University of Twente

Twente House Cup Point Counter

Authors:

Egbert Dijkstra
s1700618
e.dijkstra@student.utwente.nl

Alain Jansen
s1694839
g.a.a.jansen@student.utwente.nl

Bob Oldengarm
s1859870
b.m.j.oldengarm@student.utwente.nl

Jesse Rengers
s1838989
j.j.rengers@student.utwente.nl

Supervisors:

Yeray Barrious Fleitas Msc
y.d.c.barriosfleitas@utwente.nl

Jeroen Klein Brinke Msc
j.kleinbrinke@utwente.nl

Department of EEMCS
Enschede
The Netherlands
November 6, 2020

UNIVERSITY OF TWENTE.

ABSTRACT

The number of students following the TCS program has been rapidly increasing over the past few years. A house system analogous to the system of Hogwarts has been introduced to give the students a sense of belonging and make everything more manageable. Of course, an integral part of the Hogwarts house system is the annual House Cup where the houses gather points over the course of a year. That is why our client asked us to design and implement a system to facilitate the TCS House Cup.

One part of the system is the online management tool. It is integrated with Canvas and is used to view and update the scores of the houses. In the Harry Potter books, the score is also represented by giant hourglasses filled with gems, which meant our system should also contain a physical representation of the scores. This has been implemented as tanks filled with water, where the amount of water represents the score of the house.

In this report, the system concept as well as their requirements will be laid out. Furthermore,the design choices that were made and how they were implemented will be discussed. Finally, an evaluation will be given of both the system and our process during implementation.

CONTENTS

1 INTRODUCTION

With the decision to make the Technical Computer Science an international study, the student numbers sky rocketed. The bachelor went from 60 to 70 students to the 350 we have today. This obviously changed a lot of processes, courses needed to be revamped and language courses were being given so that everyone had a good experience. One big hurdle to take was 'the University of Twente' experience. In dutch we have a saying: "ons kent ons", everybody knows each other. Students to teacher, but also teacher to students. In order to keep this kind of idea something needed to change, because it is not possible with 350 people.

A new system was invented, or well, was lend from Harry Potter. The houses system was build. It was introduced in the 2019-2020 college year and was a great success. It consists 4 houses with 4 distinct colors: red, blue, green and yellow. But it needed something more, some form of interaction or gamification. For this inspiration was drawn from, again, Harry Potter. In the Harry Potter Books, an annual House Cup is organised. We set out to implement this in a proof of concept.

2 CONCEPT

The system consists of two main parts: an online management tool and a physical representation of the scores.

The management system is a Canvas plugin. In this way, teachers, housemasters and students can easily access it without downloading other software or going to another website. The management system shows the current scores of the houses while teachers and housemasters are also able to increase or decrease the scores. A logbook should also be included on the page to keep track of all mutations.

In the Harry Potter books, the scores are represented by giant hourglasses filled with gems. This is of course a bit difficult to implement, but a physical representation should of course be included in a competition that draws inspiration from Harry Potter. That is why our client would like for the scores to be represented by cylinders filled with colored water, where the color of the water represents the color of the house the water tank belongs to.

Our task was to create a fully functional proof-of-concept prototype for a tank for one of the houses. The management system and the physical representation of the scores are almost two separate systems. They are only linked by the fact that the physical representation needs to retrieve the scores from the management system. That is why the project was divided in two parts: the software part for the management system and the hardware part for the physical representation of the scores.

3 REQUIREMENTS

As explained in the previous section, the project was divided in a software and a hardware section. As such, the requirements for each part will be discussed separately.

3.1 Software requirements

The requirements for the software part have been analysed by looking at the use cases for teachers and students, since those are the main groups of people that will use the system.

1. As a student or teacher, I want to be able to access the management system.

3.1.1 Requirements

(a) The management system should be accessible through a Canvas Plugin.
(b) Students and teachers should both be authorized to access the Canvas plugin.

2. As a student or teacher, I want to be able to see the scores online.

3.1.2 Requirements

(a) The current scores should be clearly stated per house.
(b) The current scores should be shown graphically, replicating the way in which the scores are shown by the physical representation.

3. Only teachers should be able to mutate the scores online.

3.1.3 Requirements

(a) Teachers should be authorized to mutate the scores.
(b) Mutating the scores should be intuitive.
(c) Students should not be authorized to mutate the scores.

4. As a student or teacher, I want to be able to see all mutations.

3.1.4 Requirements

(a) The plugin should include a logbook where all mutations are listed.
(b) Rows in the logbook with negative mutations should be colored red.
(c) The logbook should have columns for date, house, mutation and teacher.

3.2 Hardware requirements

It was hard to capture the required functionality of the hardware component by using use cases. Therefore, hardware requirements are represented in MoSCoW format.

1. **MUST HAVE:**
   (a) The water level in the tank must represent the score of a house.
   (b) The system must be able to raise the water level.
   (c) The system must be able to lower the water level.
   (d) The system must be controlled by a Raspberry Pi or an Arduino.
   (e) The system must be able to retrieve the score of a house from the software component.
   (f) If the water level changes while it should not, the system must be able to get the water level back to the correct level.
2. **SHOULD HAVE:**
   (a) The water level in the tank should be precise within 2 mm of its intended height.
   (b) The sensors that measure the water level should not be visible.
   (c) The water should be colored in such a way that it represent the color of its designated house.
   (d) The sensors used for the prototype should cost no more than 10 euros.
3. **COULD HAVE:**
   (a) The tank that displays the score could look like a finished product.
   (b) All hardware that is not the tank that displays the score could be hidden from sight.
   (c) The system could be able to detect leakage.
   (d) If the score went from the minimum to the maximum amount, the tank that displays the score could be filled within five minutes.
4. **WILL NOT HAVE:**
   (a) The system will not be designed in such a way that it is optimal for four water tanks.
   (b) The exterior of the (prototype of the) system will not be made of the materials that would be used for a real version.

4 SOFTWARE DESIGN

4.1 Global view

A global overview of the structure of the software can be seen in figure 4.1. We chose for a service-oriented architecture, separating the plugin and the data server to keep our product modular. In this way, we can make changes to the plugin without affecting the data server and vice versa. The plugin is opened inside a Canvas course. Opening the plugin will start the LTI flow that authenticates the user. More details about the LTI protocol will follow in a next section. After the LTI flow, the plugin knows who the user is and whether the user is a teacher or not. The plugin uses this information to render the correct application. Teachers get the option to mutate the scores while normal users don't get this option.

4.2 Plugin

The plugin is created using Express.js, a backend web application framework for Node.js. Express.js is used because it is a popular and widely used Javascpript framework. A lot of documentation and tutorials are available for this framework, which is kind of the standard choice as server framework for Node.js. The frontend of the plugin was made using the Handlebars templating framework, which is easy to use and intuitive.

The plugin shows two main things, four bars that represent the score of each house and a logbook. The height of the bars show how much points a house has relative to the other houses, just like the physical presentation does. The bars are colored just like the houses. The logbook lists all mutations in a table with columns for mutation score, issuer, house and date.

Through the use of web sockets these items can be updated in real-time.

4.3 LTI Protocol

The Learning Tools Interoperability (LTI) protocol is a protocol that serves to standardize the communication between learning systems such as Canvas and Blackboard and external tools like our application. The current version of LTI, version 1.3, uses Oauth2 for authorization, OpenID Connect for authentication and JSON Web Token (JWT) as standard to encrypt access tokens.

When the plugin is started, the LTI flow sends data back and forth between the Canvas LTI service and our application. After that, our application has contextual data about the user. This data includes the users name and it's role in the course that the plugin is opened in. We can then use this role for authorization.

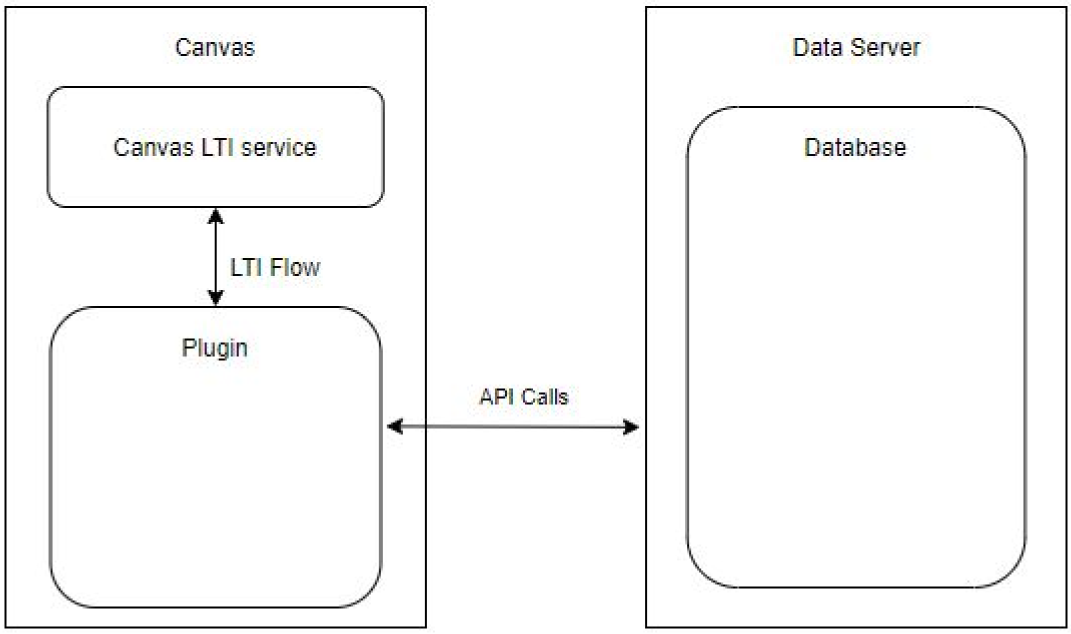


Figure 4.1: Overview of the software parts and their communication.

4.4 Data Server

Next to the plugin, we have a data server with an API that the plugin can make requests to. This data server is also created with Express.js and stores its data in a MongoDB database.

4.5 Privacy

The application stores two things in its database: the amount of points that a house has and the mutations that are done. The amount of points of a house are not seen as privacy issues since these are not related to a person (data subject) but rather to a house.
All mutations are labelled by the name of an issuer. This means that they are related to a person and can be seen as personal data. To safeguard the privacy of all issuers, we have looked at the GDPR and believe that we adhere to it when the issuer of a mutation has given consent.

5 HARDWARE DESIGN

The hardware component is used to display the score by means of (colored) water that fills a water tank. The amount of water in the tank represents the score. This component has three main tasks: retrieving the point total of a house, monitoring the water level and adjusting the water level to represent the point total. The hardware can be controlled by either an Arduino, Raspberry Pi or a similar device. Several design choices had to be made. These will be detailed in this chapter.

5.1 Main controller and retrieving the scores

One of the requirements was that the system is controlled by either a Raspberry Pi or an Arduino. While the Raspberry Pi (RPi) is basically a tiny fully fledged computer, the Arduino is only a micro-controller. This means the Raspberry Pi has a lot more processing power. However, the tasks of the system are only performed periodically and thus the processing power does not matter all that much. To be able to perform all three tasks discussed in the introduction of these parts, the board will need both a GPIO interface and internet connectivity. Both the Arduino Uno (or mega) and the Raspberry Pi model 3 have a GPIO interface. The Raspberry Pi has a built in WiFi module, while the Arduino requires a WiFi shield to be able to connect to the internet. Finally, the Arduino is cheaper than the Raspberry Pi model 3B by about 20 euros.

Because of the built-in WiFi connectivity and the option to stay flexible in system architecture, the Raspberry Pi model 3B has been chosen as the controller of the hardware component. The internet connectivity also makes retrieving the scores very easy. All it needs to do is make an API call to the back-end of the software component. Then the scores can be manipulated in such a way that they can be used as a reference value to fill the tank.

5.2 Monitoring the water level

5.2.1 Potential sensing technologies

Different sensing options were considered for monitoring the water level. The most interesting options will be discussed in this chapter. There were other alternatives, but these were either too expensive, needlessly complicated or not suitable for application in this system. The sensors have been evaluated on the following metrics:

- Cost: What is the price of the sensor (approximately)?
- Mounting place: Where will the sensor be placed?
- Visibility: Is the sensor visible in the display tank?
- Ease of installation: How difficult is it to place the sensor? Does it require specific changes to the tank?

- Potential problems: What are some concerns when using the sensor?

Ultrasound Sensor
This sensor sends an ultrasound wave which will bounce off the water surface. It then measures the time difference between sending the wave and receiving it back. The distance between the sensor and the water surface can be calculated using this time difference.

- Example sensor: **HC-SR04**
- Cost: 5 euros
- Mounting place: In the lid of the tank.
- Visibility: Not visible.
- Ease of installation: Easy. A hole must be created in the lid to place either the sensor of the wires in.
- Potential problems: The sensor will likely have accuracy problems. The ultrasound signal it sends will diverge, and unless a very large diameter is used for the tank, it will bounce off the walls of the tank and thus affect the resulting measured distance.

Time-of-Flight (ToF) laser sensor
Works on the same basic principle as the ultrasound sensor. However, infrared light will not bounce off the water. To solve this, a floating plastic disc can be placed on top of the water.

- Example: **VL53L1X**
- Cost: 10-20 euros
- Mounting place: In the lid of the tank.
- Visibility: The sensor itself is not visible. However, a floating object must be placed on the water for the light to reflect on.
- Ease of installation: Easy. A hole must be created in the lid for the wiring.
- Potential problems: The same problems as with the ultrasound sensor apply here. However, the expectation is that this will occur to a lesser extent, since the signal the sensor sends has a lower degree of divergence.

Load cell
A load cell can be used to measure weight. It would be placed directly underneath the water tank. By using the known weight of the water tank and the total weight measured, the amount of water in the tank can be calculated.

- Example: **TAL220** (10kg load cell)
- Cost: 8 euros
- Mounting place: In a construction under the display tank.
- Visibility: Not visible.
- Ease of installation: Difficult. To use it, a custom weighing scale has to be created so the weight of the display tank is nicely distributed.
- Potential problems: The accuracy of the sensor must be tested. Furthermore, the sensor is difficult to use.

Gauge pressure sensor
Measures the difference in air pressure. Some air will be trapped in a tube located in the display tank, which the water can also enter. One end of the sensor is connected to this pipe, to other end to the air outside of the tank. When the water level rises, the trapped air will compress which increases the pressure in the tube. The sensor can use the difference between this pressure and the atmospheric pressure to calculate the water level.

- Example: **MPX2010GP**
- Cost: 10-20 euros
- Mounting place: Near the lid of the display tank.
- Visibility: Visible, because of the extra tube in the display tank.
- Ease of installation: Difficult. An extra tube must be installed in the display tank and must be completely (vertically) straight, otherwise the calculations will be off. Furthermore, the sensor needs to be connected to the air outside of the tank. This means there will be a hole somewhere that could be misused if someone decides to tamper with the system.
- Potential problems: Difficult to use, unknown accuracy.

Miniature Barometer
Measures the absolute pressure while submerged in water. The amount of water in the tank can be derived from this.

- Example: **MS5540C**
- Cost: 15-30 euros
- Mounting place: On the bottom of the tank.
- Visibility: If taken into account for the design, not visible.
- Ease of installation: A little difficult. The sensor is not wireless, and because of that extra holes have to be created in the display for the wires to pass through.
- Potential problems: Extra holes have to be created in the display tank and unknown accuracy.

5.2.2 Testing sensor candidates

For the first sensor tests, the sensors with the easiest working principles have been tested, namely the ultrasound, ToF laser en Load Cell sensors. For each of these, we have used the sensor provided in the examples. After making appropriate test set-ups according to the earlier mentioned mounting locations. For the ToF and ultrasound sensors, a vase of approximately 50 cm in height with a diameter of 12cm has been used. For the load cell, a smaller test set-up with weights under 1kg was used because it was less time consuming than creating a big weighing scale. The sensor have been evaluated based on the following metrics:

- Accuracy: How close was the measured value to the actual value?
- Precision: How close were individual measurements together (for the same water level)?
- Ease of use: How easy was it to use the sensor?

Ultrasound sensor

- Accuracy: The sensor reported distances of approximately 1.5 to 3 cm too short. The further away the water was from the sensor, the less accurate the measurements got.
- Precision: The sensor was very precise. Measurements were within 2mm from one another.
- Ease of use: Very easy.

ToF sensor

- Accuracy: Reported distances were approximately 0.5 to 3 cm too short. The closer the water was to the sensor, the more accurate the results were.
- Precision: The sensor was less precise than the ultrasound sensor. Measurements were within 4 mm from one another.
- Ease of use: Slightly more difficult to use than the ultrasound sensor because it requires the use of a floating object.

Load cell

- Accuracy: All the weights used were less than 1kg, while the final version will likely be around 10 KG when fully filled. The measurements fell within a margin of 10 g.
- Precision: Less precise than the other two sensors, with measurements sometimes reporting differences of 20 g.
- Ease of use: Making the little test-set up (basically a small weighing scale) was time consuming. This was because it had to be very stable, otherwise the weight of the object on it would not be properly distributed. In the end, it was not completely stable, hence the low precision of the sensor. If it were to be used for the big water tank, the whole "physical" design of the display tank would revolve around being used with this sensor. In short, this sensor is more difficult to use for this project than the other two sensors.

5.2.3 Final sensor choice

The ultrasound and ToF sensors were immediately chosen over the load cell because they are simpler to use and install. Between the two, the ToF sensor was slightly more accurate than the ultrasound sensor, but both sensors were not sufficiently accurate. However, the ultrasound sensor reported very precise measurements. This means it would be easy to calibrate the sensor by using a discrete point scale. This basically meant that between 50 and 100 "point levels" could be displayed and the ultrasound sensor would be calibrated for each point level. This is way more accurate than trying to have a continuous point scale when using cheap (and thus a bit inaccurate) sensors.

Figure 5.1 shows a schematic overview of the display tank where the ultrasound sensor is installed of the lid of the tank. The sensor is connected to the Raspberry Pi to report the distance to the water surface.

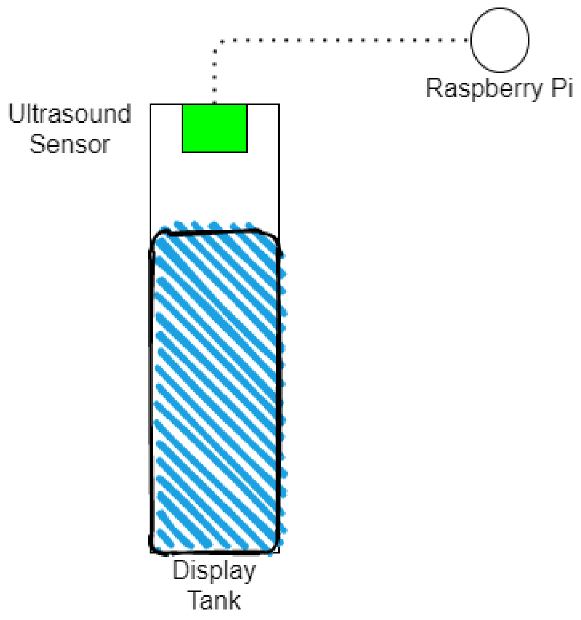


Figure 5.1: Schematic overview of the display tank.

5.3 Changing the water level

The water level will be changed by means of a pump. The 12v peristaltic pump of Adafruit has been chosen. The main advantages of this pump are that it is cheap, easy to use and the pumping direction can be changed by inverting the current direction (so it can both fill and drain). The main disadvantage is that the pumping itself is very slow (around 100 ml/minute according to the documentation). This was not deemed to be a big problem however. The system could either update periodically when no-one is present or the pumps could simply be replaced by better pumps if a final version were to be built.

For monitoring the water level, the ultrasound sensor had been chosen. The initial idea was that the sensor would be calibrated for each point level, and the pump will pump water until a moving average of measurements falls within a certain range of values that correspond to that point level. This worked pretty well after testing with an accuracy of around 2 mm which could still be fine tuned to be more accurate.

However, when looking at future proofing the system, the sensor would have to be calibrated for all tanks for all houses. This is very time consuming. Furthermore, because of sensor drift, there would be no telling when each sensor would have to be calibrated again. A sort of auto calibration function would thus be a nice addition to the system to make the system more self-reliant over a longer period of time.

To calibrate the sensor automatically, the amount of water that is added to the system at a time must correspond exactly to one point level. We considered different options for this purpose which will be shortly discussed below. For most of these options, a so-called “calibration tank” will be used. The basic idea behind this is a small tank between the water reservoir and the main tank. This tank will be used to measure exactly one point level of water. So, if the main tank has to be filled, the calibration tank will be filled from the water reservoir first and after that its contents will be transferred to the display tank. If the display tank needs to be emptied into the calibration tank which will then be emptied into the water reservoir.

5.3.1 Calibration sensor candidates

Flow sensor
The first option would be to use a flow sensor, which can measure the amount of water that is added to the display tank. The main problem when using this sensor is that it is not guaranteed to be very accurate, so this is not a preferential option.

Infrared proximity sensor
The basic idea behind this sensor is that infrared proximity sensors will be placed on two different heights of the calibration tank. The amount of water between these two heights corresponds exactly to one point level. A floating object will be placed on the water in the calibration tank. It will be noticed by a sensor when it passes it. The basic idea here is that it will fill until the upper sensor notices it and drain until the lower sensor notices it. By doing this, always exactly one point level will be pumped into or drained from the display tank.

Non-contact liquid level sensor
The idea here is the same as the infrared proximity sensor, except it does not require the use of a floating object. Once again, two sensors are placed on different heights on the outside of the calibration tank. By measuring the capacitance, the sensor will know when the water inside the tank reaches it level. From here, the procedure is the same as with the infrared proximity sensors.

5.3.2 Final calibration and changing water level choice

As discussed earlier, the flow sensor would only be used if the other options proved to be unfeasible. The non-contact liquid level sensor was the preferential option, but it costs around 16 euros a piece. The infrared proximity sensor only cost around 3 euros a piece and was thus a way cheaper option that should have a similar performance.

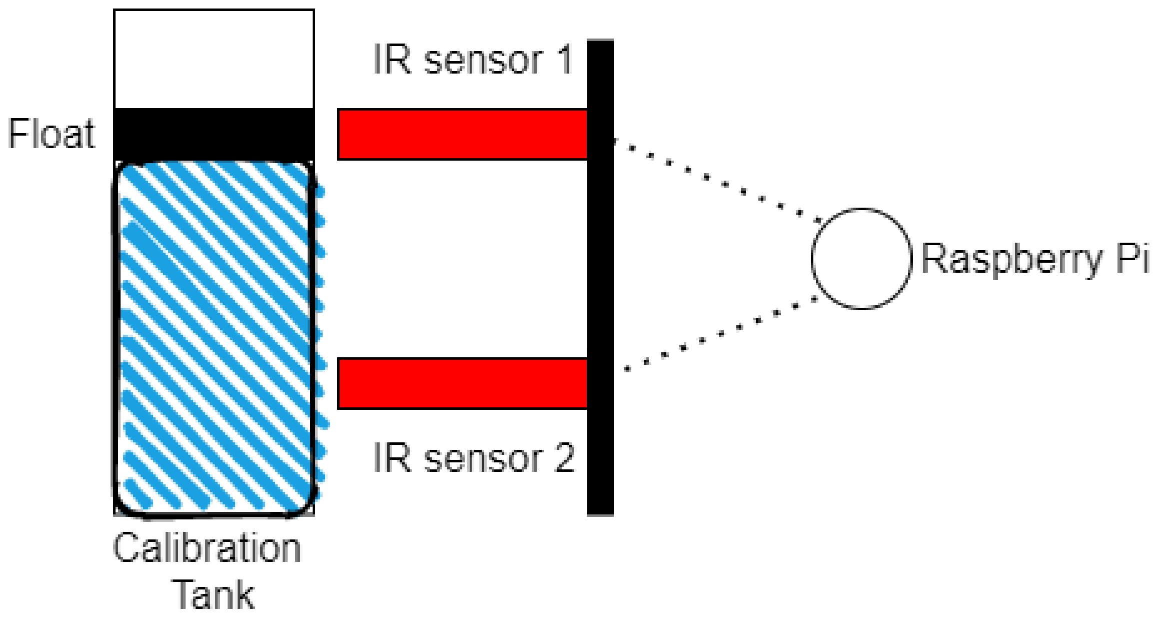


Figure 5.2: Schematic overview of the calibration tank.

Figure 5.2 shows a schematic overview of the calibration tank. The two IR proximity sensors are installed at different heights such that that they contain one point level between them. The tank

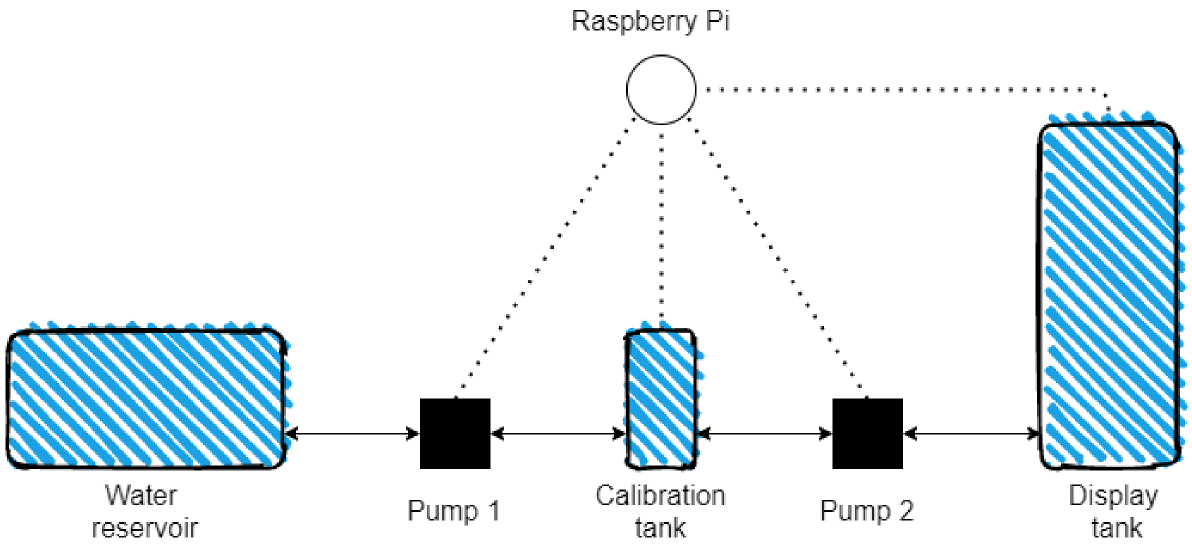


Figure 5.3: Overview of the different main parts of the hardware system.

contains a float which the IR sensors can detect. If it reached the lower IR sensor the system will know the calibration tank is empty. If it reaches the higher IR sensor the system will know the tank now contains exactly one point level's worth of water. This water can either come from the water reservoir for filling the display tank or from the display tank to drain it.

5.4 Global hardware overview

Now all the individual parts of the hardware system are known, it can be discussed how all these pieces come together. In this part, the complete procedure of changing and maintaining the water level will be explained.

Figure 5.3 gives a general overview of the components of the hardware system. The dotted lines indicate that a part is controlled by the Raspberry pi, while the solid lines with the arrowheads indicate the water flow. A quick overview of the system parts:

- Water reservoir: Contains water reserves.
- Pump 1: Pumps water between the reservoir and the calibration tank.
- Calibration tank: Used to measure one point level of water.
- Pump 2: Pumps water between the calibration tank and the display tank.
- Display tank: Used to display the score.

The procedure for filling the display tank is as follows:

1. Pump water from the water reservoir to the calibration tank until it contains one point level.
2. Pump water from the calibration tank into the display tank.
3. Repeat this until the desired amount of point levels have been added.

The procedure for draining the display tank is the same, save that the water flow is inverted (so first from display tank to calibration tank, than from the calibration tank to the water reservoir). After the tank has been filled to the desired amount, the water level has to be maintained. The procedure is as follows:

1. The ultrasound will take 100 measurements. The average of these measurements will be the reference value.
2. Every 20 seconds, 40 measurements will be taken. If the average of these measurements fall within 3 mm of the reference value, nothing happens. If they do not fall within that interval, the system will be reset.

The reset procedure triggers when the water level changes, when it should not change. The display tank will be drained into the calibration tank repeatedly. At some point, the display tank will not contain enough water to completely fill the calibration tank. After 2 minutes, a time-out will occur because the calibration tank has not been filled. The system then knows that the display tank is completely empty. Finally, the system will fill the display tank again until the water reaches the correct level.

6 SOFTWARE IMPLEMENTATION

6.1 Software implementation

Here we will cover the different parts of the software and finally, how these parts work together to get the product that is described in the Concept chapter.

6.1.1 Plugin

Front-end
As mentioned in our design choices, the front-end is created using the templating framework Handlebars. There are four files that create the front-end: *index.js*, *views/home.handlebars*, *views/layouts/main.handlebars* and *public.css*. The latter one wont be discussed since it is just a simple css file that does the styling of the application, together with the Bootstrap library.

views/layout/main.handlebars
is the wrapper around everything in the application. It contains most of the HTML that has to be added for the page to render correctly, such as the head and body elements. *views/layout/main.handlebars* also contains the navigation bar at the top of the screen (called navbar) and the popover that displays when a teacher wants to do a mutation (called a modal). Both the navbar and the modal are native Bootstrap components. Bootstrap, together with JQuery are also added to the application in *views/layout/main.handlebars*

views/home.handlebars
is dynamically rendering HTML code. It gets a houses and a mutations array from *index.js* and renders the corresponding HTML code to display all elements from the houses array and the mutations array.

Back-end
index.js is the back-end file for the plugin. It responds to the LTI flow requests that is started when the user opens the plugin and it requests data from our data server through API requests. Once the data is collected, this can be passed on to *views/home.handlebars* so it can be displayed.

Authentication and Authorization
During the LTI flow, data is being send back and forth between the Canvas LTI service and our application. After this, the user is authenticated. We then have it's name and role in the course in which the plugin is opened. We then make use of the role to authorize the user. Only users who are teachers can mutate scores and normal students are not authorized to do this. We make sure that only teachers can mutate scores by only rendering the code which is needed to mutate scores. This is done by letting *index.js* dynamically render the right code. When the user is a teacher, it sends the code to mutate scores to the client side. If the user is a student then it doesn't render this code. By doing this, a clever student is not able to look into the source code and use hidden elements to still mutate scores.

Type	Address	Parameters	Return	Description
Post	/api/v1/scores	house_name score_to_be_added issuer reason	success status message	adds 'score_to_be_added' to the 'house_name' house
Get	/api/v1/scores	-	array of house objects	gets all information about the houses
Get	/api/v1/scores/mutations	-	array of mutations objects	gets all past mutations

Table 6.1: API calls and their return values

REST API
The Application Programming Interface (API) is the link between the plugin and the data server. Table 6.1 lists all API requests that can be used, together with the expected response. The responses of GET /api/v1/scores and GET api/v1/scores/mutations are score objects and mutation objects respectively. These objects and their properties are set out in table 6.3 and table 6.2 respectively.

At the moment that a user opens the plugin, the GET request for the current scores and all mutations is done so the scores can be shown. When a teacher wants to do a mutation, the POST request will be used to increase or decrease the score of a house.
The Raspberry Pi also makes use from the REST API to get the current scores

Property	Type	Description
_id	string	Identifier given to this mutation
mutation	integer	Amount of points the score is increased or decreased
houseName	string	Name of the house
reason	string	Reason that the issuer gave for this mutation
issuer	string	Teacher or housekeeper that did the mutation
scoreBefore	integer	Score before this mutation
scoreAfter	integer	Score after this mutation
createdAt	string	Date at which this mutation is created
updatedAt	string	Date at which this mutation was updated

Table 6.2: Mutation object and its properties

Property	Type	Description
_id	string	Identifier given to this mutation
houseName	string	Name of the house
score	integer	amount of points at this moment
color	string	color associated with the house
createdAt	string	Date at which this mutation is created
updatedAt	string	Date at which this mutation was updated

Table 6.3: Score object and its properties

7 HARDWARE IMPLEMENTATION

As for the hardware implementation, the used software and hardware components as well as the materials used for the aesthetic of the system will be discussed.

7.1 Software and libraries

7.1.1 Used software

The Raspberry Pi runs the latest version of Raspberry Pi OS (previously called Raspbian). The system is run by a single program, written in python. The RPi.GPIO package is used to control the GPIO pins. This allows for the use of the sensors and actuators of the system.

7.1.2 Code

The code is used to control the pumps and read the sensor data in order to execute the protocols described in the design section.

The score representation that is implemented is the relative score of a houses compared to the total amount of points of all houses. For example, if the blue house has 50 percent of all points, their tank will be filled for 50 point levels.

7.2 Hardware components and set-up

In this chapter, the most important parts of the hardware set-up will be discussed. These are the pumps, the display tank and the calibration tank. Appendix A contains an overview of all the important hardware components that are used as well as a picture that shows how all the wires are connected.

7.2.1 The pumps

The system contains two pumps: one for pumping water between the water reservoir and the calibration tank and one for pumping water between the calibration tank and the display tank. The pumps are not directly connected to the Raspberry pi, instead they are connected to the ls293d motor driver. The motor driver can be used to turn on the pumps. Furthermore, it can also be used to determine the motor direction of a pump (so it can both fill and drain a tank). Figure 7.1 shows a schematic overview of the pumping system. The motor driver itself is powered by the Raspberry Pi. If the Raspberry Pi sends a signal to turn on a pump, the pump will be powered by the 12v battery pack that is also connected to the motor driver.

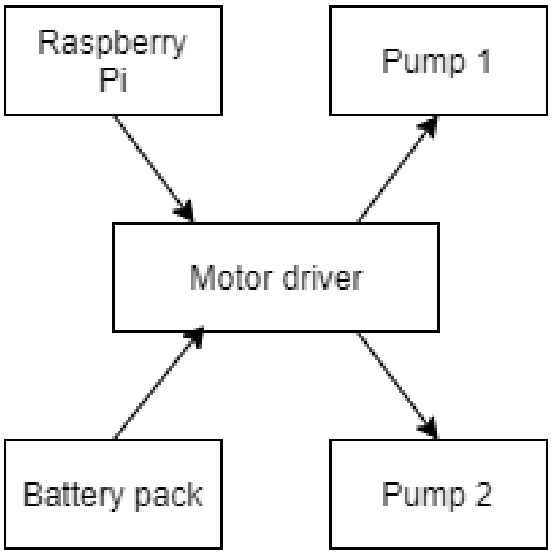


Figure 7.1: Overview of the pumping system.

7.2.2 Display tank

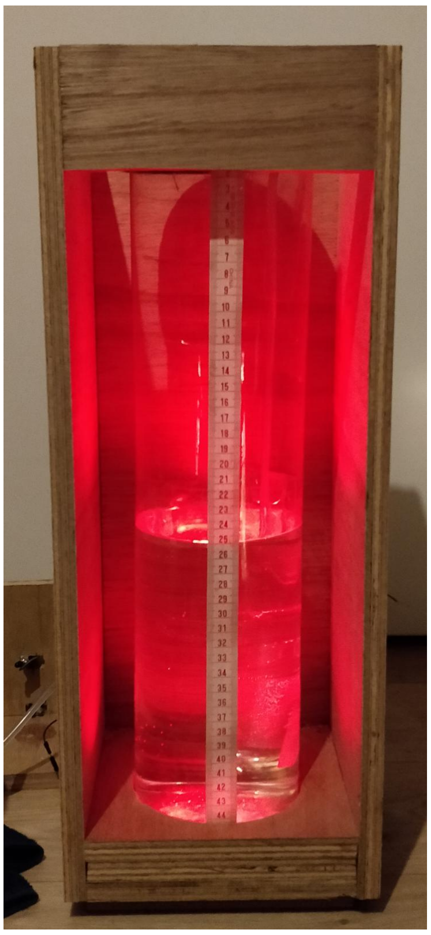


Figure 7.2: Photo of the display tank.

A vase is the water container of the display tank. It is 46 cm in height with a diameter of 12 cm. 40 cm of the tank is used to display the score, which means that each point level has a height

of 4 mm.
A hole in the lid of the vase was created such that the sending and receiving parts of the ultrasound sensor fit through it, but that the back of the sensor where the wires are plugged in still rests on the lid. The ultrasound sensor is directly connected to the Raspberry Pi. Figures 7.3 and 7.5 show the top and bottom of the lid respectively, with the ultrasound sensor mounted in it.

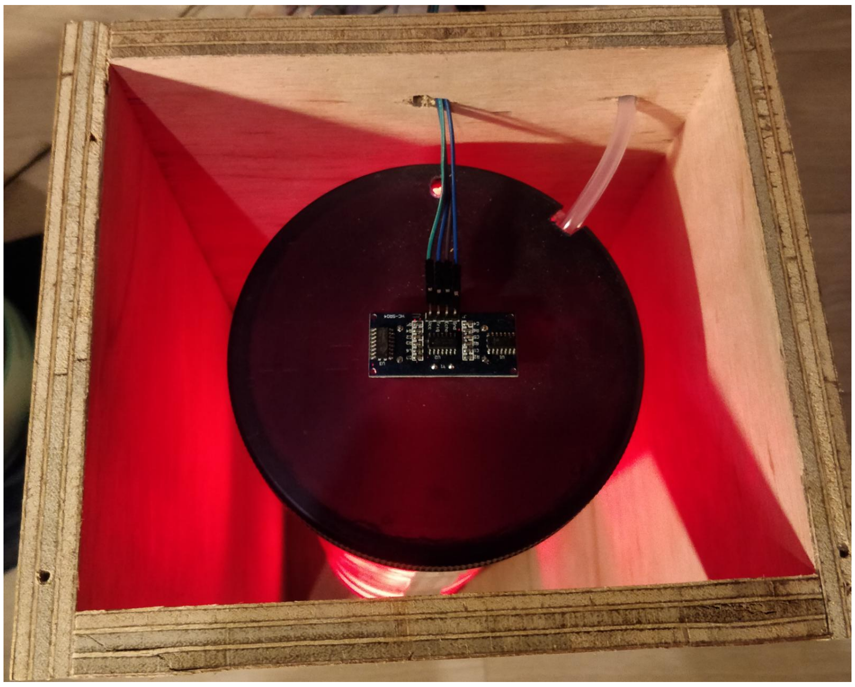


Figure 7.3: Photo of the top of the lid of the display tank with the ultrasound sensor.

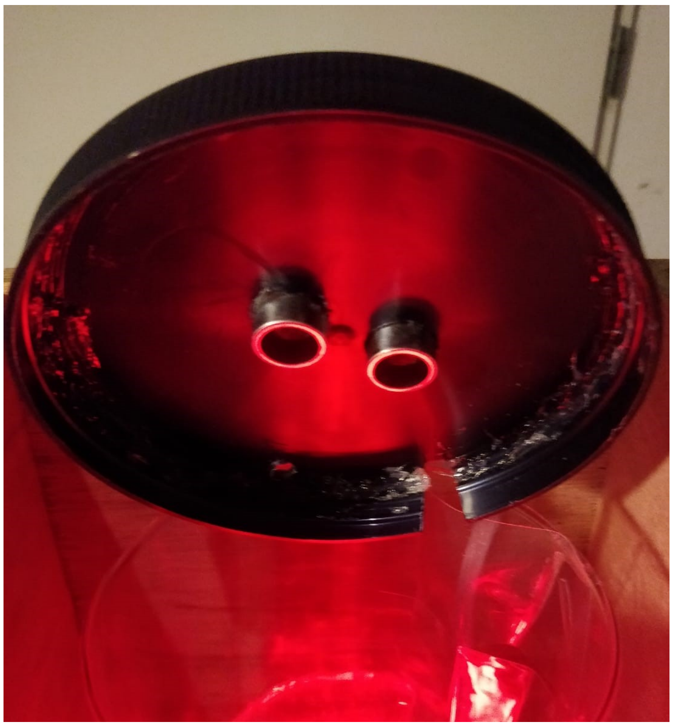


Figure 7.4: Photo of the bottom of the lid of the display tank with the ultrasound sensor.

7.2.3 Calibration tank

The part of the display tank that is reserved to be filled with water has a volume of 4.5 liter. This means that one point level has a volume of 0.045 liter or 45 mL. This means the volume of the calibration tank should be at least 45 mL, but preferably a little bit bigger. That is why the plastic encasing of a salt mill was chosen.
Then, it was calculated how far the sensors should be apart from each other to contain 45 mL between them. Finally, the sensors where mounted in a piece of wood with the aforementioned distance between them. The float of the display tank is a simple plastic bottle cap.

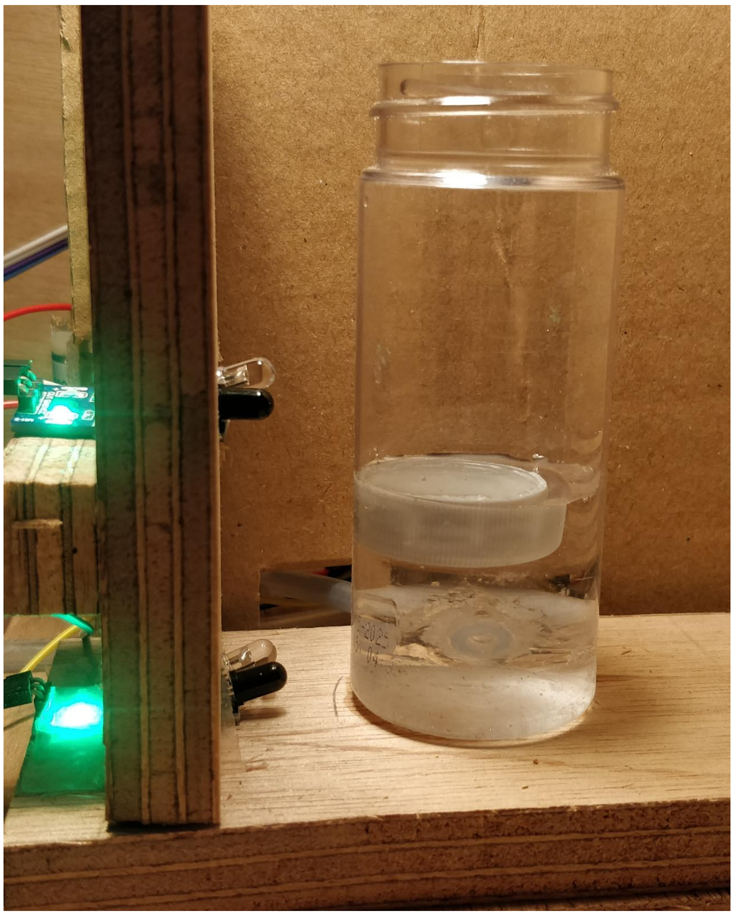


Figure 7.5: Photo of calibration tank with the two IR proximity sensors.

The IR proximity sensors are also directly connected to and powered by the Raspberry Pi. Apart from the wires for power and ground, they each have one wire that reports if something is in the proximity of the sensor. If one of the sensors senses something, it will report a value of one. It reports value of zero otherwise.

7.2.4 Aesthetics

One of the requirements was that the water should be colored. This has been achieved by placing some LED lights beneath the display tank. These lights are not controlled by the Raspberry Pi. Instead, they are controlled by an external remote.

Finally, to give the display tank a finished look, a wooden encasing has been build to put it in.

8 TESTING

8.1 Plugin Test

These tests are tasks that verify a correct working of the plugin from within canvas.

1. **Task**: Access the plugin with a student role.
   Effect: The plugin opens and the 'add mutation' button does not show.

2. **Task**: Access the plugin with a teacher role.
   Effect: The plugin opens and the 'add mutation' button shows.

3. **Task**: As a teacher, add a positive number x to a house that currently has y points.
   Effect:

   - This house now has y + x points.
   - A new mutation shows up in the logbook with x points, the chosen house, the teacher and the current date.

4. **Task**: As a teacher, add a negative number x to a house that currently has y points.
   Effect:

   - This house now has y + x points.
   - A new mutation shows up in the logbook with x points, the chosen house, the teacher and the current date. The color of this row is red.

5. **Task**: As a teacher, add a negative number x to a house that currently has y points such that $x + y < 0$.
   Effect: The teacher is not able to do this and gets an error message.

8.2 API Test

These tests are tasks that verify the correct working of the Rest API.

1. **Task**: Call Get /api/v1/scores.
   Effect: Get an array of houses objects as response.

2. **Task**: Call Get /api/v1/scores/mutations.
   Effect: Get an array of houses objects as response.

3. **Task**: Call Post /api/v1/scores with an existing `house_name` parameter and a `score_to_be_added` which is a positive number.
   Effect: The score of the chosen house gets increased by that number.

4. **Task**: Call Post /api/v1/scores with an existing `house_name` parameter and a `score_to_be_added` which is a negative number and can possible make the score lower than 0.
   Effect: The score gets decreased by that number, possibly below 0.

5. **Task**: Call Post /api/v1/scores with an existing `house_name` parameter and a `score_to_be_added` which is not a number but a string of text.
   Effect: An error is send back letting you known that the score is not a number

6. **Task**: Call Post /api/v1/scores with an non-existing `house_name` parameter and a `score_to_be_added` which is any number, possibly negative.
   Effect: A new house gets created with the chosen house name and the given score, possibly negative. The new house will also be shown in the plugin.

7. **Task**: Call Post /api/v1/score with valid parameters but leave out the reason parameter.
   Effect: The API responds with 'success' and adds the score. The reason field in the logbook is undefined.

8. **Task**: Call Post /api/v1/score with valid parameters but leave out the issuer parameter.
   Effect: The API responds with an error but adds the score. The mutation is not shown in the logbook.

The effects of some of the API request together with different parameters may be unwanted and cause problems. To overcome this, we consider the problems and possible solutions in the Future Work chapter.

8.3 Hardware tests

When the hardware component was finished, multiple tests were conducted to test different facets of the functionality of the system. In the first three tests, the amount of point levels was put into the system by hand. For the final test, the points reported from the API have been used.

8.3.1 Accuracy

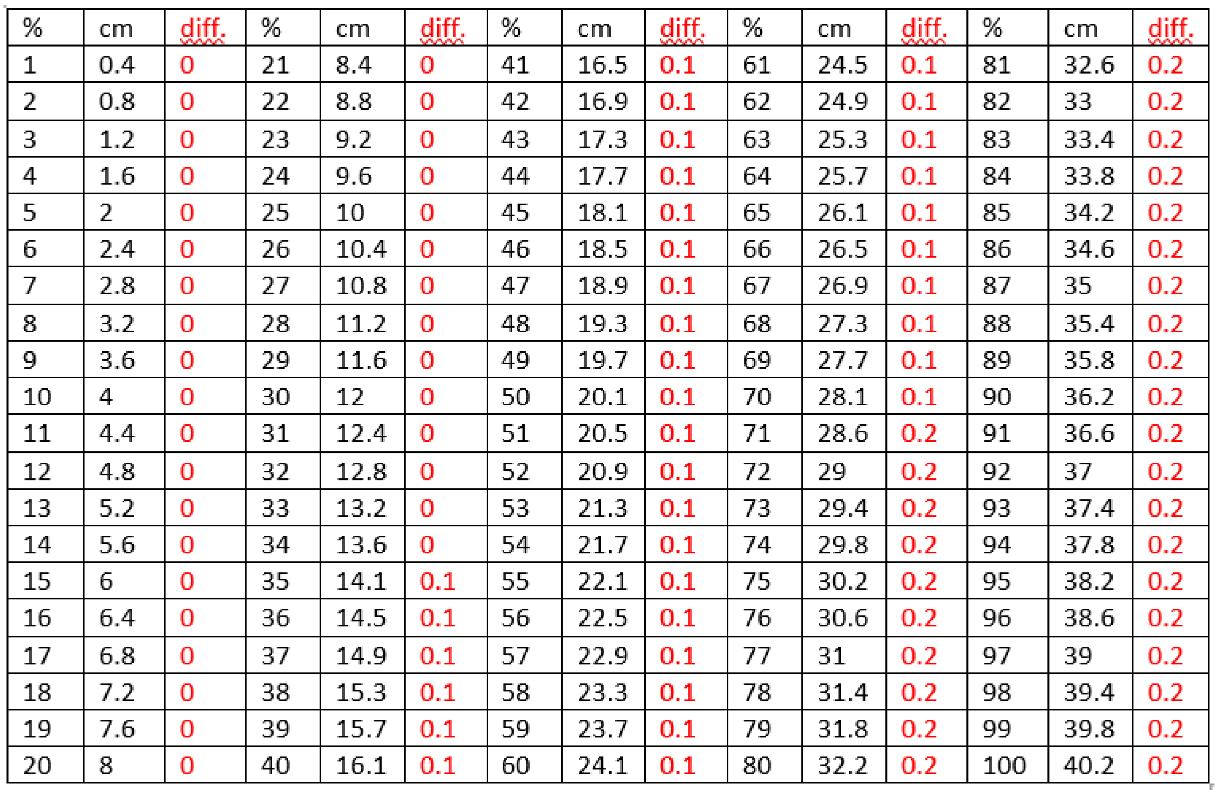
For this test, one point level was added at a time. After adding this point level, the height of the water level read from the measuring tape and the height it should theoretically be at are noted down. The difference between them is how much the pumped amount deviates from its intended level.
Figure 8.1 shows the results of the test. The column percentage contains with how many point levels the tank was filled, the column cm shows how high the water level was in cm and the column diff. shows the difference between the actual water level and what it should have been.

From the results it follows that the deviation was 1mm at around the 35th point level. This difference increased to 2mm at the 70th point level.

8.3.2 Maintaining water level

If the system were to be actually deployed, the water level would have to be maintained over a longer period of time. This test served to see that if the water level does not change, the system also reports no changes. It basically evaluates the stability of the readings of the ultrasound sensor. This has been tested for a low (10 cm), medium (22 cm) and high level (38 cm) of water in the display tank. Ideally, the test should see if the sensor reports no changes over a period of days. Due to time constraints, each water level was tested for approximately twelve hours.

All tests reported zero times that the water level was changed, which means that the ultrasound sensor was indeed very stable.



%	cm	diff.	%	cm	diff.	%	cm	diff.	%	cm	diff.	%	cm	diff.
1	0.4	0	21	8.4	0	41	16.5	0.1	61	24.5	0.1	81	32.6	0.2
2	0.8	0	22	8.8	0	42	16.9	0.1	62	24.9	0.1	82	33	0.2
3	1.2	0	23	9.2	0	43	17.3	0.1	63	25.3	0.1	83	33.4	0.2
4	1.6	0	24	9.6	0	44	17.7	0.1	64	25.7	0.1	84	33.8	0.2
5	2	0	25	10	0	45	18.1	0.1	65	26.1	0.1	85	34.2	0.2
6	2.4	0	26	10.4	0	46	18.5	0.1	66	26.5	0.1	86	34.6	0.2
7	2.8	0	27	10.8	0	47	18.9	0.1	67	26.9	0.1	87	35	0.2
8	3.2	0	28	11.2	0	48	19.3	0.1	68	27.3	0.1	88	35.4	0.2
9	3.6	0	29	11.6	0	49	19.7	0.1	69	27.7	0.1	89	35.8	0.2
10	4	0	30	12	0	50	20.1	0.1	70	28.1	0.1	90	36.2	0.2
11	4.4	0	31	12.4	0	51	20.5	0.1	71	28.6	0.2	91	36.6	0.2
12	4.8	0	32	12.8	0	52	20.9	0.1	72	29	0.2	92	37	0.2
13	5.2	0	33	13.2	0	53	21.3	0.1	73	29.4	0.2	93	37.4	0.2
14	5.6	0	34	13.6	0	54	21.7	0.1	74	29.8	0.2	94	37.8	0.2
15	6	0	35	14.1	0.1	55	22.1	0.1	75	30.2	0.2	95	38.2	0.2
16	6.4	0	36	14.5	0.1	56	22.5	0.1	76	30.6	0.2	96	38.6	0.2
17	6.8	0	37	14.9	0.1	57	22.9	0.1	77	31	0.2	97	39	0.2
18	7.2	0	38	15.3	0.1	58	23.3	0.1	78	31.4	0.2	98	39.4	0.2
19	7.6	0	39	15.7	0.1	59	23.7	0.1	79	31.8	0.2	99	39.8	0.2
20	8	0	40	16.1	0.1	60	24.1	0.1	80	32.2	0.2	100	40.2	0.2

Figure 8.1: Results of the accuracy test

8.3.3 Reset function

While the previous test tried to see if the system reports no changes if there are no changes, this test evaluates if the system notices a change in water level and fills the tank to the correct level again. This has been tested by lifting the lid with the ultrasound sensor by approximately 3 mm.

The tank was filled at the start of the test with five point levels. After lifting the lid with the ultrasound sensor, the display tank was drained and filled up to five point levels again. This was exactly the intended behaviour, which means the reset function worked correctly.

8.3.4 Full system test

This test served to test the connection with the API back-end. The points of the houses were divided in such a way that the blue house had ten percent of the points (ten point levels). After this score was correctly displayed in the display tank, the share of points of the blue house was changed to seven percent (seven point levels).

The test started with the water level at 5 points levels. After an update from the system, the water level was raised to ten point levels. After the score was updated again, the water level lowered to seven point levels. All of this corresponded with the intended behaviour. Therefore, the processing of the points reported by the API functioned as intended.

9 SYSTEM EVALUATION

9.1 Software

9.1.1 Test analysis

The tests were designed to see if we have implemented all requirements. For the Plugin, this is the case. However, for the API, there are some unwanted effects that should be addressed before the system gets deployed. Recommendations to solve these problems are addressed in chapter 10 about future work.

9.1.2 Requirements analysis

We have used the requirements as set out in chapter 3 to design the software. All these requirements have been addressed in the software. This is tested by using the test listed in chapter 8.

9.2 Hardware

9.2.1 Test analysis

All the tests functioned according to their intended behaviour. During the accuracy test, the water level was off by 2 mm in the worst case. While this is not the ideal value of absolutely no deviation from what it should be, it fell within the requirement that it should be accurate at the 2 mm level. Furthermore, this could likely be improved by more careful configuration of the calibration tank.

9.2.2 Requirements analysis

All the must and should requirements have been implemented. The could requirements that have not been implemented are that the system should be able to detect leakage, that the hardware behind the display tank should be hidden from sight and that the tank should be filled within a certain amount of time.

The first two points will be addressed in the future work section. The other requirement was not fulfilled because the pumps we bought were relatively cheap and have a slow pumping speed. It takes approximately 1.5 minutes to add or remove one point level from the display tank. This does not necessarily have to be a problem. If the system would only be updates periodically, say once a day at 04:00, the pumping speed is not a problem. However, if the scores have to be updated in real time, the current pumps would have to be replaced with pumps that support more throughput.

Finally, the will not requirements were indeed not implemented but will be addressed in the future work section.

10 FUTURE WORK

Over the course of this project, we have implemented as much features as we could. However, given that our product is a only a proof of concept, there are some features that we weren't able to create. The following is a list of features that will make the system easier to work with, nicer to look at or are vital to be implemented when the system would be deployed.

10.1 Create more roles

At the time there are only two roles that a user can be in our application: student and teacher. It is possible to adjust these group by creating a custom course where only the right persons are teachers. However, this is not an optimal solution. Therefore, we suggest to add two more roles

10.1.1 Housekeepers

Every house has one housekeeper, as already implemented in the current house points system. It would be nice if these housekeeper can also hand out points to their houses.

10.1.2 Admin

The complete system should be able to be overlooked by an admin. This admin should have additional options, such as creating events, updating or deleting mutations, creating or deleting a house. The admin should also be able to change the way in which the scores are represented, or change the maximal amount of points. Setting the group of teachers and housekeepers is also something that the admin has to be able to do.

If these roles are implemented, the application becomes more dynamical. The admin can change all settings and manage the group of teachers and housekeepers all from within the application, without workarounds.

While there is no standard canvas role for admin or housekeepers, these roles would need to be created ourselves. We should also be keeping track of who is in what role. This means that after the user has authenticates itself, we should check which authorizations it has in our application. This can be done by checking the canvas token that we get after login in to canvas.

10.2 API Security

At the moment we have not implemented any security measures at the Rest API. This means that anyone could make API calls. This causes no problem with two of the possible requests, being GET scores and GET mutations. On the other hand, posting scores is a request that should only be done by authorized people.

Right now the POST scores app can only be done from within our application if the role from the user is teacher within the course from which the plugin is opened. If the role is something other then teacher, the code to handle the POST score request is not rendered at the client side at all. This means that there is no way in which a clever student can still do the request by looking into the source code. However, once someone knows the address and the right parameters for the POST request for such a request, it can still be done with software such as Postman. There is currently no authorization needed to do this.

The solution to fix this would be to require a form of authorization for our API server. The server should then look if the person is indeed a teacher. If so, the request can be handled and otherwise, the server can response with status code 401 (Unauthorized).

10.3 API Request Handling

As can be seen in the chapter Testing, there are some unwanted effects when certain calls are performed. To prevent this, the API server should handle the request with more precaution. Three problems that are taken from the Testing chapter are now considered and a possible solution is given. The numbers correspond with the tests in the Testing chapter:

4. **Problem**: The score of a house can be set lower then 0.
   Solution: The calculation should first be made and if the score of the chosen house would go under 0, the server should respond with an error status code such as Bad Request. The score of that house should then be left unchanged.

6. **Problem**: New houses can accidentally be created when the `house_name` parameter is not set correctly.
   Solution: The server should check if the `house_name` parameter matches with an existing house and otherwise respond with an error status code such as Bad Request. A new API entry point should be created which sole purpose is to create new houses.

7. **Problem**: A mutation can be done without specifying a reason.
   Solution: The API server should check the input and throw errors when not all fields are specified.

8. **Problem**: A mutation can be done without specifying an issuer
   Solution: The API server should check the input and throw errors when not all fields are specified.

10.4 Scaling up to four tanks

The prototype we made was created for only one house. If the final system were to be made, there would be four tanks. There are several options to realize this.

The first option is to simply take the ideas behind the prototype and recreate it four times. It is the easiest but also the most expensive option.

Other, probably cheaper and more efficient options would be to use certain aspects of the prototype for multiple tanks. For example, one calibration tank could be used for all tanks to ensure the amount of water added to the tanks is the same for each of them. Another option would be to use one Raspberry Pi for multiple tanks. While these options are cheaper, they would make the system architecture more complex, because the same resources (for example one calibration tank or one Raspberry Pi) would have to be divided over multiple tanks.

10.5 Aesthetics

Right know, only the display tank has a polished design. All the other hardware is still out in the open. We have two possible design recommendations that would fix this problem.

The first is putting all the other hardware in a box behind the display tank. Th second option is to put the display tank on a table. All the other hardware besides the water reservoir could be put inside a box behind the display tank. The water reservoir could than be put under the table.

The current water container of the display tank is a vase. This should be replaced by a plastic cylinder if the final system where to be build, so that a little hole can be made in the tank for the tube that connects the calibration and display tanks (right now, the tube goes down from the lid of the display tanks to where the point levels begin),

Finally, the LED's that give color to the water are basically separate from the system and controlled by a remote. If the system were to be built, they should be controlled by the Raspberry Pi.

10.6 Power source for pumps

The pumps are powered by batteries for this prototype. Of course batteries will b drained over time, so it might be better if they were to be powered by a power outlet.

10.7 Advanced system state analysis

The current version of the system only detects if the water level changes, but not what the cause is. A leak for example would prove to be problematic, because the system will continue to reset over and over again. Furthermore, someone could walk into the system, which could cause the water to move around and force a reset while it might not be necessary at all.

This all would mean that the system would benefit from not only using the ultrasound sensor to monitor the water level, but also use it to try and determine the cause of the changes in water level. For example, a decrease in water level is likely caused by a leak. If the system were able to detect this, it could notify the maintainer of the system.

10.8 Finalising the point system

Right now, the bars of the canvas plug-in and the display tank show the relative score of a house compared to the other houses. One of the supervisors already had another point system in mind, but we did not have the time to implement this. So, after the point system itself is finalised, it should still be implemented in both the software and hardware parts of the system.

11 PROCESS EVALUATION

11.1 Organisation

11.1.1 Global organisation

The basic idea of the organisation was that we followed the principles of scrum. The time was divided in a design phase of 2 weeks and 3 sprint (also 2 weeks each). For each sprint, an end product was laid down which will be discussed in their respective parts.

We met daily or once every other day depending on how much there was to discuss. We started with a Trello board to track our progress, but in the end we moved to a textual list of ToDo's for each sprint which we kept track of in our discord server.

Because the whole system could be divided in a software and a hardware part, the group was also split in two to focus on these parts. Egbert and Jesse worked on the software component while Alain and Bob worked on the hardware component.

As for the "global" responsibilities, Alain was responsible for sending mails or Whats App messages to our supervisors and Jesse was responsible for creating invites to the meeting and creating date pickers to pick a date. Egbert was responsible for creating the poster. Finally, Egbert and Alain were responsible for giving both the poster and the final presentation.

11.1.2 Software organisation

Where the complete product was divided into software and hardware, we divided the software further in the front-end and back-end. Our initial planning is shown in table 11.1 Being the more experienced programmer, Egbert started with the creation of the Rest server while Jesse started with a new React page. After a few sprints, when the React page was almost ready and we were busy with implementing LTI to get the page inside Canvas, we realized that we needed a different architecture. The React page could not be server-less. That is why Egbert created a new version of the application, this time without React. With this application, the LTI flow could be implemented and the plugin was available from within Canvas.

During the project we mostly communicated through WhatsApp or Voice chat when direct contact was easier. We also had contact with Ard Koster, who helped us implementing the LTI protocol. We both had a physical meeting with Ard as well as online contact.
Egbert: Back-end creating, API design, LTI implementation

Jesse: Front-end design, front-end creating in React, report documentation

Phase	Week/ end date	Goal
Design period	Week 3 18-09-2020	- Mock up sketches for web application - Information communication schemes
Sprint 1	Week 5 02-10-2020	- Finished setting up web server, database and Canvas course - Basic web application, able to connect to the database
Sprint 2	Week 7 16-10-2020	- Finished the interaction between Canvas' API and the web application - Created a web application that can be tested
Sprint 3	Week 9 30-10-2020	- Finished the complete canvas plug-in

Table 11.1: software Planning

11.1.3 Hardware organisation

Working on the he hardware part was made more complex due to the outbreak of the corona virus, which basically meant we could not work on the system together on the university. In the beginning, we both worked on the analysis of water level sensing technologies and testing potential sensor candidates. Afterwards, Alain was mostly involved with designing the principles behind the hardware component, analysing potential problems that may arise, thinking of solutions to them and setting up the system tests. Bob did most of the hardware implementation work and wrote the code for the system. He in turn also looked if Alain's solutions to potential problems were feasible. He also made the system look nice and conducted the tests. A summary of the responsibilities is the following:

Alain: Literature analysis, sensor testing, system design and analysis, designing tests, video script, documentation

Bob: Literature analysis, sensor testing, system implementation, system aesthetic, conducting tests, shooting video

Phase	Week/ end date	Goal
Design period	Week 3 18-09-2020	- Design for the water tank - Decide on which sensors to use - Decided how to adjust the water level
Sprint 1	Week 5 02-10-2020	- Proof of concept version of the system with manual score management
Sprint 2	Week 7 16-10-2020	- Prototype of the final version with correct materials and connected to the software component
Sprint 3	Week 9 30-10-2020	- Finished the final version of the hardware component

Table 11.2: Hardware planning

Table 11.2 shows the planning of the hardware part, which was followed pretty well. We mostly communicated using Whats App and through voice chat.

11.2 What went well and pitfalls

Overall, we are happy with our process. The ambiance in the group was very good and the meeting where fun and productive. Communication between the group members was also very good. While the communication was good, we believe that we could have worked a bit more efficient if we could work in-person with each other.

Finally, the contact with our started great and very enthusiastic. However, after a few weeks the communication hit a rough patch. Our supervisors had some personal issues and were plagued by illness. We are not sure if we maybe should have pushed for more communication when that happened. Near the end of the project, we had regular contact with our supervisors again.

During the period of low communication, we continued working according to what we thought the system should be. Luckily, our supervisors had the same basic ideas about this. Unfortunately, after we had established contact again, we could not implement some specific aspects that they wanted (such as a different scoring system).

12 CONCLUSION

We implemented a system that could facilitate a House Cup for the houses of TCS. The system consists of two main parts: a canvas plugin that serves as the point management system and a hardware system that is used to physically represent the scores using tank filled with colored water.

Because the software and the hardware components are two loosely connected systems, the project was also divided in two parts. Two people worked on the software part and two people worked on the hardware part. Because the whole system was broken up into these two smaller, modular parts, the responsibilities could be clearly divided between us.

Despite having to work remotely from each other due to the coronavirus, the system fulfills nearly all requirements that were laid out in the beginning of the project. For the requirements that were not satisfied or other flaws in the system, recommendation have been made on how they could be implemented or solved.

A HARDWARE APPENDIX

This appendix shows the hardware components, including the schematic overview and the connections of the components to each other and to the Raspberry Pi.

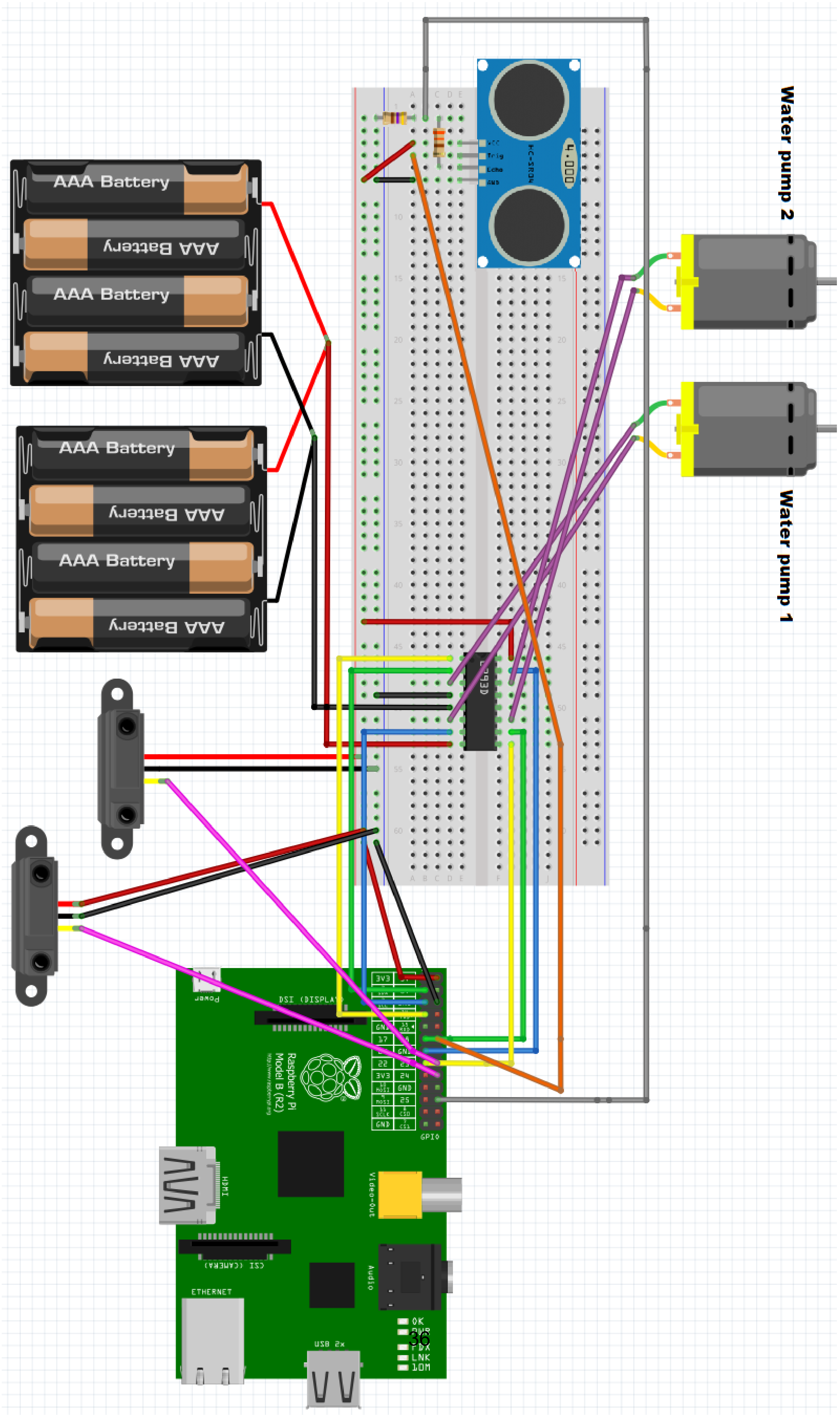
Raspberry Pi 3 Model B+:
https://www.kiwi-electronics.nl/raspberry-pi-3-model-b-plus

HC SR-04:
https://www.kiwi-electronics.nl/ultrasonic-sensor-hc-sr04?lang=en

Motor driver L293D:
https://www.hobbyelectronica.nl/product/l293d-motor-driver/

Infrared Proximity Sensor:
https://www.hobbyelectronica.nl/product/infrarood-obstakel-vermijdingsmodule/

Peristaltic Pump 12V:
https://www.hobbyelectronica.nl/product/peristaltic-pomp-12v/


AAA Battery
Water pump 2
Water pump 1
HC-SR04
L293D
Raspberry Pi Model B (R2)
GPIO
DSI (DISPLAY)
Power
HDMI
CSI (CAMERA)
ETHERNET
USB 2x
Video-Out
Audio