Assignment 2: Final Report

Interaction Technology (B3IT)

Egor Dmitriev

July 7, 2018

Abstract

This paper features an extended discussion on the design of an automatic plant watering device with capabilities to act on its own or the use of 2D visual gesture recognition.

Contents

1 Introduction

In this paper the design of the automatic plant watering device is discussed. First in section *Embedded System Design* will be discussed how the device is physically built and the functionality of it's firmware. Then we will discuss the best way to implement 2D gesture recognition using computer vision. In the final section the tests will be discussed which were performed to test the device.

2 Embedded System Design

This section explains how to create the automatic plant watering device, which is intelligent enough to keep itself the plant alive, while still allowing user input and providing insight on the condition of the plant itself.

2.1 Hardware

Materials used for the hardware design for the device are from the Iot kit.

Amount	Material
1	NodeMCU ESP8266 development board
1	Soil moisture sensor
1	BME280 Pressure, temperature and humidity sensor
1	OLED display, 0.96, 12864
1	Micro servo
1	Custom Analog multiplexer SMD
1	LDR Light dependent resistor

To create the final device the OLED display and BME280 sensor were connected to the ESP8211. Pins number D5 and D3 were used to connect them the SCL and SDA pins which are used to run the I2C protocol.
After that the LDR and soil moisture sensor are both attached to the custom analog multiplexer board which itself is connected to analog pin 0 and the digital pin 4 to switch between the two sensors.
Finally the servo is connected the ESP8633 with the digital pwm pin 6 as control pin.

The device used by moving it beside the plant so the temperature, air moisture and light exposure are similar to what is measured around the plant. The soil moisture sensor is then carefully placed inside the plant pot so no roots are hit. The servo is attached to a straw which is connected to a filled water bottle. The straw should attached be at 12 hours angle when its resting and directed to the plant pot so when the straw is turned 90 degrees it will be right above the soil the plant is in.

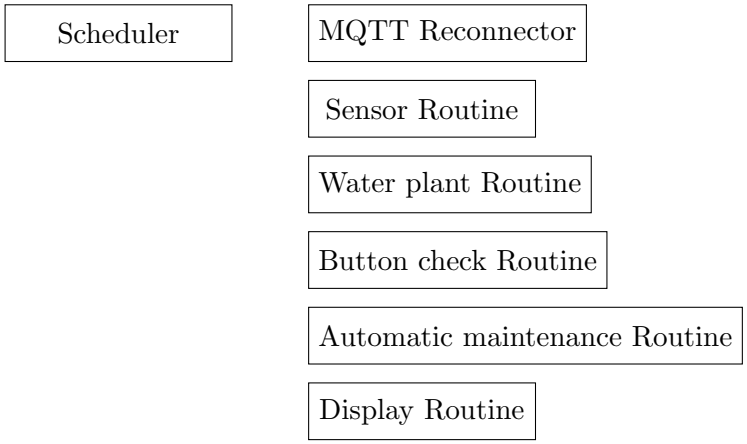


Figure 1: Architecture of the device

2.2 System Architecture

While designing the architecture for the firmware of the device extensibility and non blocking task execution has played a played a huge role. This is needed because the OLED screen, MQTT client and WIFI need to be able to run in the background. Therefore the created scheduler library which was used for the previous assignment was reused.

2.2.1 Scheduler and tasks

To add extensibility to the firmware I have split every function of the device into a task which is either scheduled to run at an interval or when certain conditions are met. A high level representation of the used structure can be seen in figure 1. Because every task is scheduled to run at a certain interval no delays are required and system is thus non blocking.

MQTT Reconnector Routine is responsible for keeping the MQTT connection up to date. If every five seconds the task is run to check if the connection with MQTT broker is lost. On connection loss the device will try to reconnect. If device still could not connect successfully the process will be repeated after another 5 seconds.

Sensor Routine is scheduled to run every two seconds to poll the sensor measurements and publish them using the MQTT client. Measurements will also be stored in the store where all the tasks can access the values.

Water plant Routine is run only when it is requested to by other tasks or a request of MQTT subscription handler. It is responsible for the whole plant watering cycle. It can run a customized amount of time. 50% of the watering time will be spent lowering and holding the watering straw down using the servo motor. The other 50% will be spent raising the straw back up.

Button check Routine is used to check if the on board flash button is pressed. If it is, then the automatic mode will toggled. The result will be published using MQTT client. Because the task is run regularly but not every tick, this also acts as a debounce.

Display Routine is responsible for displaying and animating different slides five seconds. There four different slides of which three cycle in a loop and display measurements and device state, while the fourth one only becomes visible when device is watering the plants and contains a progress bar to indicate its state.

2.2.2 Stores

Every task can access the other tasks to invoke some kind of action of read it's state. For more global information **store** is used to write data to. There are two kind of stores used:

Config *(Config store)* which contains all the data which is configured via administration panel which can be accessed by vising the IP address of the device on port 80. Config store is synchronized with EEPROM and is therefore preserved if device is turned off.

Sensor data *(Sensor data store)* contains all the sensor measurements and relevant data like timestamp when the plants last have been watered.

As stated earlier the administrative UI (figure 2) is used when to adjust configuration. On initial start if no last connected WIFI network is found the devices goes into **Access Point** mode and allows user to fill the new WIFI information first. After thet the full configuration menu is shown where user can adjust various settings like MQTT credentials. On save settings are sent via **REST** requests to the device.

2.2.3 MQTT client

An external library is used to maintain connection with PubSubClient. Connection maintained by **MQTT Reconnector Routine** task.

On every connection request **last will** is attached to the headers by the task. This last will sends a zero character payload to the topic "... /status/alive"

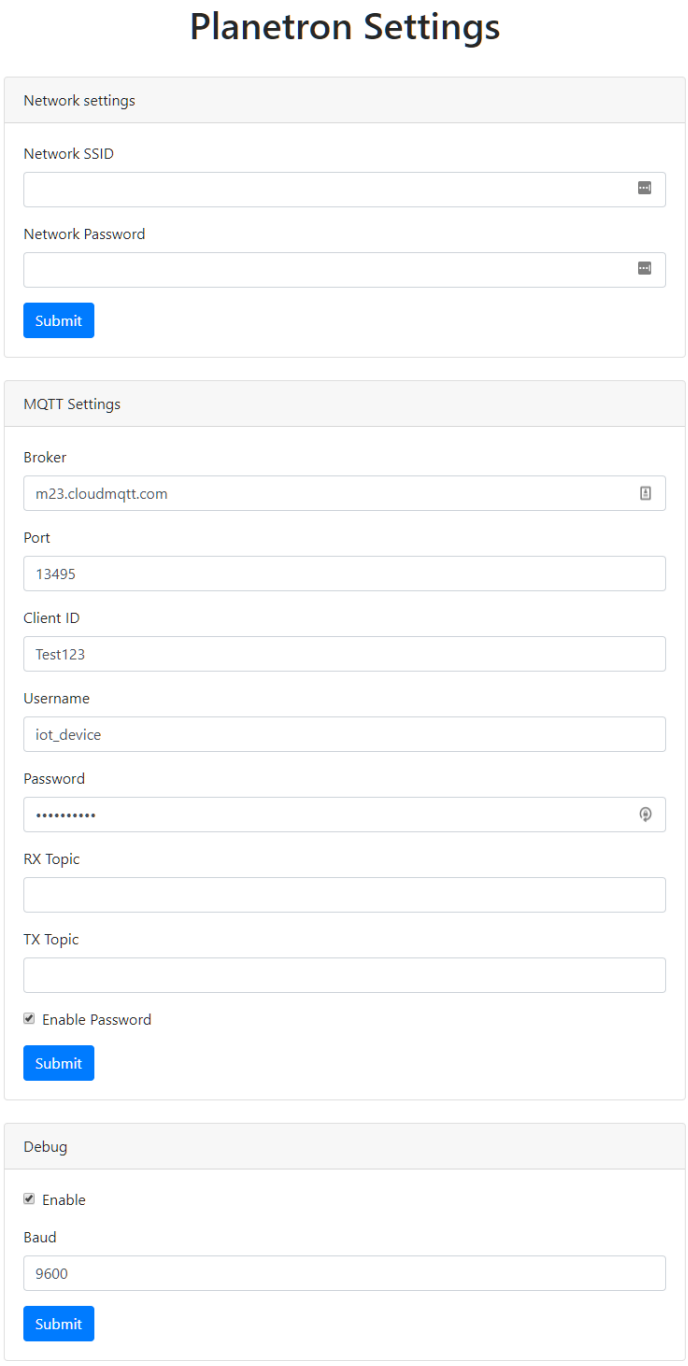


Figure 2: Admin panel

with retain flag as true and quality of service 2. By doing this broker notifies everyone subscribed that given device is offline. Naturally, when the device is connected it will publish a character one to the same topic and retain flag to let the subscribed clients know that it is alive.

Aside from bragging about it's sensor measurements the device is also subscribed to multiple different topics on connection including "plant watering", "force refresh measurements" and "toggle automatic mode" topics. Callback for this topic is also inside the scheduler but not as a task. Instead its a callback function attached to the scheduler instance which gets called by MQTT client.
In the callback function the topic and its payload are matched to the given intent. Additionally if intent is to toggle the automatic mode on or off the built in led gets also updated to the corresponding state.

Topic naming is quite intuitive and always uses *"devices/assignment_device/"* as root to maintain extensibility if more devices are needed. From there there are two topic groups *"interaction"* which contains all the topics which are interactive like topic to start watering the plants and there is a topic group *"sensors"* which is root for all the sensor topics which device publishes. A example few topics would be:

devices/assignment_device/sensor/temperature
devices/assignment_device/sensor/alive
devices/assignment_device/interaction/automatic_mode

2.2.4 Libraries

A few external libraries were used for this assignment. First of all the platformio build platform was used to create and compile the project. Instead of using sketches it compiles standard C++ .cpp and .h files which makes code easier to see (**Note: because of this code is submitted instead of the sketch file. I hope that is not an issue. It can easy be compiled by installing: https://platformio.org/ and pointing it to platformio.ini**). Also it downloads the needed libraries stated below.

EEPROM

PubSubClient For mqtt support]

Wire To use the arduino sensors that work with IC2

ESP8266_SSD1306 To use the OLED screen

Adafruit_Sensor To use the multisensor for temperature and pressure. Its a dependency for Adafruit_BME280_Library

ConfigManager To automatically reconnect to wifi and act as server to display admin panel.

Adafruit_BME280_Library To use the multisensor for temperature and pressure

DNSServer Dependency for ConfigManager

Servo To control the servo

2.3 External integration

As stated in the assignment requirements some external tools are also integrated in the whole device stack. The broker is being hoster on the MQTT cloud.

2.3.1 Node Red

I have created a Node Red flow graph to create a MQTT client which can interact with the device and read it's measurements.

Flow graph starts at the entry MQTT node which is subscribed to all sensors and interactive topics using the wildcard # *"devices/assignment_device/#"*. Then the output of MQTT node goes into the topic switcher where different topics are matched and their payloads are passed to the correct output. At each output there are UI nodes which are from "node-red-dashboard" library. These nodes display the input data. The flowchart can be seen in figure 3 and the dashboard in figure 4.

2.3.2 MQTT.fx

The MQTT messages can also be monitored from desktop pc with a MQTT client. MQTT.fx in this case (figure 5a).

2.4 MQTT buddy

The MQTT messages can also be monitored from mobile with a MQTT client. MQTT buddy in this case (figure 5b). It is interactive and can display control the device like watering the plants and mode switching with a click of a button.

3 Computer Vision

Second part of the device is a client with a running program which can perform gesture recognition on given webcam video feed and send corresponding commands to the device itself to perform action. In following sections I will explain how extracting gestures from a 2D images is implemented.

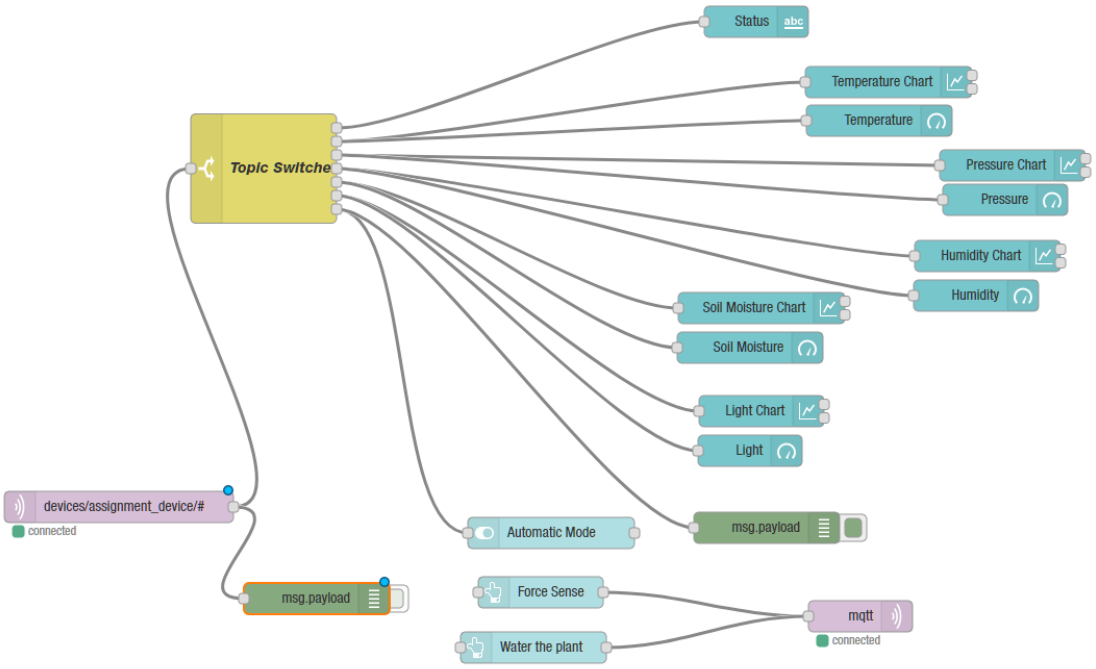


Figure 3: Node red flowchart

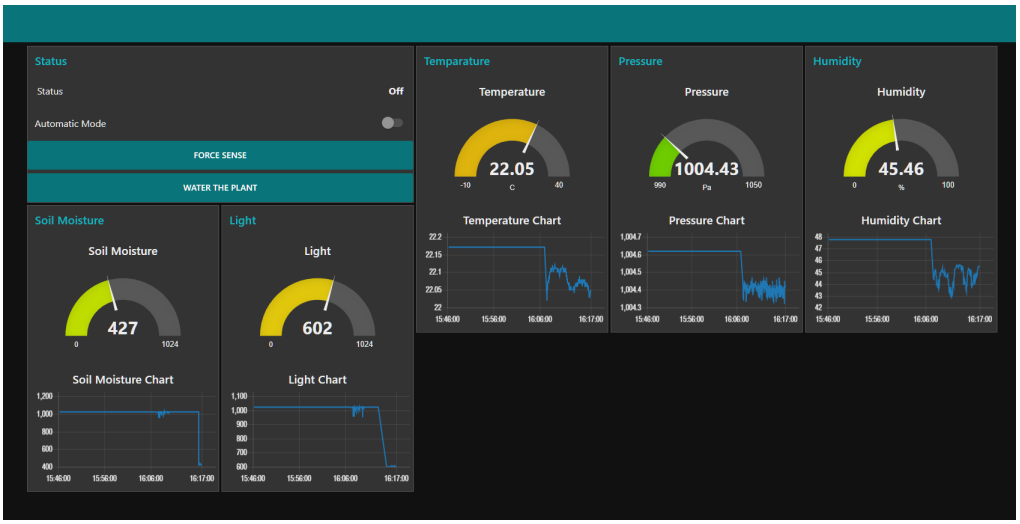
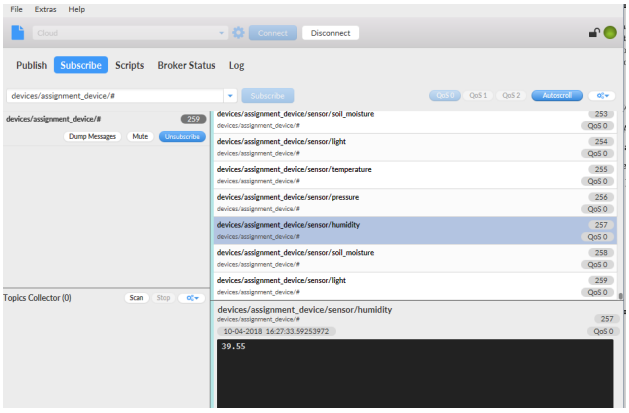
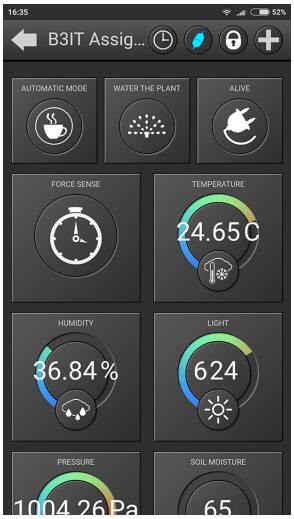


Figure 4: Node red dashboard



(a) MQTT.fx window



(b) MQTT Buddy

3.1 System Architecture

First of all the program is split into two parts: ***"cvision"*** library which should not be confused with *"cv/opencv"* library and the **main** ui.

cvision is created for this assignment but is meant to be reused with other projects. It has implementations of general methods for histogram extraction, image segmentation, limb recognition and object tracking. OpenCV library is used to achieve that.

main consists of a collection of ui components and a defined order in which the different features of cvision are used.

User UI exists of a window and a ordered set of different components named **window helpers**. Window rendering starts by creating an empty image matrix and passing it in order to every window helper which can draw on it or swap it for another image matrix. With this the whole gesture recognition task is broken in a set of small tasks. One helper is responsible for task of segmenting the image while the other is responsible for cleaning the segmented mask. Window helpers are also interconnected and can pass data between each other (For example: segmentation helper passes mask to segmentation cleaner). The whole flow can seen in figure 6.

The final image is then drawn by the window helper. Before that the image is passed between different helpers that either modify it, do nothing or draw some debug information. The convention is not to modify the image if it should not be visible in the final result to avoid building of a unorganized flow (Passing masks as result image to next layer is prohibited. Instead it is accessed by the layer that needs it).

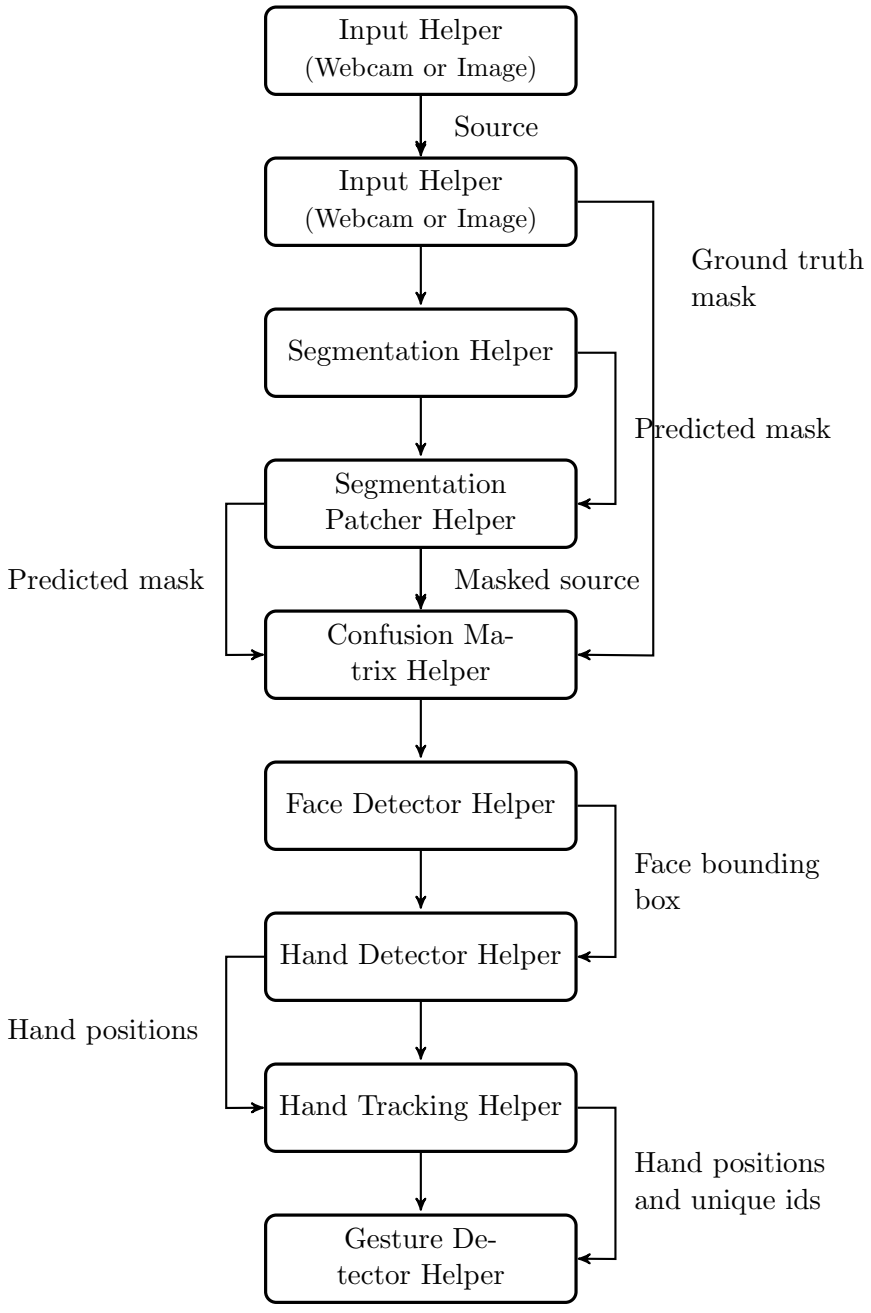


Figure 6: Structure of window helpers for gesture recognition task

Every

3.1.1 Performance

The performance was kept in mind when creating all the layers and segmentation. Therefore calculating the bayes probability per pixel is happening in parallel. *(file: cvision/segmentation.cpp method: complex_segmentation*). For Face detection Viola Jones algorithm is used. This is provided by opencv out of the box and could not be optimized. Overal with face detection the frame render time is 50ms(20 fps) wand without 23ms(43fps) which is enough to hold a steady frame rate.

3.2 Segmentation

Before hand can be extracted from the image the hand itself needs to be found. Since we are working with a 2D image, it will happen by extracting the relevant color from the image which can be classified as skin color.

3.2.1 Probabilistic skin extraction

In order to do that two histograms are made from a dataset of images with pre-masked skin parts. First histogram the occurrences of skin colors while the other is for occurrences of non skin colors (Figure 9). This histogram is takes values 0-255 from RGB colorspace [1] and counts the sightings into the histograms. Finally the confidence of wether something is a skin and non-skin color can be used in to predict the final probability of a pixel is a skin pixel or not with Bayesian theorem (equation 1). If probability is larger than a certain threshold then is seen as a skin pixel. Threshold is adjustable by the user. This statistical approach of segmenting skin from images approach is borrowed from Jones and Rehg [1]. This process can be found in *(file: cvision/segmentation.cpp method: complex_segmentation*) and *(file: cvision/evaluation.cpp method: bayes_probability)*

$$P(\textit{skin} \mid \textit{non-skin}) = \frac{P(\textit{non-skin} \mid \textit{skin})\, P(\textit{skin})}{P(\textit{non-skin})} \tag{1}$$

Extracted data is saved as a separated file which can be loaded anytime so histograms don't need to be calculated every time. File is in *"data/saves/complex_seg.save"*

3.2.2 Morphology

After the generation of the mask it is very noisy. This noise is removed at morphology stage of the segmentation. First erode and then dilate is applied to the mask. A kernel of size 3 px proved to provide the best results. *(file: cvision/segmentation.cpp method: clean_segmentation*)

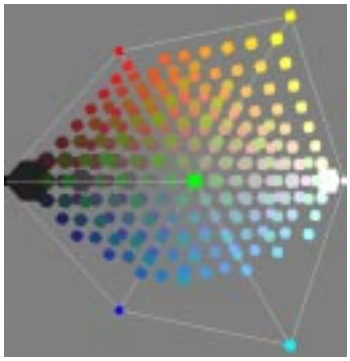


Figure 7: Skin histogram Jones and Rehg [1]

3.2.3 Evaluation

The results of of the masks are passed to the Confusion Matrix helper which if provied a ground truth compares the two masks and calculates the confusion matrix. The result is then logged. By evaluating the whole dataset and averaging the following results were extracted. **Note:** the validation dataset was included in the training dataset, so the results might have a slight bias. *(file: cvision/evaluation.cpp method: evaluation::make_confsion_matrix)*

Statistic	hgr1 dataset	Pratheepan dataset
True positives	40.08%	9.31%
True negatives	57.31%	0.60%
False positives	1.20%	89.57%
False negatives	1.40%	0.50%
Relative Error	0.0162933	0.0111449
Total Preformance	0.0160321	0.011022
Bulls eye %	100%	94.89%
Precision	96.15%	93.93%
Recall	100%	94.89%
F1 score	98.03%	94.41%
Specificity	97.32%	99.33%
Generality	40.00%	9.81%

Two different datasets were used. While hgr1[2][3] dataset has different lightning conditions and a lot of data there is not much diversity in skin color. Pratheepan[4] on contrary has a lot of diversity but very few samples.

3.3 Hand Recognition

Once skin colored pixels are segmented, the contours are created around the left over parts. If area of a contour is larger than the specified threshold then

is seen as a significant candidate to be a hand.*(file: main/detection_helper.h method: HandDetectorHelper::draw*)

Face detection is used to remove corresponding contours from the candidate list and thus thinning the list. Every contour from here on is regarded as hand if it passes minimum requirements like having at least three vertices. False positives can't be efficiently eliminating without creating as large abundance of false negatives since a closed hand has not much characteristics. This is compensated by gesture recognition by initiating with some hard to falsify characteristics like presence of five fingers.

Hand recognition works as follows. *(file: cvision/hand_recognition.cpp method: hand::recognize_hand)*

3.3.1 Hand Palm extraction

The most reliable way to find the palm center is to fine largest incircle of the contour. Circle with palm center as center and radius 3.5 times larger than palm center is regarded as **region of interest**.Yeo et al. [5]

My own touch on this was adding the weight to certain areas of the contour for incircle to be found. Sometimes there is an arm attached to the hand which can produce a larger incircle than the palm itself. Therefore the convex defects in the contour will be seen as points of priority. The weight of those points is tuned to the largest incircle will be found but still nearby these points.*(file: cvision/geometry.hpp method: incircle)*

Once this palm circle is found we can cut the arm off by checking what is outside of the region of interest. If something is it is regarded as an arm and will be cut off from the palm itself and not the area of interest.*(file: cvision/hand_recognition.cpp method: hand::find_enclosing_circle)*

3.4 Finding fingers

Fingers are categorized as defects in the convex hull of the hand contour. Aside from that they are pointy and are angle constrained. These characteristics are used to locate the finger positions in the given contour.Yeo et al. [5]

To be more precise the following checklist used to find the fingers: *(file: cvision/hand_recognition.cpp method: find_fingers)*

1. The depth of the defect is larger than palm center radius but shorter than minimum enclosing circle radius.

2. The angle of the defect is smaller than 100°

Figure 8: Where are your fingers? Vsauce

3. The curvature of the tip is less than 60°

The following approach is very effective for finding two fingers or more. But unfortunately it doesn't work for when only one finger is visible. Therefore there is also a fallback function if no fingers are found. *(file: cvision/hand_recognition.cpp method: find_fingers_fallback)*

1. The depth of the defect is larger than defined threshold.

2. The curvature of the tip is less than 80

Finally when fingers are found the fingertip is extracting by finding the smallest curvature point. Form there the x amount of points to the left and right in the contour are taken, summed up and divided by two which finds a point in the centre of the finger. Now since we have two points the direction can be estimated. Points on the same finger should point to the same peak. If they don't they are still near each other and are removed if the distance between them is smaller than the 0.75 * finger width. *(file: cvision/hand_recognition.cpp method: remove_duplicate_fingers)*

All the fingers and hand information are then stored in a struct which can be further analyzed on.

3.5 Gesture Recognition

3.5.1 Hand Tracking

In order to track the hand gestures we first need to be able to identify the instanced of the same hand in multiple different images. For this the so called nearest neighbor matching is used. The hand positions are saved inside the history every frame. On the new frame the hands are again found. To find which instance this hand belongs to all centers the hands in

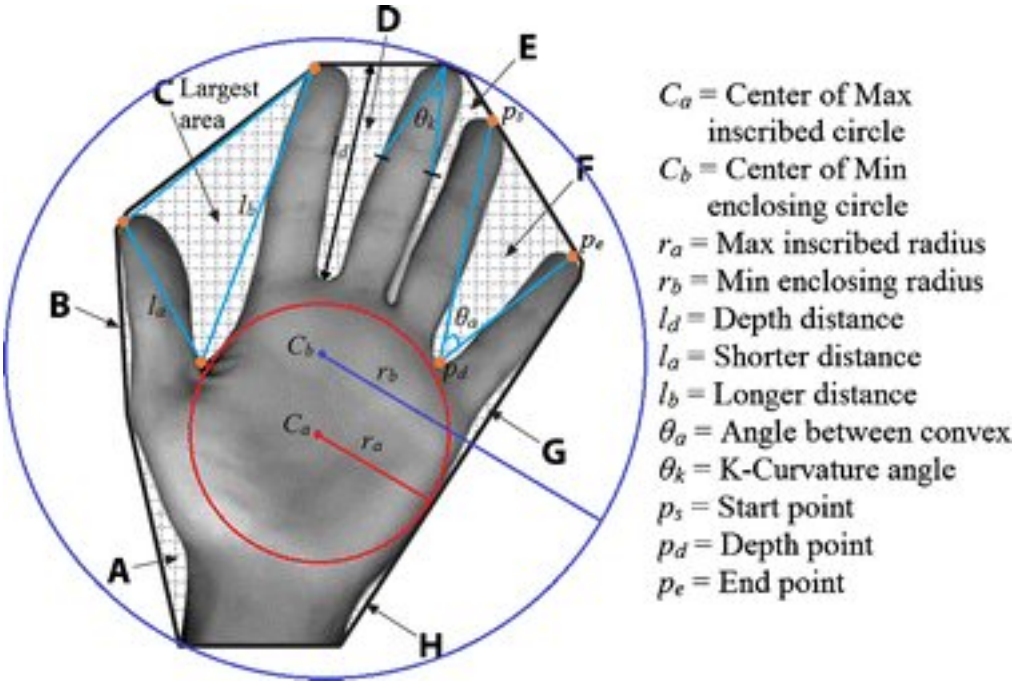


Figure 9: Convex hull, convexity defects of hand shape, extracted from [5] Learning OpenCV

the frame are taken en the nearest neighbor to it is found using the following algorithm:*(file: cvision/object_tracking.cpp method: match_points)*

1. All new points are tupled with the most recent points in history, distance is calculated and the tuple stored in an array.
2. Array is sorted on distance
3. The first(with shortest distance) tuple in the array is taken and the two corresponding instances are a match (they belong to the same object).
4. All the in tuples containing any of the two points are removed from the array. And step two is repeated untill array is empty.

After that the objects that have no history are added and ones that don't exists anymore have their history removed. To add bit more leniency in this step a time out is used. If object is not present for x steps anymore then is removed. Same goes for new objects, only if their history ¿x entries long they are considered relevant.*(file: cvision/object_tracking.h method: update)*

3.5.2 Identifying gestures

To identify gestures I have opted in for the Dollar since it's already made and works "oke'ish". The gesture consist of three different states. *(file: main/gesture_recognition_helper.cpp method: GestureDetectSession::update)*

Preparation User states the intent to that the gesture by opening their hand and showing five fingers. (Hopefully that is not an issue ;))

Stroke User draws a shape which is later passed to the dollar recognizer. For this user should only use one finger.

Termination User shows again five fingers to confirm that the gesture is complete. If at any given moment user closes their hand so no fingers are visible, then the gesture is canceled. More on this in user evaluation.

The recorded location of the finger that is drawing the motion is saved and stabilized using the Kalman filter.

4 User Evaluation

To evaluate usability of the system a controlled test is performed. This is to evaluate how easy it is to use the gesture recognition and test a hypothesis:

Terminating a gesture with an open palm and canceling with closed is more intuitive for user than vice versa.

4.1 Methods

4.1.1 Participants and Environment

Participants used in this test are all within my friends and family circle. I have managed to get five people to participate in my despicable test.

The environment used is the living room and the bedroom. Both rooms are used at different times and therefore vary in background and lighting which are the independent variables which vary and should not affect the results of the test.

For the test itself my laptop is used which has a webcam and more than enough power to run the gesture recognition program. Also it runs Arch linux ;)

4.1.2 Procedure and design

Before the test all the participants are explained the working of the gesture recognition software and the two cases that will be tested.

Case 1 User draws three different gestures: a circle, a square and a triangle. To do that user activates the gesture by opening their and and terminates by closing it. To cancel it user can open the hand again which will activate the new motion.

Case 2 User draws three different gestures: a circle, a square and a triangle. This time user closes hand to cancel the motion and opens it to terminate/confirm the motion.

At after each case user gets a survey with three different questions there user can answer using Likert scales. For this five options are given: strongly disagree, disagree, neutral, agree and strongly agree. After this the answers are summed up and compared.The questions that are asked.

1. Drawing gestures works seamless.
2. I find myself trying to remember how to use the system.
3. Gesture drawing process feels natural.

Aside from the survey task completion time for every gesture is measured and the amount of times user fails to draw the gesture and hast to restart.

4.2 Results

Firstly the survey results are analyzed. The summarization of the survey results is as following. If user strongly disagrees it is counted as -2 and if strongly agrees it is counted as 2. Of questions that ate negative the score is multiplied by -1.

Question	Summary Case 1	Summary Case 2
1 (positive)	5	6
2 (negative)	-2	-5
3 (positive)	-3	7
total (corrected)	4	18

Because the amount of test subjects was small and thus was the population and sample size we cannot do perform Significance testing using z-Score. For this I use the t-Score. Also it is possible to calculate the confidence interval. Confidence level of 95% is used Results can be found below:

Case	Task	Mean time	Standard Deviation	Confidence Interval
1	Draw Circle	12.3	1.9	12.3 ± 1.7
1	Draw Triangle	14.1	2.7	14.1 ± 2.4
1	Draw Square	16.6	3.2	16.6 ± 2.8
2	Draw Circle	11.7	1.5	11.7 ± 1.3
2	Draw Triangle	12.8	2.1	12.8 ± 1.8
2	Draw Square	14.2	2.2	14.2 ± 1.9

4.3 Discussion

Case 2 seems to be the better of the two judging from the results of the survey. SInce it has a higher score of 18 compared to case 1 that has score of 4. Aside from that Completion times for the gesture drawing tasks seem to be lower for case 2 which suggests that that gesture recognition flow is more intuitive. The results of question three seem to confirm that. Its seems that it is more natural to confirm with hand open and cancel a gesture with hand closed.

With this the hypothesis seems to have been confirmed and will be used in the final version of gesture recognition system.

5 Conclusions

Overall the device as end result has turned out better than expected. Its very extensible, has beautiful UI and works as expected. Some more intelligence

like more advanced prediction when the plant needs to be watered would be better than just checking if soil is dry or not. Some cover for the device to prevent shorts if water has been spilled would also be a great idea.

Computer vision also turned out great and I have learned a lot making the system which I am very proud of. The extensibility because of its modular design is insane and worth saving. Aside from that i now have a library with a lot of handy implementations of different algorithms which can be reused in different projects. The skin segmentation works great on the datasets themselves but unfortunately these datasets still have very few skin data. Al through smaller bin sizes in histograms can be used they seem to work degrading for the end results. More data is needed since in practice I have come a lot of (skin) colors which were not in both skin and non-skin histograms and therefore were giving nan when calculating the bayes probability.

Gesture recognition worked out also fine, but dollar recognizer seems to work not very good in practice. A lot of times a cross or a triangle is detected when user is drawing a square. This can be solved by writing own better shape recognizer. Finding hands worked out also very good. Aside from a lot of true positives the real hand gets detected correctly a large portion of the time with fingers aligned correctly.

References

[1] Michael J. Jones and James M. Rehg. Statistical color models with application to skin detection. *International Journal of Computer Vision*, 46(1):81–96, Jan 2002. ISSN 1573-1405. doi: 10.1023/A:1013200319198. URL `https://doi.org/10.1023/A:1013200319198`.

[2] Michal Kawulok, Jolanta Kawulok, Jakub Nalepa, and Bogdan Smolka. Self-adaptive algorithm for segmenting skin regions. *EURASIP Journal on Advances in Signal Processing*, 2014(170):1–22, 2014. ISSN 1687-6180. doi: 10.1186/1687-6180-2014-170. URL `http://asp.eurasipjournals.com/content/2014/1/170`.

[3] Tomasz Grzejszczak, Michal Kawulok, and Adam Galuszka. Hand landmarks detection and localization in color images. *Multimedia Tools and Applications*, 75(23):16363–16387, 2016. ISSN 1573-7721. doi: 10.1007/s11042-015-2934-5.

[4] Y. Pratheepan W.R. Tan, C.S. Chan and J. Condell. A fusion approach for efficient human skin detection. *IEEE Transactions on Industrial Informatics, vol.8(1):138-147 (T-II 2012)*, 2012.

[5] Hui-Shyong Yeo, Byung-Gook Lee, and Hyotaek Lim. Hand tracking and gesture recognition system for human-computer interaction using low-cost hardware. *Multimedia Tools and Applications*, 74(8):2687–2715, Apr 2015. ISSN 1573-7721. doi: 10.1007/s11042-013-1501-1. URL `https://doi.org/10.1007/s11042-013-1501-1`.