Information and Geovisualization*, vol. 10, no. 2, pp. 112–122, October 1973.

[13] J. Canny, “A computational approach to edge detection,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 8, no. 6, pp. 679–698, November 1986.

[14] OpenCV source code. [accessed 03.04.2014]. [Online]. Available: https://github.com/Itseez/opencv

[15] A. Tepljakov, E. Petlenkov, and J. Belikov, “Gain and order scheduled fractional-order pid control of fluid level in a multi-tank system,” in *2014 International Conference on Fractional Differentiation and its Applications*, 2014, accepted for publication.