Umeå universitet **2020-03-26**
Department of Department of Applied Physics and Electronics
Project report

IOT_VT20 - Project Report

Design och tjänsteutveckling för Internet of Things, Spring 20, 7.5 Credits

HappyCat

Aiming grade: VG

Authors Signe Bennmarker, Frida Aringskog, Linus Lundqvist, Adam Hörnemalm

Email sibe0038@student.umu.se
frar0030@student.umu.se
adho0014@student.umu.se
linus.lundqvist@gmail.com

Introduction

Have you ever thought about what your cat is doing when you are not home? That question is the foundation of this project, and also the question that the end product of this project is trying to answer. The product created to answer this particular question is called HappyCat and it is a system that monitors a cat when it is home alone. For the user, it is a simple setup consisting of four hardware components and a user interface. The user interface makes it easy for the user to monitor the system from afar. It also offers a way of making the experience more personal by customizing the settings, e.g. the user can enter the cat's name and get more personal notifications. After some consideration, it was decided that the most important information about the cat would be its food, water and toilet habits. Thus, these are the areas that HappyCat monitors. But, HappyCat does not only provide the user with information about how much food there is left in the food bowl and how frequently the cat uses the litter box. It also has other features, like providing a live stream of the cat when it is eating, estimating how much cat food the user has left at home and sending out cute tweets about the cat. One of the more interesting services provided by the product is giving the user a daily health check on the cat. In conclusion, HappyCat is a perfect product for people who are invested in their cats because it gives them valuable information about the cat's wellbeing.

Overview

The goal was to create a product that can monitor a cat's behavior when it is left home alone. The areas that were decided as most important to monitor are the cat's food court, in other words, the cat's food and water bowls, and the cat's litter box. Other areas of the house were deemed unnecessary to monitor for the function of the product. The project consists of a couple of main components. It includes a raspberry pi that functions as the brain and server, three ESP32 that forwards information, a couple of different sensors that collect data and a camera. Further evolution of the project was considered, e.g. connecting a cloud platform to store historical data. This historical data was later supposed to be used to detect anomalies in the cat's behavior. Sadly, this turned out to be too time-consuming and is not a part of the end product of this project.

The raspberry pi is the most important component in the project. The pi acts as an MQTT broker, and all of the communication in the system passes through this broker. The Pi also manages the node-red server and the project's user interface. The Raspberry pi is connected to two units, a motion sensor, and a pi-camera. The camera's function is to record the area around the cat's food bowls. The motion sensor was added later on in the project and is used to detect motion around the same area. The information from the sensor is used to send notifications to the user when the cat is hanging out around the food court.

In this project, three microcontrollers called ESP32 are used. Two of them are connected to a load-cell and a load-cell amplifier each. These ESP32 are responsible for collecting weight data from the food and water bowls and then forwarding that data to the raspberry pi via MQTT. The third ESP32 is connected to a motion sensor located at the cat's litter box. It sends information about movement in the litter box to the raspberry pi via MQTT.

Theory

The following section will contain a short description of the most important devices and software used to create the product.

ESP-32

ESP32 is a series of, low-power systems on a chip microcontroller. It is low-cost and has integrated Wi-Fi and dual-mode Bluetooth. Except for the actual ESP-chip, the module contains a flash memory card for program code (520 KiB) and the Wifi and Bluetooth antenna. This module is mounted on a PCB, where external components like LEDs, pushbuttons, LCD screens, and temperature sensors could be connected. These are all controlled via close-to-hardware programming.

The manufacturer supplies a development package called ESP-IDF. To have the best possible control over the hardware, this can be used directly to develop applications. To avoid disadvantages like extensive development time and having to re-implement some usual functions, there are software layers that lay on top of ESP-IDF available.

For different parts of the project, different software layers have been used. For the ESP32 where a PIR sensor is mounted, the software layer that has been used is called mongoose OS. For this function, the code is written in javascript.

As for the two ESP32 using load-cells to measure weight C code was written using the Arduino IDE.

Raspberry Pi

The organization that created the Raspberry Pi wanted programming to become more available for a bigger audience. And thus, they created the RPi. The Raspberry pi is a small computer with a big area of use. For example, it is commonly used as servers, media centers or as a part of smart IoT-products. The device used in this project is a Raspberry Pi 3 Model B. Newer models of the raspberry pie do exist, but the RPi 3 Model B is sufficient enough for the purposes of this project. The RPi 3 is a quadcore computer with 1GB Ram. The operative system used on the device is Linux. There is no operative system pre-installed on the device, this has to be inserted in the form of a prepared operating system SD-card. Raspberry Pi 3 is equipped with a lot of different features. The ones used most frequently in this lab are wireless LAN, HDMI, 40-pin extended GPIO and USB-ports. The 40-pin GPIO made it possible to connect devices such as a PIR-sensors directly to the RPi. As mentioned earlier, the Raspberry pi in this project acts as a broker. The client-server platform in use is called MQTT.

MQTT

MQTT is a messaging protocol that uses subscribe and publish to send messages between clients. It is a lightweight messaging protocol which makes it ideal for IoT-projects since it does not require a large bandwidth. It uses the transport protocol TCP/IP for reliable data transfer and the port 1883 is reserved för

MQTT-messages. MQTT has a server-client architecture where a server, often called a broker, distributes messages between clients. As mentioned earlier the clients use “publish” and “subscribe” to send messages to each other. This means that one of the clients subscribes to a topic whilst the other client publishes messages on said topic. In this way, data can be transferred between clients connected to the MQTT-broker.

NodeRed

NodeRed is a flow-based, visual programming tool part of the JS Foundation. It has a browser-based editor which makes it easy to set up and use. The term flow-based refers to how the programming tool works. The developer simply creates a flow using nodes, either from a palette or by creating their own, and wires them together. Every node in the flow has a specific task that it performs on some type of data before passing it on to the next node in the flow. The code language used in node-red is JavaScript. There are two great assets in node-red. It is very easy to use MQTT to subscribe to topics and publish messages. node-red also provides a way to create an elemental but functional user interface.

Motion & pi-camera

Motion is an opensource software that detects motion, only by using a camera. It is compatible with different types of cameras, e.g. a web camera and a pi camera.[1] The software detects motion by comparing the pixels in the video from frame to frame. If the pixel-value changes from one frame to another then it is registered as a motion. The motion software is easy to install on the raspberry pi using the aptget install command. To make changes in the software, simply access the configuration file through the terminal. The configuration file makes it possible to makes changes in the software, e.g. the number of frames per second, saturation and hue, to stream on localhost or not, etc.

Sensors

In this project two different kind of sensors are used, PIR sensors and load-cells. The PIR-sensor is a motion detector that uses IR to detect movements in the surroundings. The sensor uses the changes in the infrared radiation that bounces at it to determine when there is motion.[2] Different objects have different temperatures, e.g. a human have a warmer temperature than the wall behind it, and different surfaces, e.g. stone, wood, fabric etc. These changes in temperature and surfaces creates changes in the infrared radiation and these anomalies are detected by the sensor and noted as movement.

The load-cells are used to determine weight. The load-cell used in this project can manage weights up to 5 kilos. The load-cell measures weight when the iron rod is bent. This means that the load-cell on its own is not very useful and that some extensions needs to be added. In order for the load-cell to work some kind of scale has to be built, for more information see the implement section. Another necessary extension is a card that amplifies the signal from the load-cell. In this project, an HX711-chip was used for this purpose.

---

[1] Motion, *About Motion,* (accessed 2020-03-18)
[2] Wikipedia, *passive infrared sensor*, (accessed 2020-03-26)

Implement

As mentioned in the introduction the end product consists of four hardware components, the raspberry pi, and three ESP32. Each of these components is responsible for collecting some type of data. In the initiation of the project, not all of the sensors were available, and therefore the work on the project started with the equipment at hand which in this case was the raspberry pi and the pi camera.

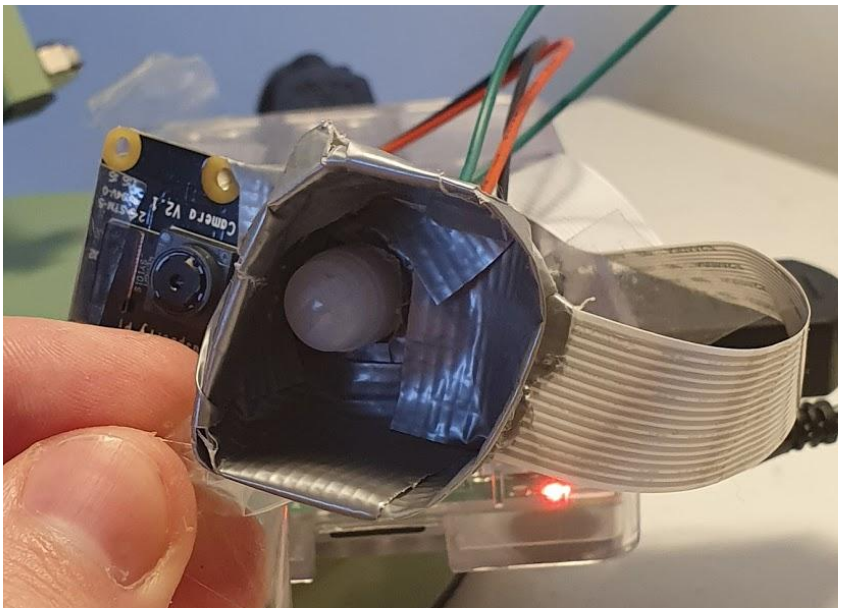
The pi camera was easy enough to install to the raspberry pi since the pi has a special slot dedicated to cameras. After the hardware installation, some updates had to be run on the Raspberry Pi to enable the camera function. To make sure that the camera worked the raspistill-command was used. This is a terminal command that uses the pi-camera to take a still picture. After these configurations the basic camera setup was done, now the hard work started. The first thing to consider was how to get the output from the camera to show up on the node-red UI. In the beginning, the intention was to use the camera as both a motion detector and a video stream. This was to be made possible by installing the software Motion and run the software as a daemon (an application that runs in the background when the computer is turned on) on the camera. Motion provides a URL with a live stream from the camera. This was very useful when it came to getting the live stream up on the node-red UI. In node-red a UI-template was used to succeed with this. Though Motion was very beneficial to use for this purpose, it could not be used to alert the user of motion in the picture. Because if this, a PIR- sensor was used instead.

```
<html>
<style>
img {

    padding: 4px;
    width: 300px;
    height: 240px;
}
</style>

<script type="text/javascript">

document.getElementById("video").src = "http://192.168.0.125:8081";

</script>

<div style = "margin-bottom:40px;">
    <img src = "" id="video">
</div>

```

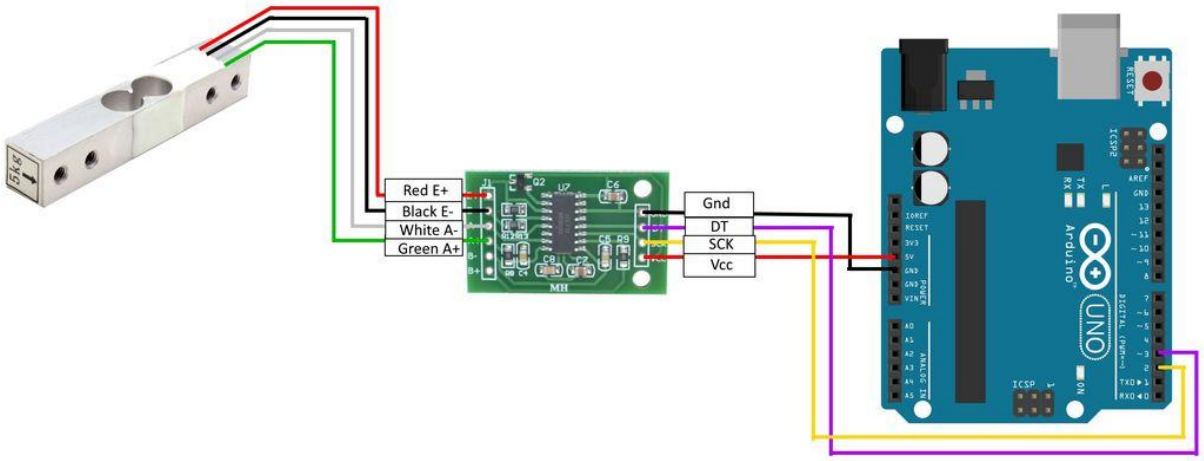
Two of the sensors in this project are PIR-sensors, these are sensors that use IR to detect motion. The setup on these sensors is pretty basic. They are set up as GPIO-pins and then the command GPIO.read is used to see whether the pin detected motion or not. If it reads 1, a motion was detected and if it reads 0 it means that there was no movement. Then the unit connected to the sensor communicates with the node-red server via MQTT, sending ‘yes’ when there is movement and ‘no’ otherwise. Some if-statements and Boolean expressions are used to make sure that the device only sends a ‘yes’-message when there is a new movement and not every time the pin reads 1.

```
let PIR = 36;
let movement = false;
let posted = false;


GPIO.setup_input(PIR, GPIO.PULL_DOWN);

function min_timer_callback(){
  if(GPIO.read(PIR)===1){
    movement = true;
  }else {
    movement = false;
    posted = false;
  }

  if(movement && !posted){
    let res2 = MQTT.pub('move', 'yes');
    print("published: ", res2 ? 'yes': 'no');
    posted = true;

  } else {
    let res2 = MQTT.pub('move', 'no');
    print("published: ", res2 ? 'yes': 'no')


  }
}

Timer.set(1000, Timer.REPEAT, min_timer_callback, null);
```

One problem with using the PIR-sensors is that they are to sufficient, they detect motion in a bigger area than necessary for this project. To deal with this there had to be some additions added to the hardware. Some duct tape and cardboard were used to create a funnel that was later attached to the PIR-sensor. This funnel is used to “box in” the sensor and thus focusing the motion detection on a smaller area.

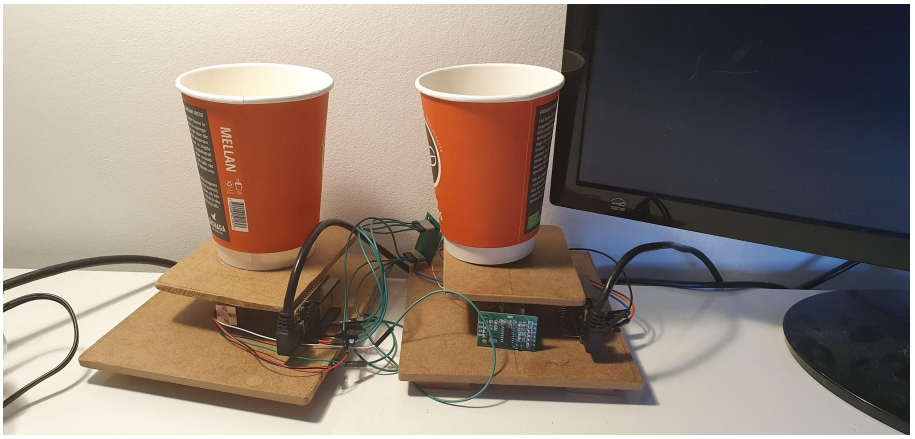


When it came to the implementation of the two load-cells the work began by soldering the HX711 chips to make them useable and later on connecting them to the load-cells and ESP32. They were then wired together according to the following diagram:



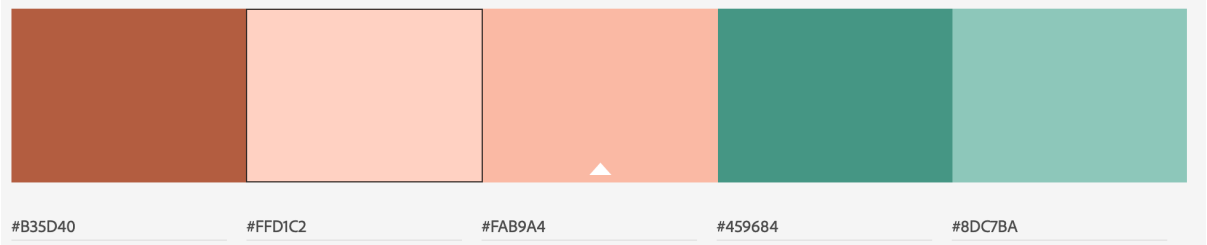
Although this diagram shows an Arduino Uno being wired, all that was needed to do was to switch out the pin numbers for the SCK and Vcc connection that were equivalent in function to those used in the diagram. It was then decided pin number 36 and 4 on the ESP32 was to be used. After wiring it all together testing was made to see whether the load-cells worked by setting up a basic program that printed a read value from the load-cell. This, of course, would not show any meaningful values since the scale had not yet been calibrated. Although, it did verify that the load-cells actually worked.

After verifying that the program responded to bending of the load-cell the process of building the actual scale began. A rough sketch was made, and from this, a wooden frame for each load-cell was cut out.



When the frames were built, the work with the code began. The choice to switch from the earlier used Mongoose to the Arduino IDE was made since the code would be easier to write in C, and at the time being it felt like Arduino IDE was better to write C code in, and the reason to write in C was made because there was a lot of useful open source libraries for the HX711 chip written in C. After the switch to C and some code were written there were immediate results. The scale started and printed out numbers, however not the correct numbers. After a lot of debugging to no success, the cable work were investigated and the problem was found, some of the soldering work was not done right. When the soldering was fixed the scale worked much better and just needed a calibration. After the calibration the scales worked better then expected and no major changes had to be done to the code.

The node-red UI was a fun part of this project, and there was a lot of experimenting with HTML and the node-red dashboard nodes while creating it. To make the layout custom made for the product, a simple graphical profile was decided on and this has been the foundation for the UI design. The graphical profile was as mentioned very simple and only consisted of a color scheme as shown below. Although using a unified color scheme to create consistency on the UI may be simple, sticking to this customized color-scheme made the User interface look more like a real webpage and not just as a node-red UI.



One of the most important things when it comes to the user interface is deciding what things are most urgent for the user to know. What information needs to be on the UI and what information is unnecessary. The end version of the user interface consisted of two pages, a home screen, and a settings page. The settings page was created to give the user more control over the product and system.

Interaction

This product is a product that is to be used in the presence of an animal. This complicates it a little bit since it is important that the animal in question don't get afraid of the equipment, accidentally ruins the equipment or accidentally gets hurt by it. After some consideration, it was decided that the best way to monitor the food and water-levels was to weigh the bowls and use the differences to calculate how much the cat drinks or eats. The motion sensors that were chosen for this particular product are two mini PIR sensors. These were preferable since the small size makes them convenient for a discrete mounting on the litter box. The decision to use twitter for notifications was made because of the want to have notifications that would show up on the user's phone and other devices and not just as a flashing notification on the node-red UI. After exploring different types of notification options like push-bullets, e-mails, texts, etc. twitter was chosen since the social media provided an easy way to connect twitter to the node-red-server, whilst also contributing to a more personalized product.

As mentioned earlier in the report the most important component in this product is the raspberry pi. If the pi loses power or breaks down, the entire system will be damaged since it hosts the MQTT-server and the node-red server. If the other components, though important for collecting data, break down, the rest of the system should still work okay. Although some measures on how to handle the loss of data input should probably be added before the product is out on the market. Right now, if the system loses data input from one of the sensors, it would forward faulty information to the user interface and thereby also provide the user with faulty information. This is an aspect that was not considered that much while developing the product but alas a very important aspect that should be considered more in future projects. Most parts of the system work without an internet connection, but it still needs to be connected in some sort of private network in order for the devices to be able to send wireless information to each other. The only feature in the system that is depending on the internet connection is the notifications since they are via tweets and twitter requires the internet.

The user interface is designed to give the user simple supervision of the system. It consists of a homepage and a settings page. The settings page gives the user the chance to make the experience more customized. The user can enter the cat's name which will make the daily checkup more personalized. Enter how much cat food they have at home which will give them information when the cat food is estimated to run out. And there is also an option for the user to turn off the twitter notifications. In a further development of the product, more features in the settings page could be added, for example somewhere to enter whether the cat is a kitten or full-grown cat, etc.

The homepage of the user interface uses level-graphs to display the amount of food and water in the bowls and a counter to show how often the litter box is used. These widgets make it is easy to monitor the home status from afar. The user interface also presents a live stream video of the cat's food area. This stream is always turned on, but the user is only notified when there is movement close to the camera. The user interface also has an embedded twitter timeline. This means that the user can see the tweets right there on the UI instead of visiting the twitter social media platform. The last feature on the user interface is a daily health check, a square of text that updates the user on how the cat ate, drank and went to the toilet that day. This message is scheduled to be updated every evening and the content of the message is based on the cat's behavior that day. If the cat ate and drank a normal amount the message would be positive and say that the cat seems healthy. On the other hand, if the cat strayed from its normal behavior that day the message would

warn the user that the cat might be ill. As for now, the amount of food and water and the number of times a cat should use the litter box is hardcoded for an average cat. The desire, however, was that the analysis would be based on historical data and thus be customized for each individual cat.

When it comes to security the information passed on through the system is not very vulnerable information. At least not the weight of the food and water bowls or the number of times the cat uses the litter box. Although one feature that could be sensitive when it comes to security is the camera-function. As mentioned earlier the camera runs an opensource software called motion. There are a few security-aspects in motion that are important to consider. For instance, as it is configured right now, motion allows streaming that is not only localhost. This means that any device connected to the same network and aware of the Raspberry Pi IP-address and the motion stream port can access the live stream in any web browser. This is a big security issue, but using the localhost-only setting would mean that the Livestream would only be accessed directly from the Raspberry Pi and not on the user's other devices. Therefore being able to stream outside of the localhost was deemed necessary. Another security aspect is that there is no login to the user interface. If this project were to be developed into a real product, some kind of login function would probably be added to make sensitive information like the video stream more secure.

Discussion

Overall a working product was created, and all of the hardware worked as expected. Although some issues that were encountered along the way that were not resolved and instead other solutions had to be created to work around these issues.

One of these issues involved the camera feature, or more specifically the software motion that was used on the camera. In the configuration file of the motion software, different features exist that allow the developer the option to run scripts when the software detects motion. This was a feature that was desired to use so that notifications could be sent to the user when the camera detected motion. This, however, did not work. With the help of google and supervisors, a lot of time was spent trying to solve this issue. The motion software is run from a special user on the Raspberry pi called motion. This particular user is very limited when it comes to access and it is only allowed to read and write in files that are owned by the user motion. Because of this, the problem seemed to be that of access and rights. But, changing the ownership and giving motion the access to read, write and execute the scripts did not change the situation. Therefore, new solutions on how to send out notifications when a motion was detected had to be explored. In the end, it was decided to use another PIR sensor to notify the user.

During the project, there were a few issues making the load-cells function as intended. For instance, the HX711 card that amplifies the load-cell signal did not have ready-made libraries in Javascript. Because of this, exploring other programming languages to use was necessary. After some consideration, the conclusion was that writing the code in C would be the best choice as there were plenty of libraries to choose from. But, because of this, a decision was made to switch from Mongoose to the Arduino IDE as Mongoose lacked proper support for compiling C code.

The first big problem, code-wise, during the implementation was the different speeds the HX711 chip and the ESP32 ran at. Because the ESP32 was a lot faster than the HX711 it leads to the program having many outliers when reading in values from the load-cell resulting in the program sometimes showing very incorrect values. This was solved by slowing down the ESP32 to 80MHz resulting in the program only reading in and displaying the correct values.

There were also some hardware issues when it came to the construction of the scales as well as the soldering of the HX711 pcb board. These issues were not complex in any way but wasted a lot of time since a lot of the soldering and building of the scales had to be redone, not to mention the time spent not knowing those things was the issue all along.

Since another programming language was used the network configurations and the MQTT-client setup looked a bit different from the one used in the javascript programs. After advising google and stack overflow, this issue was solved. In conclusion, after some work, the load-cells operated fine and sent accurate weight data to the MQTT-server.

Another issue that was not resolved during this project was working with the cloud service. When the camera and the PIR sensors were working somewhat faultless and the load-cells were almost working as well, different cloud services were explored. The IBM-cloud service was chosen as a good option to use since it was free and have a lot of tools for IoT-projects. Exploring the IBM-cloud and its services was a lot of fun and educational although in the end it was decided to not use a cloud service in this project. The reason for this was mainly because the cloud-service proved itself to be a bit complicated and time-consuming. The main purpose of the cloud service was to store and analyze historical data. This would mean that it would be possible to map the cat's behavior. If anomalies that is, weird behaviors were detected it could mean that something was wrong with the cat. This would have been a very cool feature to this product, but instead, it was decided to do a simple more local analysis on the node-red server that was not based on historical data.

Conclusions

Even though some of the functions that could have given this product added value, for example in form of more personalized data analyzation, were not added, due to lack of time the main objective was achieved. The main objective of the project was to create a cat monitor. This was accomplished with a satisfactory result. The sensor components were all installed and calibrated so that they in an efficient and precise way could provide the system with the required information. The system as a whole works well.

Comment

This project has been a group project with four members consisting of Signe, Frida, Adam and, Linus. When it comes to the work dedicated to this project, it was early on decided that we were to divide the workload in two. We decided to stay in the same pairs of two as in the earlier labs and divide the sensors between us. Signe and Frida worked on the pi-camera, the PIR-sensors, and the node-red-server. Signe and Frida also explored the options of cloud services, and worked with the user interface. Adam and Linus worked with the load-cells and the building of the scales, which turned out to be more work than first anticipated. Since we only had the camera to work with the first week, Signe and Frida started to work earlier than the rest of the group. This was discussed and agreed upon by the entire group. After the first week, we scheduled when we were to be in the lab. This meant that although we were working on separate parts of the project we spent equally many hours working each day.

The demo and pitch video was fun to make and no real problems occurred when it was made, the hardest part was to get the cat to cooperate.

Since the university facilities are locked, most of the report and the demo has been made from home. Because of this, it is harder to tell how much time everyone has spent on these tasks. But since the implementation of the different sensors has been divided between group members, everyone has been contributing to the writing of this report. In the end, the collaboration on this project has worked very well and every group-member has been an important and contributing factor to this project.

Reference

MQTT, *FAQ*, http://mqtt.org/faq (Accessed 2020-03-18)

Motion, *About Motion,* https://motion-project.github.io/ (Accessed 2020-03-18)

Wikipedia, Passive infrared sensor, https://en.wikipedia.org/wiki/Passive_infrared_sensor (Accessed 2020-03-26)