New Blood Level Measurement System in Blood Separating Machine

Miloš Petković, Miroslav Božić, Dragiša Popović, Darko Todorović, and Goran S. Đorđević

***Abstract*—Standard versions of blood separators typically use medium-price color sensors for a detection of a boundary level between red blood cells and plasma, at the last gate – at hose clamps. Discrete number of sensors is related to a number of significant levels to be detected thus making blood separation potentially faulty and unreliable. Our target was to make flexible, low cost replacement for level detection system that can be easily integrated into the existing product. We came up with an image processing solution that uses USB web-camera, ARM based off-the-shelf board – BeagleBone black and free OpenCV library. Flexibility is held in much higher, selectable number of levels, freely positioned USB camera and brand-free independent processing platform, as well as semi-automatic calibration system. By adding minimum additional electronics, we managed to integrate our solution into existing Blood processing machine. In conclusion, we added a new value to the machine at lower cost in production, increasing measurement frequency and resolution needed for improvement of blood separation process. Next step is to try to use two USB cameras on a custom-made board, for simultaneous level detection on two-channel blood separator, bringing the system integration to the higher level.**

***Index Terms*— Blood separator, red cells / plasma level detection, USB web camera, BeagleBone black.**

Original Research Paper
DOI: 10.7251/ELS1418075P

I. Introduction

BLOOD separation machines are used for separating red blood cells from plasma. The centrifuged bag of blood is inserted into machine where it is being squeezed. The bag content is drained away through two separate tubes, one on the top and one on the bottom of the bag. Red blood cells are settled at the lower half of the bag after centrifuging. When the bag is squeezed they flow through bottom tubing into another bag. Similarly, lighter plasma flows through top tubing into third bag. This method is so called top-bottom and it is common for this type of bags, with top and bottom tube openings. There are other methods for extraction that use different types of bags, with different location of draining tube openings. In order for this particular, top-bottom, method to be efficient it is important to maintain flows through top and bottom tubes equal. Plasma is not only lighter it is more fluent than red blood cells. This means that unregulated draining will cause all of the plasma to leave the bag before the red cells. If this happens, a lot of the remaining content will be trapped. That is why a constant tracking of ratio between those two liquids is needed. When the balance is lost, a clamp stops the flow of less dominating fluid. The more dominant fluid continues to run, eventually returning the balance. When balance is set again, the clamp is opened.

The ratio between plasma and red cells can easily be tracked visually. There is significant difference between them both in transparency and color. A distinctive boundary level can be seen in the centrifuged bag. Blood separators, currently available on the market, typically use medium price color sensors to detect whether red cells reached some level. This means that there must be as many sensors as there are significant levels to detect. Usually, it is eight. The total cost of this solution gets even higher if maintenance is considered. Specifically, additional drawback that comes with these sensors is periodic calibration. Increased number of sensors, due to number of levels, automatically means longer maintenance time, as well. This increases maintenance fees as well as loss of profit during halt in machine usage.

The high price, the need for calibration and somewhat longer maintenance time lead us into search for better solution.

II. Image Based Level Detection Method

We browsed the Internet for existing solutions for blood level detection in blood separators. We found no papers written on particular subject. Most blood separator manufacturers are not sharing design details of their products.

Manuscript received 20 October 2014. Received in revised form 10 December 2014. Accepted for publication 15 December 2014.

This research and paper were funded by Ministry of science, education and technical development, through project III44004.

M.Sc. Miloš Petković is with the Faculty of Electronic Engineering, University of Nis, 14 Aleksandra Medvedeva, 18000 Nis, Serbia (e-mail: milos.petkovic@elfak.ni.ac.rs).

Miroslav Božić is with the Faculty of Electronic Engineering, University of Nis, 14 Aleksandra Medvedeva, 18000 Nis, Serbia (e-mail: miroslav.bozic@elfak.rs).

Dragiša Popović is with the Faculty of Electronic Engineering, University of Nis, 14 Aleksandra Medvedeva, 18000 Nis, Serbia (e-mail: gile.sherif@gmail.com).

M.Sc. Darko Todorović is with the Faculty of Electronic Engineering, University of Nis, 14 Aleksandra Medvedeva, 18000 Nis, Serbia (e-mail: Darko.Todorovic@elfak.ni.ac.rs).

Prof. Dr. Goran S. Đorđević is with the Faculty of Electronic Engineering, University of Nis, 14 Aleksandra Medvedeva, 18000 Nis, Serbia (e-mail: goran.s.djordjevic@elfak.ni.ac.rs).

Also no particular patents were found on online patent sharing sites. However, we found quite a bundle of papers for liquid level detection in other applications, as well as some commercial sensors. There are several companies that make industrial liquid level measuring sensors, based on optical technologies. Whether they use laser reflection or some other CCD camera method, they are made for industrial purposes and harsh operating conditions. E.g. for casting industry where temperatures rise up to 2000^{o}C, on the contrary, room conditions where blood separators work are quite regulated, due to conventions for good blood preservation. Stated make industrial, off-the-shelf sensors quite expensive and thus in conflict with design goals. Other inconvenience with most of liquid level detection sensors is that they are mostly used for detecting levels in tanks, not small, flexible containers like blood bags. This primarily means that sensor is placed on top of rigid container and use some sort of contactless distance measuring. Without emphasizing any particular solution we will reference following [1-6]. In [1-4] they are using optical methods with camera to measure level in tanks. Although not quite same as what we did, our solution could be described as their mixture, as it uses similar concepts. [5] and [6] are referring to liquid level check in bottles. Although they concentrate on smaller liquid level variation and detection of badly filled bottles they concern with methods that are of interest for our solution. [5] gives quite nice comparison of different image processing methods and their accuracy. However none of the work of others, that we have found, was done for liquids in bags.

The solution that we came up with is image processing based as well. The basic idea is to take video of blood bag, during the separation process, with commercial USB camera, and then do image processing, on some mini PC board, in order to determine boundary level. The camera itself would be placed inside the machine, facing toward already existing small window on the side of machine. The current purpose of this narrow window is to make visual contact between color sensors and the bag. When the sensors are removed there is quite enough space for camera to be placed. Furthermore, body of machine is quite deep, without any parts in the window region. So the camera can be placed far enough from the window to capture it wholly. The decrease in needed distance is also due to wide viewing angles of web cameras. However we used this extra space to leave possibility of slight misalignment between camera focal point and center of the window. So the camera image is set to capture quite an area around the window. This surrounding area appears mostly black, since very little light breaks into machine. This creates quite good contrast between region of interest, which is the window, and the rest of image. After all initial feasibility checkups of the use of a generic web camera gave positive feedback, we went on with image processing algorithm and processing platform selection.

We considered it convenient to use some off the shelf mini PC platform for image processing. Mini PC platforms offer possibility of programming with higher abstraction languages and easier code portability. They are also available in quite a number in terms of processing power and other features. We chose ARM based board - BeagleBone Black [7]. Primarily cause it was already at our disposal. Another reason is that there is additional plug-in board (cape) with good video camera, particularly for this mini PC. Later performance comparison of this camera cape and a USB camera showed no significant difference. Since latter is much cheaper, but equally suitable, we based rest of the research and the solution on it.

The goal for image processing software was to be as portable as possible. We decided that OpenCV library is therefore a good choice [8, 9]. It is a free library that contains a lot of implemented image processing algorithms, and thus reduces design time. It is highly portable. It offers initial development and testing of image processing algorithm to be done on PC, which is faster than on some mini PC. Latter, only simple code recompiling is needed for it to run on other platforms like Beagle Bone Black (BBB). As source code of library is available, it can be made, with more or less effort, compatible with any platform. However, already compiled shared library versions are available for most known mini PC platforms. Luckily BBB is one of them, which saved us some time. Shared libraries are even installed on official OS. It is Angstrom (Ångström) version of Linux. There are other OS available, like Ubuntu or Android, but we chose to keep using the initial one, which is farcically pre stored on BBB on-board flash memory. We would like to point out that previously mentioned camera cape requires some of processor pins otherwise used for communications with flash memory. Therefore, when camera cape is used, OS should be put on and then loaded from external micro SD card. Although we used official release of Angstrom for SD card, we have noticed, during camera cape testing, that it is less stable than the flashed one. This inconvenience, altogether with already stated in previous paragraph, put us off the camera cape use.

A. Image processing

As mentioned before, our region of interest (ROI) is the narrow vertical window on the bag side of a blood separator. Since both the camera and the window are stationary to each other during machine run, we decided to use static approach to extraction of ROI. In other words, ROI always has the same position in acquired image. So there is no need to constantly run ROI finding algorithms during separation process. This greatly reduces necessary processing power of underlying hardware. Opposed to that, higher number of processed video frames, or lower response time, can be achieved.

Every time blood separator is started ROI coordinates, in terms of image coordinating system, are found. This way we wanted to ensure that possible slight movements of camera do not reduce measurement accuracy. Potential cause of camera twitch could be for example rougher machine handling during transportation. At least, our initial shoddy camera holder was susceptible to this. It was replaced later on with better one, but we kept on startup ROI initialization as security measure.

Anyhow, extraction of ROI coordinates should be fast and solid. Although there is high contrast between window and surrounding, we noticed its fluctuations on window boundaries. As said before, very small amount of light enters the machine. However, above the window there is a strain gauge mass measuring sensor, which aluminum body reflects and scatters light. So, upper side of the window appears pretty irregular. Other sides also shimmer a bit, from time to time. We decided that it is best to add fixed frame with four corner LEDs onto the window. They are solid markers for rectangular ROI determination. This solution is less ambient lighting dependent and thus less error prone. LEDs are the brightest objects on the image and therefore easily extracted, as it can be seen in Fig. 1. If any additional bright enough sources appear, they can be discarded by taking into consideration positional relation of the LEDs to each other. This makes calibration absolutely accurate.

In order to extract the LEDs position from image, the following procedure was used. Camera parameters are firstly set to low exposure and high contrast. Then an image is acquired. In the next step, erode, dilate and blur functions are used to eliminate noise and light dots from grabbed image. This filtering leaves all-black image with four white circles. At the end, we used Hough Circle Transform [10] to detect position of white circles. Unfortunately we had to change this algorithm a bit. The USB web camera, that we used, has some auto white balancing option that we couldn't disable. This means that although very low exposure was set, the camera itself brightens the image, so it looked like in Fig. 1. Note that image is rotated for 90 degrees counterclockwise. The window itself is vertical, but the camera is in portrait position. It was set this way in order to match window elongated height with width of 16:9 resolution camera. During development we rotated image back to vertical position. Later on we left this code out in order to avoid unnecessary loss of time. If image from Fig. 1 were submitted to previously described algorithm, more than 4 white circles would remain in the image. This means that selection of circles would have to follow. We thought that this will increase ROI initialization time and went with different approach. After adjustment of camera parameters, a frame with all LEDs powered off is taken. Then another frame is grabbed after the LEDs are light. Difference between two frames is found, which leaves very few artifacts for removal. After filtering, that is same as in originally planned algorithm; image looks like in Fig. 2. It is further processed with Hough Circle Transform in order to find coordinates of white circle centers. It should be stated that OpenCV functions were used for all mentioned steps like erode, dilate, Hough circle transformation, etc. The last numbered function, returns both center coordinates and diameter of recognized circles. How precise this information is, can be seen in Fig 3. Black dots with coordinates of LED circles were overplayed over image from Fig. 2. It can be noticed that they match the centers of LED circles with quite an accuracy. The rest of white circles are covered with empty black-line circles. They have same diameters as the ones of

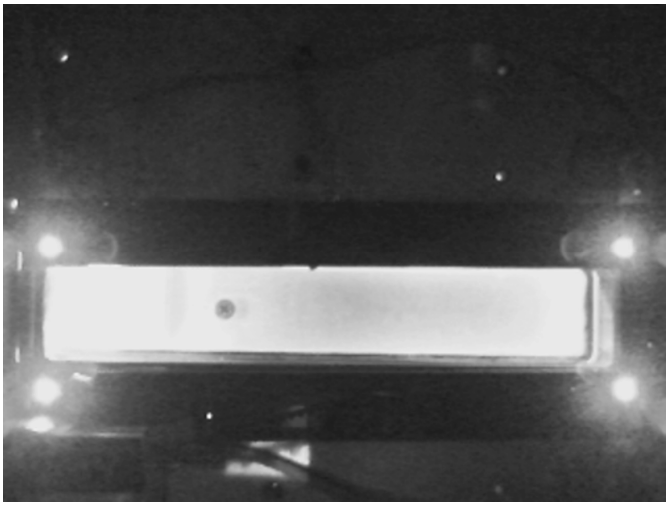

Fig. 1 Unprocessed image from camera, LEDs are turned on. Note that image is rotated for 90 degrees, the window is normally vertical. This is due to camera orientation.

Fig. 2 Image of turned on LEDs, after reference frame subtraction and filtering. Reference frame is taken just before LEDs are light.

Fig. 3 Found coordinates of LEDs, designated with overplayed black dot in center. The rest of white circles are covered with empty, black line, circles to designate recognized sizes of white circles.

LED circles approximated by the function. This information was used just for the purpose of evaluation of quality of image filtering and precision of OpenCV functions.

After window coordinates are known, other pre-processing transformations can be applied. We used only rectification, as we believed it improves accuracy of the system. Finally, ROI can be extracted for further processing by a boundary level detection algorithm.

ROI is extracted from camera image while settings are close to default ones, typically used with normal ambient light. We lowered brightness only a little. Since center of LEDs are slightly away of the window corners, we have reduced ROI width for arbitrary number of pixels. This doesn't affect measurement accuracy. We don't need all the pixels from the window's width to determine boundary line, as it will be described later in more details.

Image of real ROI can be seen on the right in Fig. 4. Ignore the solid horizontal full and half line, for now, as they were overplayed in post processing. The lower portion is red as transparent red foil was placed over outer side of the window. The upper part is partial view of our laboratory. It is blurred by semitransparent blood bag that was also placed at exterior side of the window. During development, and later on during testing, this setup was used as a replacement for real bag with blood. We considered it as quite a good substitute. It should be noted that this image was acquired on PC. From within the code, a real time stream of ROI was being shown on monitor. This is something that is not possible to achieve only with OpenCV functions on BBB. There seems to be lack of their proper implementation for this platform. The image was saved via snapshot tool from within Windows. It is also pre-rotated in oppose to Fig. 1.-3.

First step in level detection is to find all pixels in ROI that belong to red blood cells part. Since it appear as the reddest part in the image, only single threshold level needs to be determined. All pixels that are red above this level are considered to belong to red blood cells portion. After threshold was experimentally determined, OpenCV function "threshold" was used to extract these pixels. The result is shown on the left in Fig. 4. All pixels that are considered red as red blood cells are shown as white. Others are black. Due to lighting conditions, there can appear some black pixels in white region. We used erode function to remove those rouge black pixels.

Next step is boundary level search itself. It is determined according to the average value of horizontal lines. Rapid change in these average values is a good descriptor of the boundary line. Initially, we calculated average value of entire row. However, we noticed that boundary line was slightly curved at the edges of ROI. Filters and image preprocessing that we used cause curvature. In order to exclude this source of error, we reduce average pixel calculation to middle 60 pixels, thus excluding pixels at window edges. However, later on, results showed that there is no significant increase in accuracy. This ROI reduction also reduces the processing time. Finally, rapid change in these average values is searched in order to determine exact boundary level location within the window. Again, there is quite distinctive difference in plasma and red cells color.

After boundary level was found for image on the left, it was displayed as overplayed gray line over original ROI image in Fig 4. It could be seen that it is determined with solid accuracy. The thin black line, which appears above gray line, comes from boundary line blurred reflection. In other words, no web camera is able to show edge line with ideal contrast. So edge lines will always appear as blurred transitions from one value to another. So we take middle of the edge as an actual boundary. The other overplayed half lines designate current position of color sensors. We draw them just to show how big the increase in resolution can be achieved with our method.

Fig. 4 Level detection on processed image. Left image shows recognized red blood cells portion of bag with white color, and rest of pixels in black. Right image shows real ROI image with designated found boundary line. Horizontal half lines mark the current position of discrete color sensors and gain in accuracy in this method.

We are able to determine boundary level position virtually within accuracy level of about one image line. This is much

higher resolution than with color sensors. However, since our system is to be integrated into existing machines, level conversion is needed. Output from our system is at end with eight significant levels.

B. Software implementation

As previously said, OpenCV library is used for image processing. In particular for fetching camera frames, setting camera parameters, image rotation and rectification, circle (LED) position extraction, inverting image color and detecting blood level. All image-processing code is divided into two functions. One is initialization and calibration function. This function is called once on every machine start up. It accepts as input a *mode* constant. If it is called in service mode then a checking of ROI is performed. In other words, when camera is initially inserted it needs to be faced properly toward window. All four corner diodes must be visible in image. If any of 4 LEDs is not visible due to camera-window misalignment, function will signal this through second argument. In reality, this information is communicated back to camera installer through four LEDs on accompanying board to BBB. Distinctive relation is made between visibility of corner LEDs and displaying LEDs on board. If one corner LED is out of image then corresponding board LED is not shining. When all four board LEDs are turned on then camera is correctly in place and service mode can be exited. When the same function is executed in calibration mode, ROI coordinates are being calculated as previously described. Also, significant middle and other levels are determined, as well as other pre-processing parameters. Separate image processing function takes frame, applies preprocessing like ROI extraction and rectification, and searches for boundary level as mentioned. Function returns detected level.

Other part of software is initialization and communication code. At startup it initialize BBB used peripherals. Altogether, 7 GPIOs and one UART are used. One GPIO is used as input and others as output. Input pin is connected to pull down pushbutton and is used at machine startup to make program run in service mode. If pushbutton is pressed during software initialization phase then service mode is activated. Otherwise it is not checked. So initialization function reads state of input pin only once at beginning. As previously said 4 outputs are used for driving signaling LEDs on accompanying board. The same outputs are used later for signaling level to control part of machine. One output is used to drive latch enable on those 4 outputs. Latch is inserted between BBB and machine control part in order to prevent false code readouts during level change. Last output is used for controlling window LEDs. They are on only during service mode and calibration. Last peripheral, UART, is used for communication with blood separator control unit. A RS232 to RS485 convertor IC is used to interface these two. BBB UART RTS pin is inverted and used as input to IC control pin for direction switching. IC is placed on accompanying board. This board contains all numbered additional electronics. It was designed as plugin board for BBB. If all peripherals are successfully initialized two main threads are created. Otherwise, error code is displayed on the 4 LEDs. Error codes, as well as status and other important info are being constantly sent to BBB UART 0. It is BBB default stream, and used normally for debugging. We used it for this purpose as well.

The main program basically consists of two threads. One thread is image processing thread. It is infinite loop that calls the image processing function that returns level. The following thread code stores returned value into global level variable and call digital output refresh thread. Digital output refresh thread has short life cycle. It just refreshes state of 4 output pins and dies.

Second main thread is communication thread. It is plain infinite loop that listens for any incoming data from the RS 485 communication conversion IC. When it receives message that is with address of level detection system, it process it and responds. Basically there are two types of message. One is address setup message, and other is level query message. We implemented this according to current machine control unit protocol. Protocol itself allowed us to send 3 bytes of arbitrary data. It was quite sufficient. We used one byte for level, one for system status and third one for error codes.

Whole program was written in C++. Image processing code was developed and tested initially on desktop PC. It was afterwards recompiled and linked with OpenCV libs that were already on BBB Angstrom distribution. We found that these libraries do not support some OpenCV functions, like viewing camera stream in live desktop window. Those functions are primarily important for image processing algorithm development. So, desktop PC was necessary for development of image processing part of code. Rest of program code development was done on desktop PC, but with cross-compiling and remote testing. In other words, code was written in Eclipse IDE with cross-compiler and remote system plugin. When code was compiled it was run directly and debugged remotely on the BBB. Remote debugging did not go that smoothly. It failed from time to time for some reasons.

III. TESTING

System evaluation with real blood has too high price tag for primary testing. It's not just the price of the blood and their somewhat unethical accidental wasting. The blood needs to be kept and handled in a special way. It decays after prolonged exposure to ambient conditions. It would also be needed to re-centrifuge once used blood bag. Due to lack of funding, proper storage and processing machines, we tested our system using red foil placed over bag filled with water. On image, they looked similar enough. At worst case, the system would require only minor reconfiguring before being fully operational. This primarily means setting some new value for the threshold level of the red color detection code.

Test showed that level detection works pretty well even with just room lighting. Nevertheless, we still implemented separate light source control. As far as time is concerned, it takes about 230ms to process small 360p image. For higher

resolution picture, 720p, it even takes 1.2s. The source of long processing time is image rectification. It requires approx. 200ms for 360p image. This was unacceptable and we were forced to remove it. However this only impacts, negatively, cases where camera is highly misaligned with side window. Such circumstances can easily be avoided if fixed camera holders are used, that align window and camera. That way less powerful processing platforms like BBB will suffice.

After rectification removal, processing time was approximately 30ms for 360p frame. For 720p resolution it got up to 70ms. This is still less than 0.1s what was the upper limit set by blood separator manufacturer.

The time itself was measured by comparing BBB system time before call of the image processing function and after it returned detected level. Other part of code took no significant time consumption in overall system cycle.

It is worth noting that processing time could be even faster if direct frame grabbing from camera was done. We had another bad experience with OpenCV and BBB. During initial feasibility check OpenCV was communicating with web camera properly. However after overall code porting and system integration, we suddenly came across their incompatibility. E.g. frames could not be grabbed and some camera settings were inaccessible. Because of that and lack of time we had to perform a quick fix. We managed to set camera properties by executing shell commands, from code, that invoked Linux default driver. In other words, our program calls system program that sends commands to default driver. This can be avoided by using system library for direct communication with default camera driver. This would reduce code flexibility to Linux systems only. However most of mini PC platforms are Linux based anyway. In similar manner we managed camera frame grabbing, too. One possible reason for later inoperability between OpenCV and BBB could be driver update. Also recompiling of the library for BBB could solve the problem. However at the time of testing we had no time to determine the correct cause and the later solution worked.

IV. Conclusion

We managed to implement image processing level detection system in blood separation machine. It well surpassed requirements set by blood separators manufacturer. In other words, we accomplished level detection at more than 10 times per second, which is more than enough for the process.

Apart from reducing the system costs, by replacing discrete color sensors, we managed higher detection precision. Near the fairy end of bag squeezing, color of the part with red blood cells tends to lose its strong red color. It becomes somewhat pale. When this happens, color sensors with fix threshold, detects it as plasma instead as red cells. Since process is ended much earlier, approximately 60g of content is left in the bag. Although remaining content is being collected and re-separated, some blood processing companies sees this as great waste. They buy exclusively systems that generate below 30g of remnant. Our solution doesn't necessarily use fix threshold. We believe that, with proper tuning, it can be made to detect proper boundary level in both cases. In order to accomplish such feats, we would need to get into second, more expensive, phase of testing. This would require use of the real blood.

Another development step is to try to use two USB cameras on a custom-made board, for simultaneous level detection on two channel blood separator. This would bring the system integration to the higher level.

After feedback from manufacturer about our system, we concluded that it would be a really good idea to make a black box shielding. It should guard the conical region between the camera and the window, at least. There are various reasons for this. It would prevent manufacturers to block this empty space with some other inner component, or prevent a loose wire to accidently do the same. Black non glossy shielding would ensure better image contrast. Currently, contrast is reduced when the machine side is dismantled. Maintenance personnel would like to be able to run the machine in this state during troubleshooting and testing. At present state of our system integration, this would enable ambient light to reflect from various inner components into camera area and possibly cause false reading. At last, integration of our system under one case would make it truly the component for boundary level measuring for blood separating machines.

Finally, we are still determined to improve our algorithm. We removed rectification to achieve shorter processing time, but at the cost of flexibility and self-calibration. It is desirable to keep these two characteristics and we already have a few ideas of achieving them. However, our solution is still quite unique, there are no similar for particular purpose, as far as we know, and it beats currently used design in both performance and price.

References

[1] T. Wang, M. Lu, C. Hsu, C. Chen and J. Tan, "*Liquid-level measurement using a single digital camera*", Measurement 42, Elsavier, 2009, pp. 604–610

[2] C. Yu, X. Tian, Y. Zhang and Y. Bai, "*Liquid Level Measurement by Using an Image Method*", in Proc. MACE Third International Conference on Mechanic Automation and Control Engineering, 2012, pp. 1595-1598

[3] Y. Na, S. Kim, J. Kim and J. Lee, "*Development of liquid level control system using web cam*", in Conf. 11th International Conference on Control, Automation and Systems (ICCAS), Gyeonggi-do, 2011, pp. 1527-1528

[4] I. Gil, M. Amparo and M. Pozo, "*High resolution simultaneous dual liquid level measurement system with CMOS camera and FPGA hardware processor*", Elsavier, Sensors and actuators A: Physical 201, pp. 468-476

[5] K. Pithadiya, C. Modi and J. Chauhan, "*Selecting the Most Favourable Edge Detection Technique for Liquid LevelInspection in Bottles*", International Journal of Computer Information Systems and Industrial Management Applications (IJCISIM), vol. 3, Mirlabs, 2011,pp. 034-044

[6] Suntory Limited, "*Method and device for detecting average liquid level in a bottle*" U.S. Patent 4733095 A, March 22, 1988.

[7] *BeagleBone Black System Reference Manual, Rev A5.2*

[8] *OpenCV library*, v 2.4.8

[9] G. Bradski, A. Kaehler, *Learning OpenCV: Computer Vision with the OpenCV Library*, 1st ed. Sebastiapol, CA, USA: O'Reilly Media, Inc., 2008.

[10] D.H. Ballard, "*Generalizing the Hough transform to detect arbitrary shapes*", Pattern Recognition, vol. 13(2), Elsavier, 1981, pp. 111–122