Surveillance camera

Webbsystem för civilingenjörer, VT20, DT514G

Alexander Benteby & Fria Khorshid

2020-11-30

Table of Contents

The idea

Our vision

We had the idea to make a surveillance camera system with motion detection that could be accessed by the user whenever they wanted. Our vision was to have a system that was always online, alert the user if some motion was detected and always record. We wanted the user to have control over the system and allow them to add other users to be able to see the camera feed, delete users and configure the system as they see fit. To implement this idea we used a raspberry pi, a pi camera and a computer. These components together made up our whole system. We will go into further detail about the characteristics of the system a bit later.

Our final product

The above description was our vision when we started out. Our final product differs from our starting vision. An example of this is that in our first draft we wanted to implement a recording function. This function was supposed to allow the user to record the stream when motion was detected by the system. This is very useful if you want to watch the surveillance footage later. However due to time constraints and some issues with the streaming from the raspberry pie to our website we had to cut this feature. We will dive deeper into the problems and how we fixed them later in the report.

Characteristics

Simple website

We wanted to create a very simple and easy to use website with as few distractions as possible. So we made our website as user friendly as possible. When opening the website for the first time the user is presented with a login screen. If the user has an account they can simply log in and access the website. If they do not they have the option to register an account. After logging in the user is immediately presented with the stream from the camera as well as a few indicators that show the status of the system. When logging in as an admin you can also see a list of users that can access the website.

Motion detection

We also implemented a system that can detect if there are any movements in the frame of the camera. How the camera detects motion is described in detail later in the report. If motion is detected then the system will turn on the corresponding indicator on the website and also send an e-mail notification to the user.

E-mail notification

As mentioned above, when the camera detects motion it will send out an e-mail notification to the user. This notification also includes a snapshot of the frame that triggered the motion detector. The only users that get the notification are the ones that have been accepted by the admin. This means that the owner of the system can choose who gets the e-mail when a motion is detected.

High fps

Unlike other surveillance systems and cameras that have very bad quality and fps, ours have a good quality stream with a high fps. Since we didn’t implement a recording function we could instead increase our streams quality and fps.

How the system works

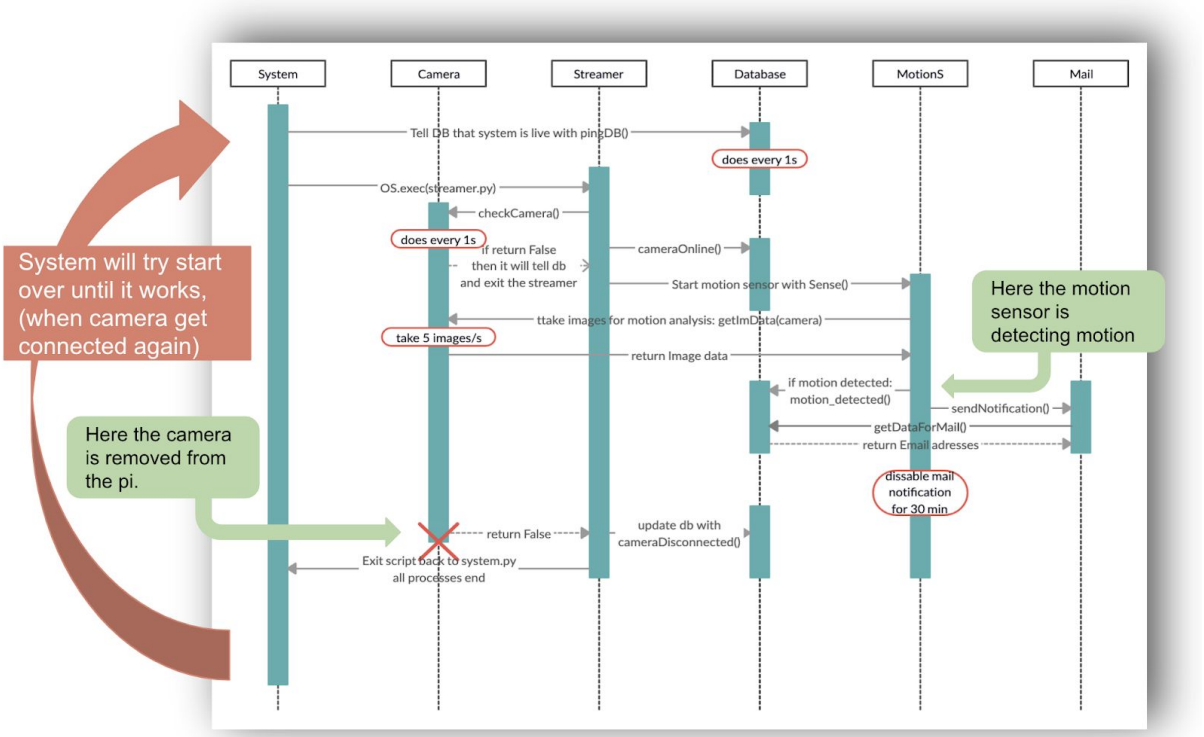


figure 1, A sequence diagram of how the system works when started. System, Streamer, Database, MotionS, and Mail are individual files which run and control its specific part of the entire system. The Camera box represents the physical camera. The green boxes show when certain events trigger different parts of the system, one being that a motion occurs, and the other being that the camera is unexpectedly removed.

System structure

The main file which controls all the different systems we've created is *streamer.py* (Streamer). When Streamer runs, it will periodically check if the camera is connected to make sure everything is running correctly. It is necessary to periodically check the camera because of the way the video streaming and the motion detection systems are implemented. When Streamer is executed with a camera connected, it will continue to run without problems, even when the camera is disconnected. Therefore, a periodic check is necessary to make sure the camera is connected and the website gets the videostream it expects.

The execution of Streamer is controlled by *system.py* (System). System contains the code which will always run at the boot of the raspberry pi, and therefore it makes sure that Streamer is always running when the requirements for it are met. System also updates the database regarding the status of the raspberry pi. Respectively Streamer updates the database regarding the status of the camera.

database.py (Database) and *kikiMailService.py* (Mail) only contain functions which are used by the other systems. The motion detection system is initiated by the Streamer and it is only running when Streamer is running.

Thread structure

In System the first thread is started. The purpose of that thread is to tell the database that the system is online. Streamer is actually not a thread running within System, instead we use the *OS*-function *exec* to run *streamer.py*, and thus System waits for Streamer to finish its execution before continuing running. The *exec*-command (the command that executes *streamer.py* from *system.py*) is within an infinite while-loop.

The next two threads are started in Streamer, after Streamer has specified the settings for the camera and started the webstream. The function of the first is to check if the camera is still connected and update the database accordingly, and the second one belongs entirely to the motion detection system.

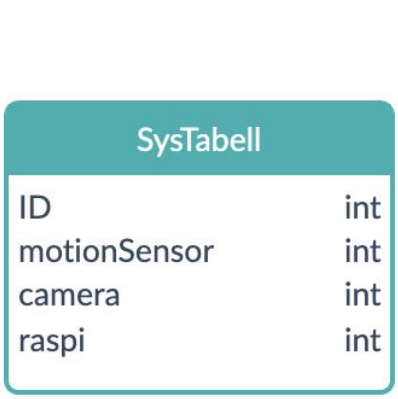
Check the status of the system

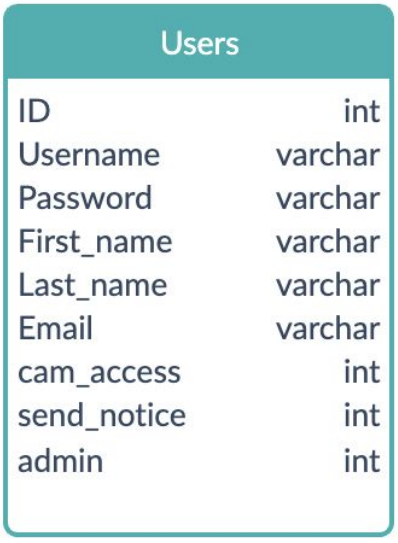
The way the raspberry pi communicates that it is online is to constantly update a space at a given ID in the database. This way, if another system wants to see if the raspberry is online, it will simply check the database two times, one second apart, to see if the number has changed or not.

If the program is started with the camera connected, and then you remove it, the system won't notice and still continue to run. For this reason, the function which checks the status of the camera, runs a simple command which returns false if the camera is disconnected. This check-function will tell the database that the camera is connected and then run this check until it returns *False* and then it will update the database about it and force the *streamer.py* to exit.

Database

Our database consists of two tables. We have one table for all the users and one table to keep track of the system status. These tables can be seen below.





SysTabell

The system table is used to keep track of the various components of the system. The first entry is used as an ID to identify a specific raspberry pie. This allows our system to have multiple raspberry pies, which means having multiple cameras set up for example around the house. The motionSensor entry is used to indicate if the motion sensor has been triggered or not. If it is triggered the database will be updated and that entry will be set to true. The e-mail notification script continuously checks the database to see if the sensor has been triggered. The camera entry is used to indicate if the camera is online or offline. The raspi entry is used to indicate if the current raspberry is online or offline.

Users

The user table is used to keep track of all the users that are registered to our system. The table has simple entries like Username, password, first and last name and an e-mail address. The interesting entries are the ones called cam_access, send_notice and admin. Cam_access indicates if a specific user can actually see the stream. Send_notice indicates if the user should receive an e-mail notification when the motion sensor is detected. The admin entry is what indicates which users have admin privileges. These users are able to edit some aspects of the system.

Motion Detection System

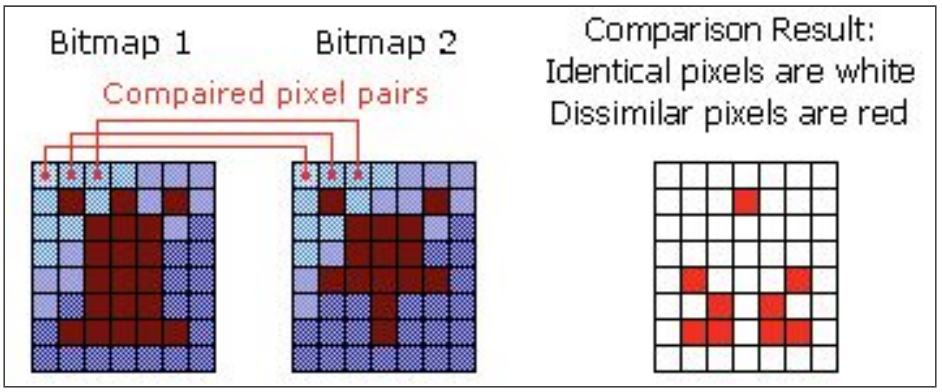


Figure 2[1], Illustrates the analytic part of the motion detection system. How it compares pixel pairs and register differences.

How the system works

Since the camera is used for live video recording, it's not possible to take regular pictures with the camera for image processing of the motion detection. This problem is solved by copying frames of streamed video from the videoport. The motion detection system (MDS) takes about three images every second with 0.3 seconds between each take. The MDS automatically resizes the images from 640x480 to 100x75 to make the analysis faster and more accurate. Each pixel is then compared to its corresponding 0.3 seconds older pixel, and if the difference is big enough the MDS classify that pixel as *changed*.
If the number of pixels changed reaches a certain level, in our case 1% of the entire image, then the MDS says a motion has been detected.
What's good about this solution is that we can easily specify the sensitivity, and easily program it to work properly as the light changes, for example raise the sensitivity at night because the frames become more alike.

Requirements for acting

In order for the system to send an email to the users when a motion has been detected, MDS needs to have three consecutive motion detections. In other words, the camera needs to detect constant motion over 1.8 seconds in order to initiate the process where motion detection is handled.

Actions

When MDS acts on a motion, it starts by updating the database so that the website can indicate that motion is positive. It then gather information for notifying the users about the motion, in this case it takes a photo so that the users can evaluate the level of emergency, it also retrieves the email addresses of the users (users labeled by the system to want

[1] Image taken from: https://support.smartbear.com/testcomplete/docs/testing-with/checkpoints/regions/how-image-comparison-works.html for purely illustrative purposes.

email-notification), and then construct an email with the image along with time and date of occurence.
The final act is that the system disable the email notification for 15 minutes, to not spam the users with mail if motion occurs over a longer period of time. When the system comes back online, it is ready to do it all again in case of a classified motion.

Conclusion

How did it go?

The project went well. We took on a bigger challenge than required. Although it became delayed, we managed to solve the problems and meet the expectations we had in the beginning. We divided the workload well and efficiently.

From the start, we were given the option to get a motion detection sensor. That would make it easier to work with, but if this would be a system developed for a company, the cost of the parts would be an important aspect and therefore we decided to solve it with the camera from the stream and our programming skills. This worked out better than we could have imagined.

We did have some problems with streaming the video to our website. We were forced to rethink our solution, and found another that works without problem

Problems

During our demonstration of the project we ran into one big problem. Our supervisor could not see the video stream from the camera when accessing our website through ssh. After doing some troubleshooting we found out that we hadn't linked the camera stream correctly to our website. The problem was that the embedded stream on our website was trying to access a local IP address. On our own computers, when connected to the same network as the pie, this "solution" worked correctly. Therefore we didn't see the problem until the demonstration. Below we talk about some solutions we tried to fix this problem.

Initial Solution

Our first attempt to get the stream working from remote connections was to embed iFrame into our website. Our raspi set up a local webpage which received the live feed from our camera. This solution works very well from local connections but since iFrame uses the local address, this solution fails with connections outside of our LAN. We did not fully understand why this solution failed after the demonstration because we did not know iFrame too well.

Second Solution

For our second attempt to solve the problem, we created a php script on our website. This script fetched the content of the website created by the raspi. Stupidly we thought that this would fix the problem since we used a php script to generate the stream, but the same problem appeared. The content returned by the php script still used the raspi's local address to fetch the live-stream, and therefore this solution failed, here we fully understood how to solve the problem; by directly connecting our php-script to the server socket of the raspi.

Final Solution

Finally we understood that we needed to fix an endpoint on the website(php script) to directly collect the bitstream generated from our raspi server socket. Since this course didn't focus too much on sockets we had problems understanding how the concept works. An alternative solution would be to simply upload each frame to the database and fetch those frames on our http-side to get a working stream. This would work and use the expected techniques learned in this course, but to us it seemed like a very bad way of implementing a live-stream, and thus we chose to work with ports.
Our raspi sets up a socket-server. Then the frames are continuously streamed to a specified port on the LAN. We listen to this port from our php-script and view the output on our site. Since we fetch the bit-stream to our webpage locally with php and return the pure content to the html-side, the stream is viewable for remote connections.

Overall we think the project went well. Choosing a system that uses a camera came with some difficulties but we managed to overcome those.