



Student Name: Matvei Popov

Title: Artificial Intelligence
Territory Surveillance System

Introduction



Security cameras' territory surveillance systems are widely used all across the world nowadays. However, these systems are not as efficient as they could potentially be. It is often difficult for their users to obtain a large spectrum of information about specific objects, areas, and subjects over a long period of time. In addition, collecting statistics for further analysis from these surveillance systems is often inconvenient and inefficient.

Let's observe an example: a manager wants to improve productivity and quality of work in the company. Using security cameras, manager wants to figure out how much time each of his employees spend working over a certain time period. Obviously, watching the recordings himself is neither feasible nor practical since such recordings could span hours, days, or even longer.

This problem can be extended to apply to security workers, scientists collecting statistics, market analysts, data engineers, private property owners, incarceration facilities and many others.

I tried to solve this problem by building a user-interactive Artificial Intelligence-powered surveillance system. This system utilizes neural networks to detect and identify human objects in surveillance cameras' recordings, then if requested provide the user with a graphical representation of surveillance statistics via its own graphical user interface (GUI) or a Telegram Bot. Below, you can see this system's logo.



Description of current system capabilities:



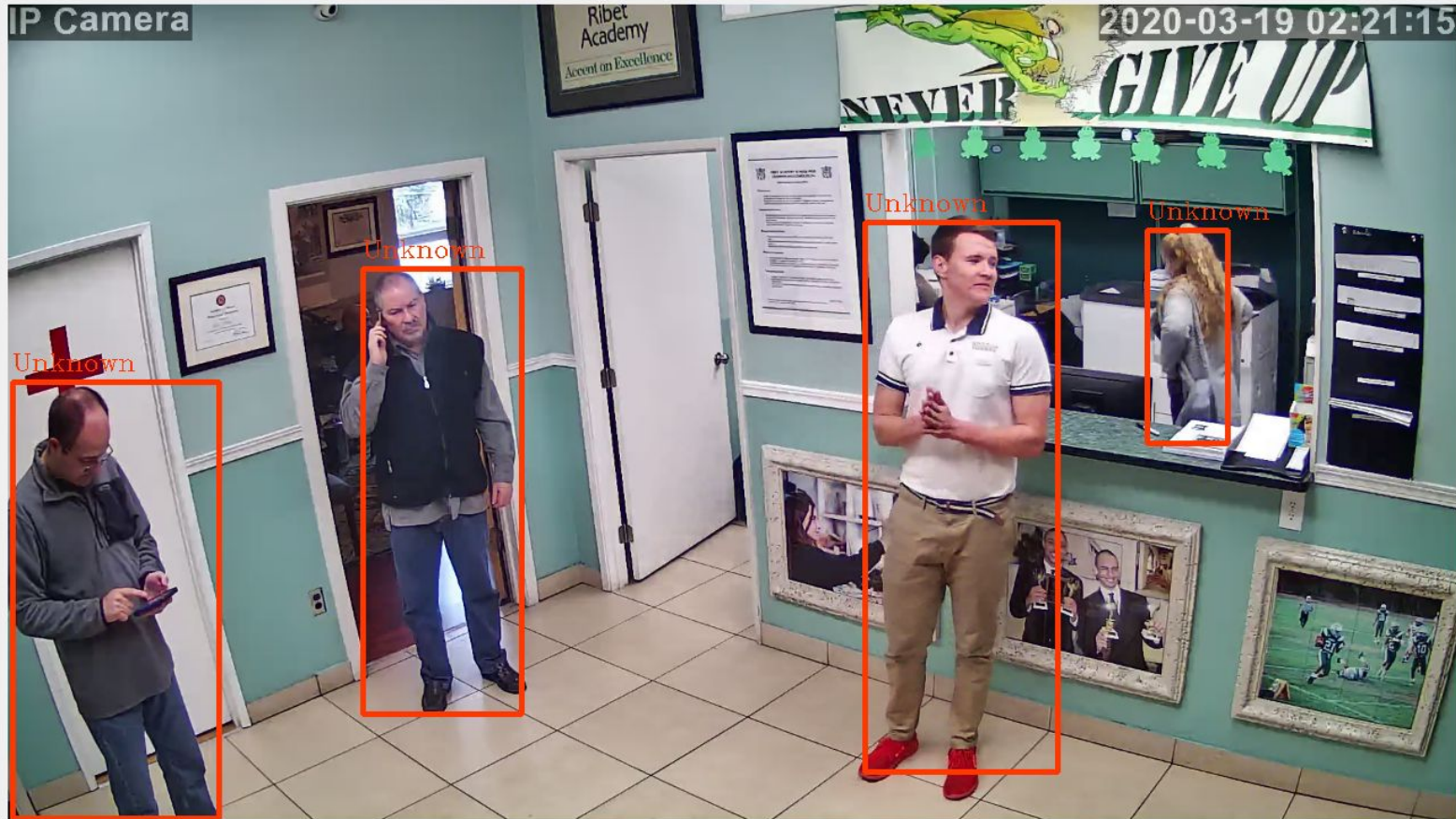
The following is a list of the current system capabilities:

1. Detection and identification of human objects in the live video stream.

The pictures on the following slides show two examples

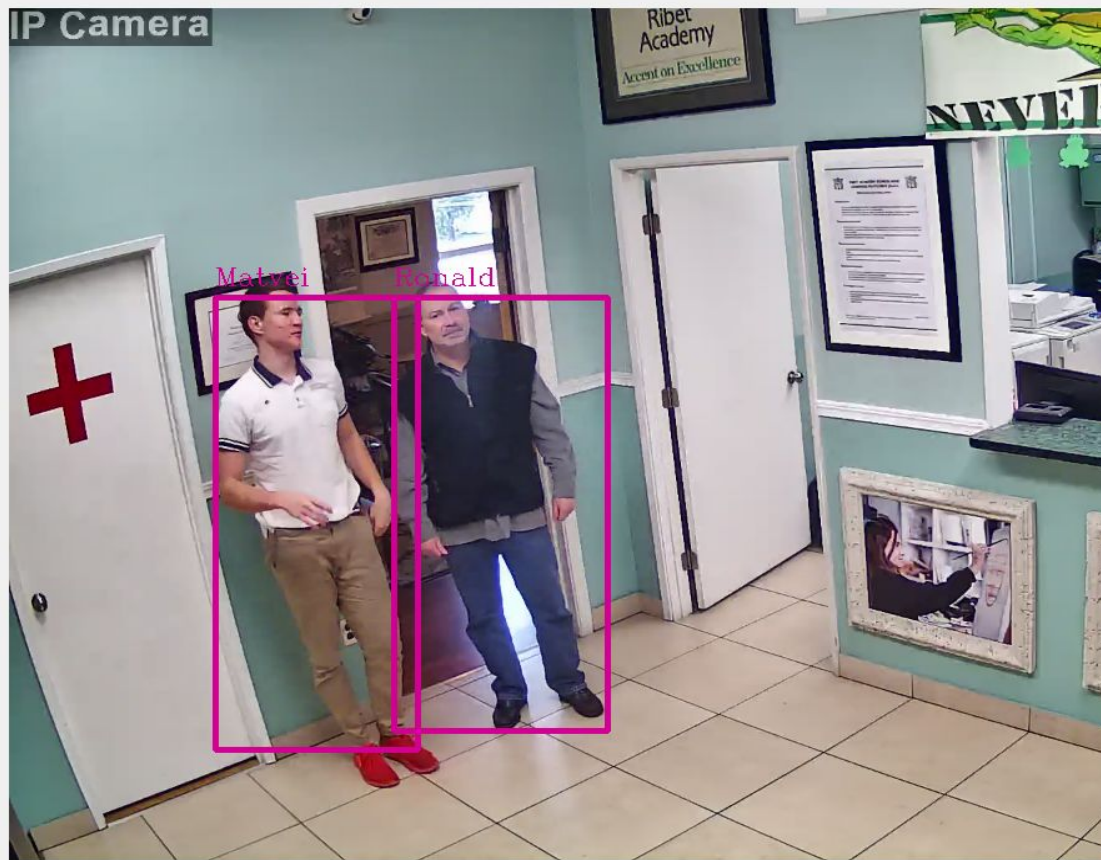
- Picture 1 shows a detected, but unidentified group of people (denoted by the “Unknown” status on each bounding box).
- Picture 2 shows detected and identified group of people (denoted by the name of each person on each bounding box).

Menu



Picture 1

IP Camera



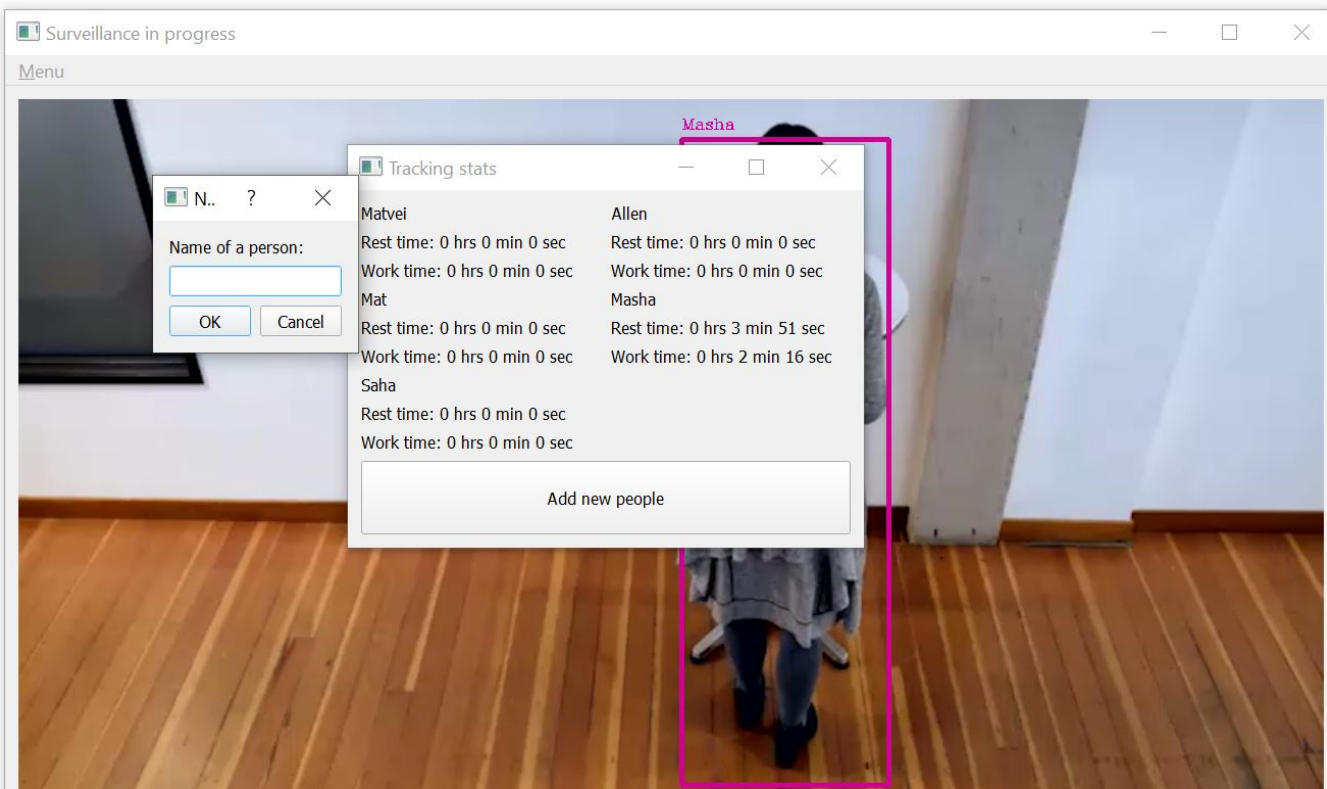
Set Area

See current presence

Shoot

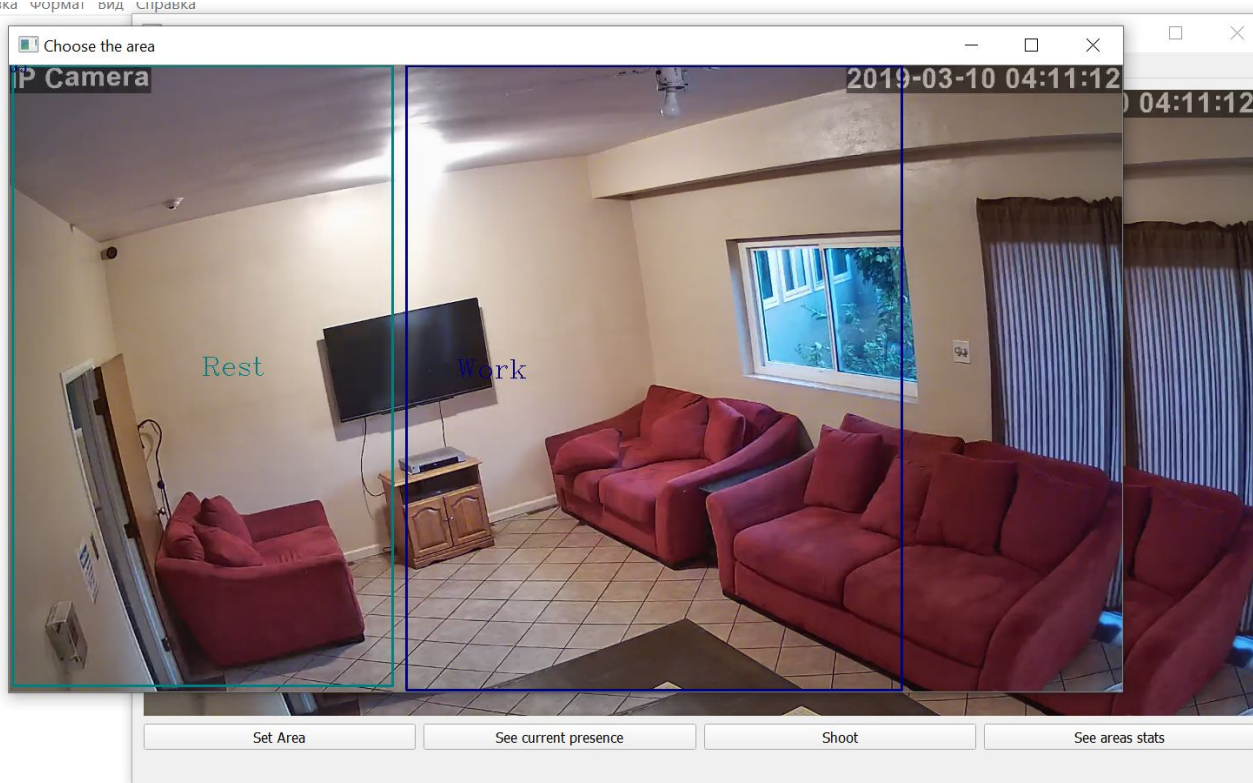
Picture 2

2. Allowing user to add new known human objects to a program (so the program will be able to identify (recognize) them in future) via GUI or a Telegram Bot (Picture 3).



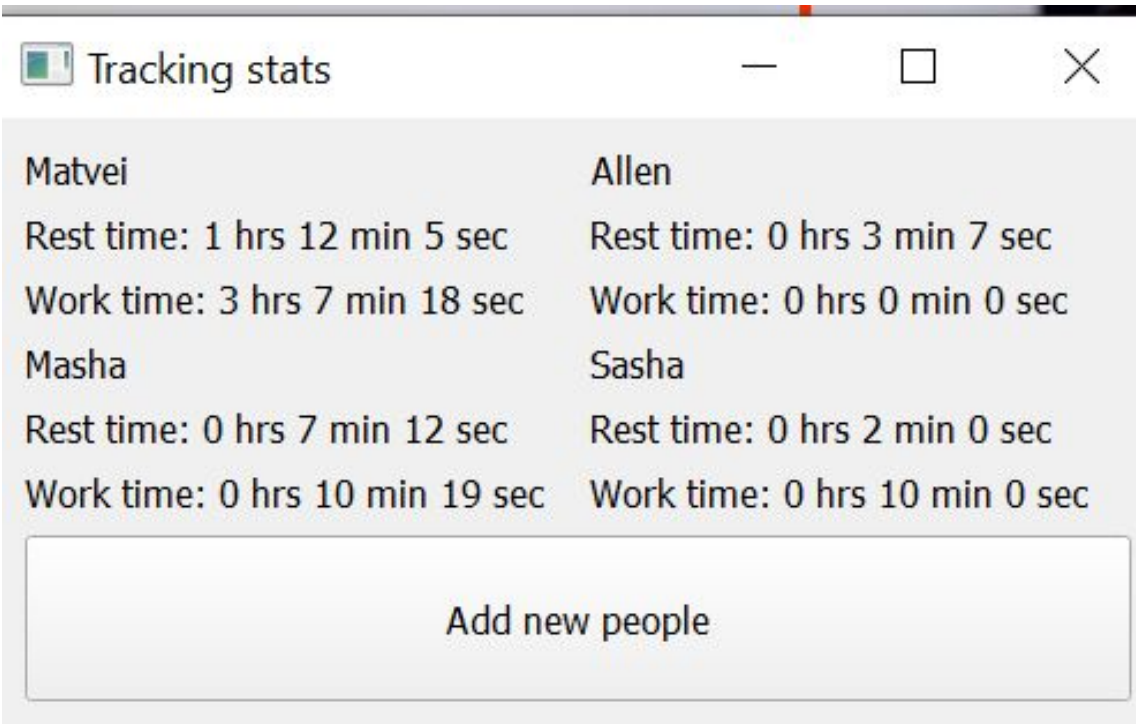
Picture 3

3. Allowing user to map and give names to the separate zones (zones user wants to keep track of) inside of a video frame, via GUI (Picture 4).



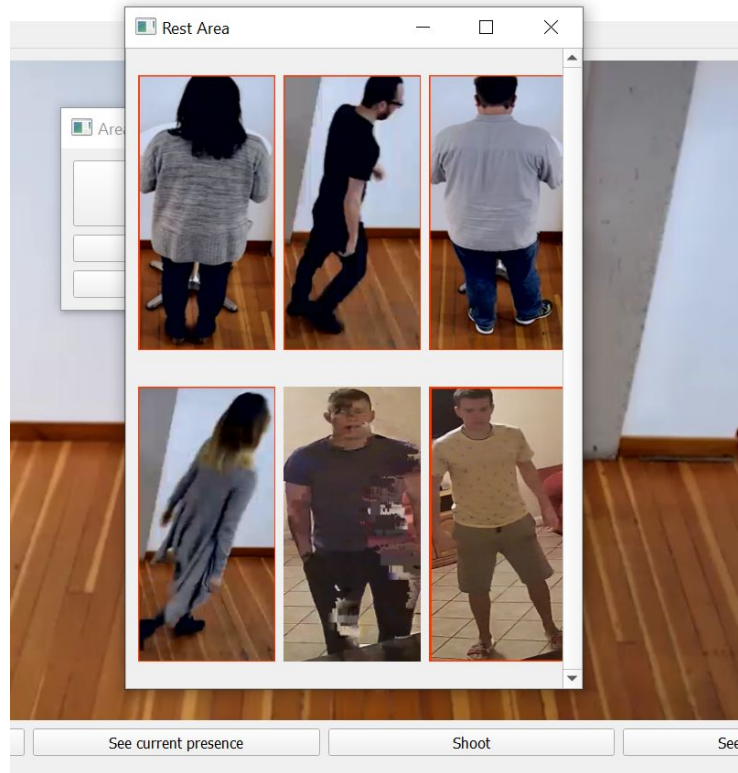
Picture 4

4. Calculating the time spent by each reidentified human within the areas specified by the user, and display it in the GUI (“x hrs x mins x sec” format) (Picture 5).



Picture 5

5. Showing user pictures of all the unknown people detected in the areas specified by the user, via GUI and a Telegram Bot (Picture 6).



Picture 6

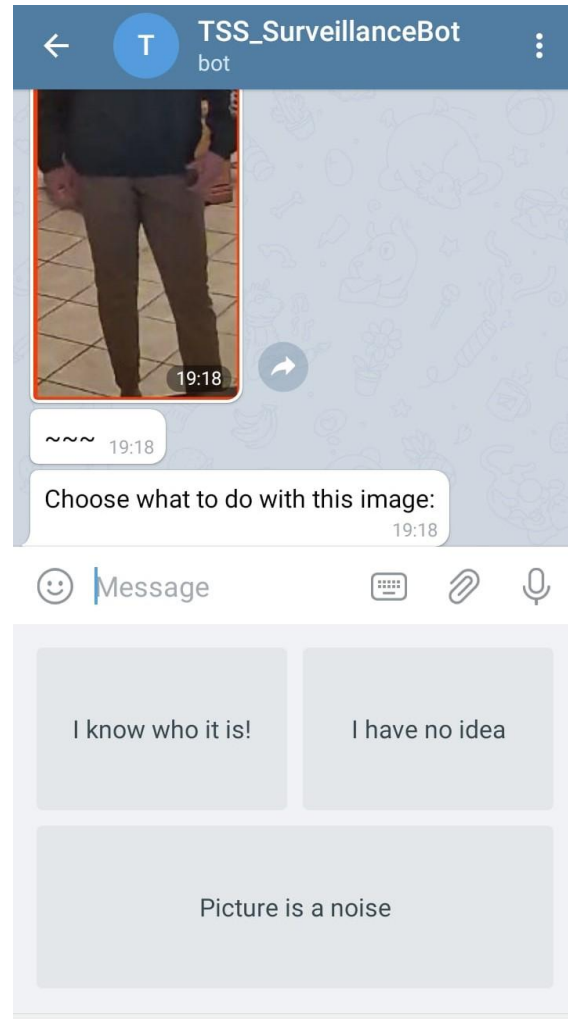
Telegram Bot overview




- System also has a Telegram bot connected to it. This bot makes user interaction more convenient (system could be controlled from your phone). Below is the quick overview of the bot functions.
 - When the system detects a new unknown human object, it adds this object to a stack of unknown objects. Then if the user types “Show last” bot pops a first human figure picture at the stack, and shows it to the user, asking what to do with it (Picture 7).
 - The user has 3 answer choices (Picture 8).



Picture 7



Picture 8

- 
- If the user will choose “I know who it is!” option, he will step by step add the new known human object into a identification network weights (Picture 9).
 - If “Picture is a noise” command is chosen, the picture will be removed. “I have no idea” option will save the image for later observation (Picture 10).



Picture 9



Picture 10

The process of getting surveillance statistics via bot looks as following:



Picture 11

Hypothesis



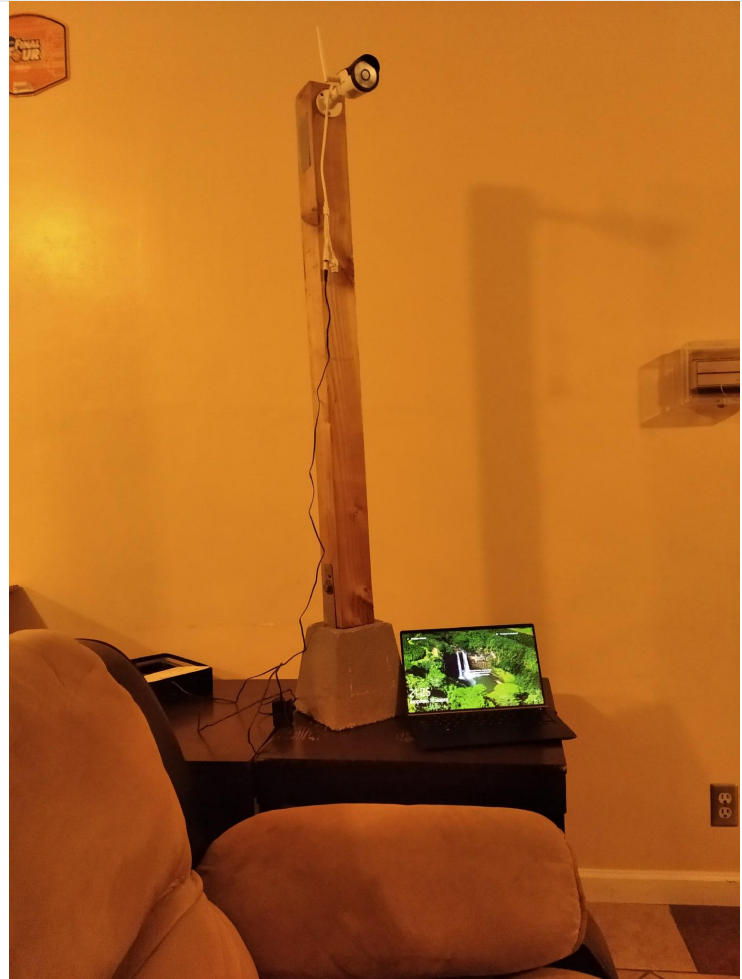
- If the steps listed below are done correctly, then the system should be able to provide the user with information about surveillance that actually represents real life, and makes territory surveillance much more efficient and convenient for people.
 - The desired system can be built in Python programming language.
 - Person detection can be performed by using either Keras implementation of the
 - YOLOv3-based convolutional neural network will be trained on a custom dataset of human figures or Intel OpenVINO's pre-trained person detection model
 - Arguments for choosing Intel OpenVINO pre-trained model are listed in Procedure section
 - Person identification can be performed by using the OpenVINO pre-trained person reidentification model.
 - All the user interaction and statistics display can be displayed via GUI built with the PyQt5 framework and a Telegram bot built with pyTelegramBotAPI.

Purpose

- This system can be applied to, but are not limited to, the following cases
 - Employees productivity measurement
 - Security services
 - Household monitoring systems (i.e. baby cameras)
 - Statistics and data collection
 - Market analysis
 - Data Engineering
 - Incarceration facilities
 - Secured military facilities

Materials

- Hardware:
 - SV3 IP surveillance camera
 - Laptop with a high-performance CPU
 - Smartphone with a portable hotspot





- Software:

- Python 3.7
- Intel OpenVino
- Keras
- OpenCV
- PyQt5
- pyTelegramBotAPI
- Flask
- Numpy
- Multiple other Python 3.7 libraries
- Telegram messenger

Procedure



Setup and software development:

- **Getting and displaying video input:**
 - Connecting a computer that runs the program and an IP camera to the same wireless network and getting the input from the IP camera, via the RTSP.
 - Reading the RTSP input from the camera with OpenCV.
 - Constructing a GUI, that displays the video captured by OpenCV and displays all the user interaction functions, by using the PyQt5 framework.

Convolutional Neural Networks for person detection and identification



The picture recognition and identification is a process that people perform on a regular basis every day. A Neural Network (NN) machine learning algorithm will be used in this project in order to teach my system to do the same thing. More specifically I will use a subclass of a NN, called Convolutional Neural Network (CNN). The CNN is a NN that has layers of neurons convolutions (convolutional layers). Convolutional layers allow us to reduce the complexity of the model, which is great for its faster and more efficient training. Most importantly, convolutional layers allow performing a great of capturing local information on the image, which is crucial in the recognition and identification processes. CNN follows a hierarchical model that works on building a network, like a funnel, and finally gives out a fully-connected layer where all the neurons are connected to each other and the output is processed.

- **Attempting to train a human object detection convolutional neural network (CNN):**
 - The first CNN that I needed to build, should have been able to detect human figures in the input video/image. In order to do that, it needed to be able to do two things. First: classification of human figures (classifying a part of an input image as a human figure), second: bounding box regression (finding the coordinates of a person in the input image).
 - After research on the best approach that I should use, I was left with a choice of whether to use YOLOv3 or Faster RCNN.
 - Territory surveillance requires very quick performance from the CNN, due to the constant movement of observed objects, and sometimes a huge amount of objects observed.
 - YOLOv3 is able to do the classification and bounding box regression at the same time, which Faster RCNN is not capable of. In addition to that, the difference in mean average precision scores of the two networks is insignificant, compared to their performance difference.

- I took the Keras implementation of the YOLOv3 (<https://github.com/qgwweee/keras-yolo3>) and trained it on my custom made dataset that consisted of pictures of me and a couple of my friends (all permissions for using pictures granted). Pictures were taken from the surveillance camera that I set up earlier, this should have helped with the classification of human objects that were recorded from surveillance cameras that hang on unusual angles (which sometimes differed from the angle at which most people pictures that were used in the COCO dataset (dataset that original YOLOv3 was trained on))

- After training, the network was tested. The results were not satisfying, the network still was not working as fast and accurate, as desired for the following reasons:
 - On many tests, where the camera was put in angles that were different from pictures in the COCO dataset, the network performed poorly and did not detect human figures. It happened because the training dataset that I used was too narrow. It consisted only of approximately 70 pictures.
 - More data for training could have improved the situation. In addition to that YOLOv3 had pre-trained weights for a bunch of other objects, so all of them were messing up the performance of a network on specifically human figures.
 - I could have added more pictures to a training dataset, get rid of redundant weights and provide a network with more training time, but at that particular moment, I found a better solution for the problem.
 - I found the Intel OpenVINO framework, that already had pre-trained models for both human figures detection and classification, so I decided to switch and use it instead.

- **The advantages of using OpenVINO models:**
 - The name of the model, that I used for human figures detection is person-detection-retail-0013, this model is trained specifically for person detection, and the average precision level is 88.62%
 - It also has a reidentification model, for human figures, called person-reidentification-retail-0076 and its mean average precision score is 82.53%
 - Both models' precision levels are extremely high, which ensures the accurate performance of them in the future, and makes it safe to build a system based on their output.
 - The main advantage of using OpenVINO models is the OpenVINO Inference Engine. Inference Engine boosts the performance of the OpenVINO models on the CPU, which is exactly what I need because most of the time this system is going to be run on the CPU.

- **Working with OpenVINO model, building the body of a program:**

The main components of the program development consist of the following:

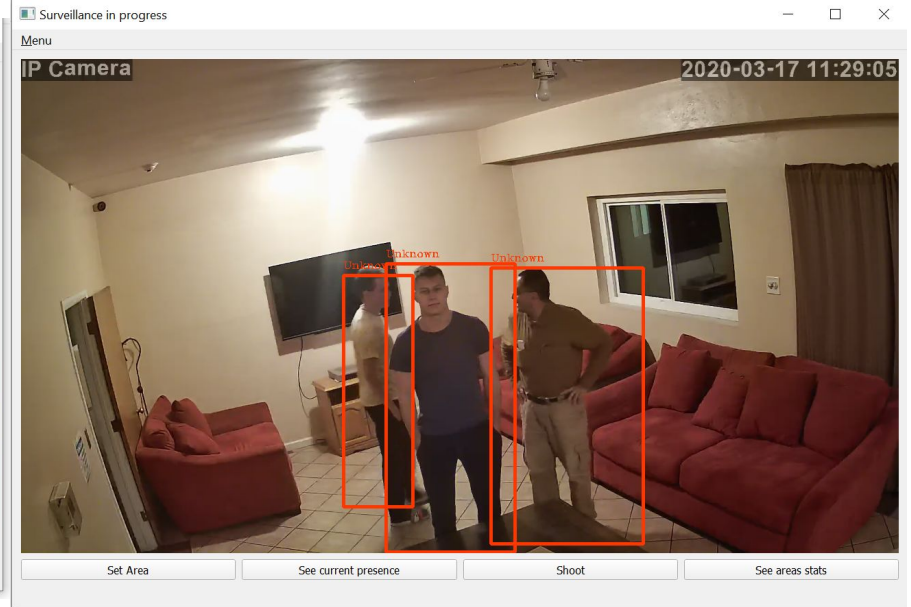
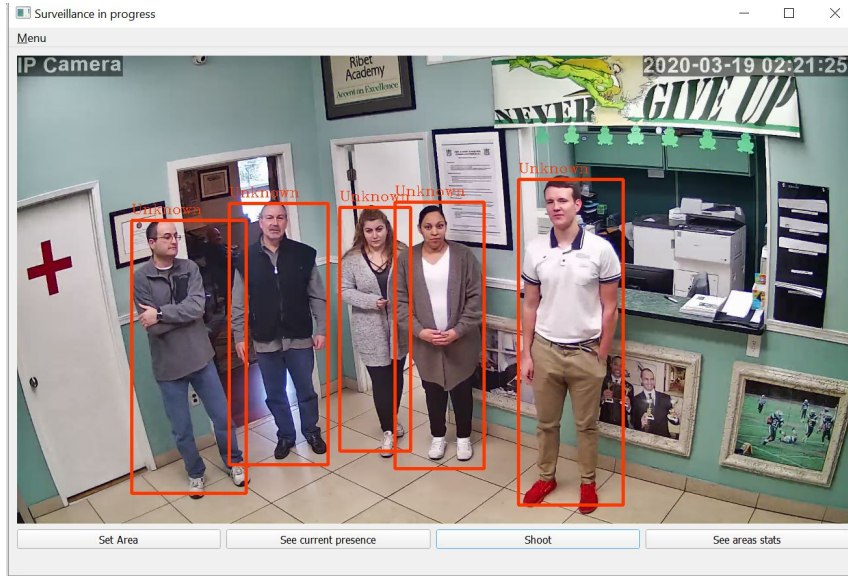
- Building an **Inference Engine Class** that boosts the performance of the OpenVINO models on CPU, applying it to each model used in the project. (From OpenVINO official sample).
- Building a **Detector** class that uses a pre-trained Intel OpenVINO model to detect human objects inside of the input video stream and returns a binary large object (blob) consisting of lists that represent every detection of a human figure in the frame. Each list has the following structure (pseudocode):
 - [image id number in the batch, (minimum and maximum x and y coordinates of a detected object in the frame)]
- Building a **Reidentifier** class that uses a pre-trained OpenVINO model to convert the input image of a person into a blob, which is a descriptor of the specific features of the input picture.

Applying the Detector and Reidentifier:

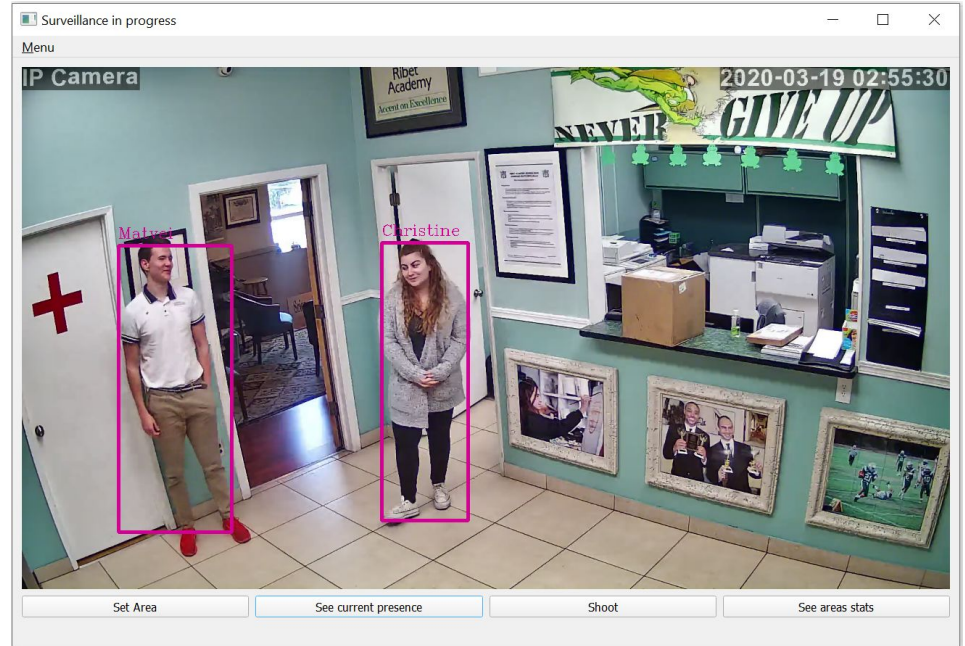
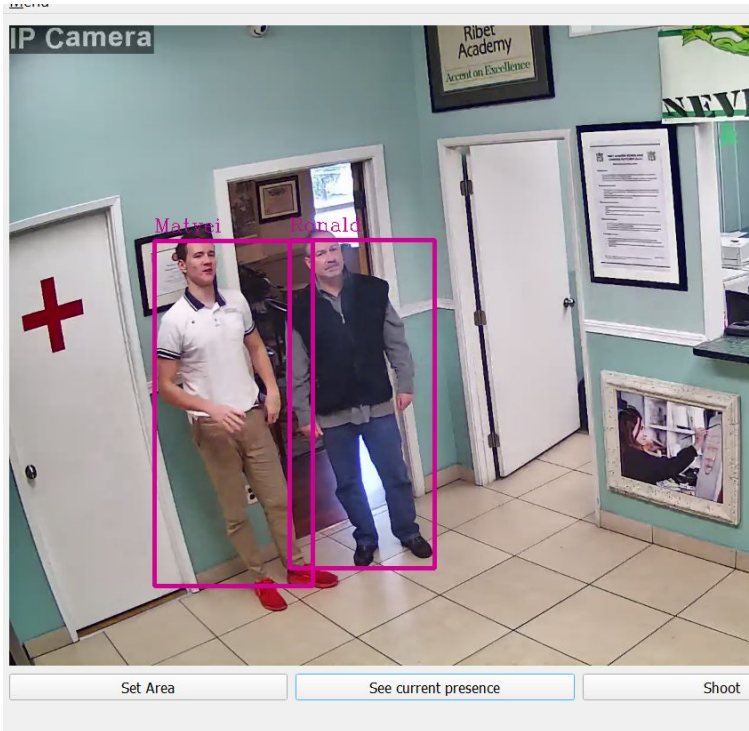
- Running all the known images of people (the ones that we want to be recognized by the program) through the **Reidentifier** network to get the data representation of the specific features of each person. Store this data in .pckl (pickle file), so next time when the program is run there is no need to repeat this process, and data could just be uploaded from the pickle file.
- Applying the **Detector** class on the input video to get the coordinates of all the people in the frame, then compare a detected person's descriptor blob to known people's descriptor blobs.
 - Each blob has a shape $[1, 256, 1, 1]$ so we can compare two blobs using cosine similarity. I will use NumPy library to calculate it. If the largest cosine similarity that the system obtains is larger than 70% then we assign the matching known person's blob name to a detected person and display this name in the GUI together with a bounding box.
 - Otherwise, just draw a bounding box and assign an "Unknown" name.

Testing

Multiple human figures detection :



Multiple human figures reidentification



● GUI and Telegram Bot development

- Building functions that allow the user to specify and give names to the areas inside of the video, which he wants to keep track of (using OpenCV for this purpose). Save the information about each area coordinates to a .csv file after each user modification.
- Building functions that save all the pictures of detected “Unknown” people within areas specified by the user, into folders with names corresponding to areas names. Store for future reference. (Compare pictures inside folders to each other using **Reidentificator** to eliminate repeating images).
- Building functions that allow the user to add new known human objects into a program from GUI.
- Storing stats about human objects in the .csv file and update them every 30 seconds of recording, for future reference. Time stats has a following format: “Time of <Human object name> in <Zone name>: x hrs x min x sec”
- Building functions that allow the user to view pictures of unknown people, detected by cameras.

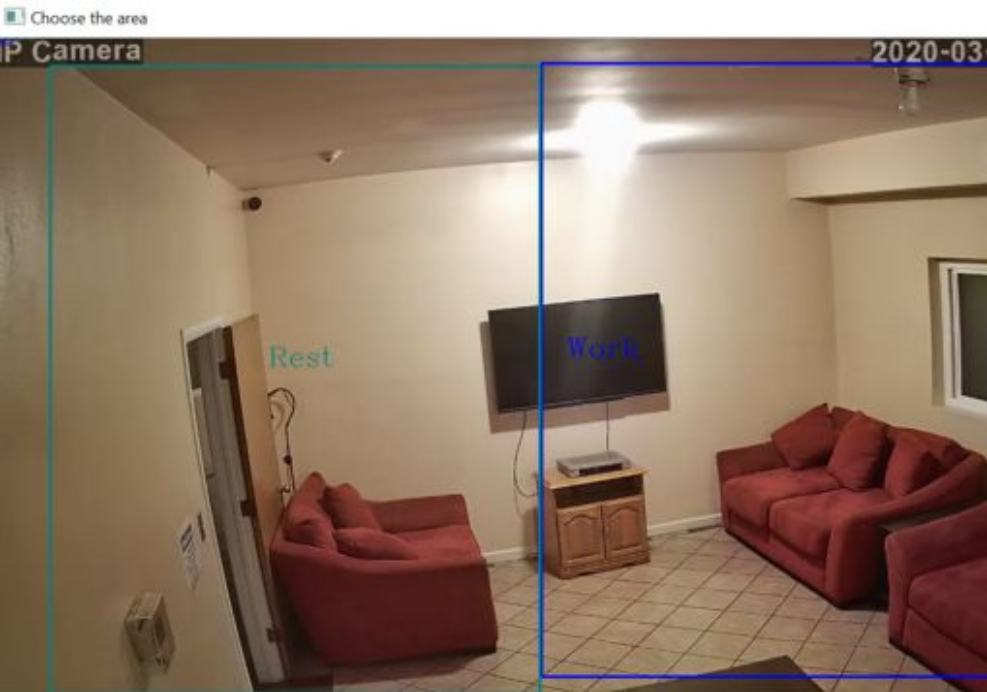
- During the process of development, it became clear that having a native GUI as the only source of user interaction is not very convenient. It would be great if a program could be controlled remotely from the user's smartphone. I decided to make this idea life and create Telegram Bot that could be run from the user's smartphone.
 - Creating a **Telegram Bot**, and setting a webhook to it via custom made domain (<http://matveitss.pythonanywhere.com/>), using Flask framework for it. I called my bot TSS_SurveillanceBot (Territory Surveillance System Surveillance Bot).
 - Adding functions that allow the user to interact with the **Bot**, and edit the data within the program, using pyTelegramBotAPI for it.
 - Separating the detection, recognition, GUI and **Bot** interactions into different processes (threads that use different CPU processes), which boosts the speed of the system performance, using a multithreading library for it.
 - Pushing the final code into a private GitHub repository, for safety purposes.

Testing

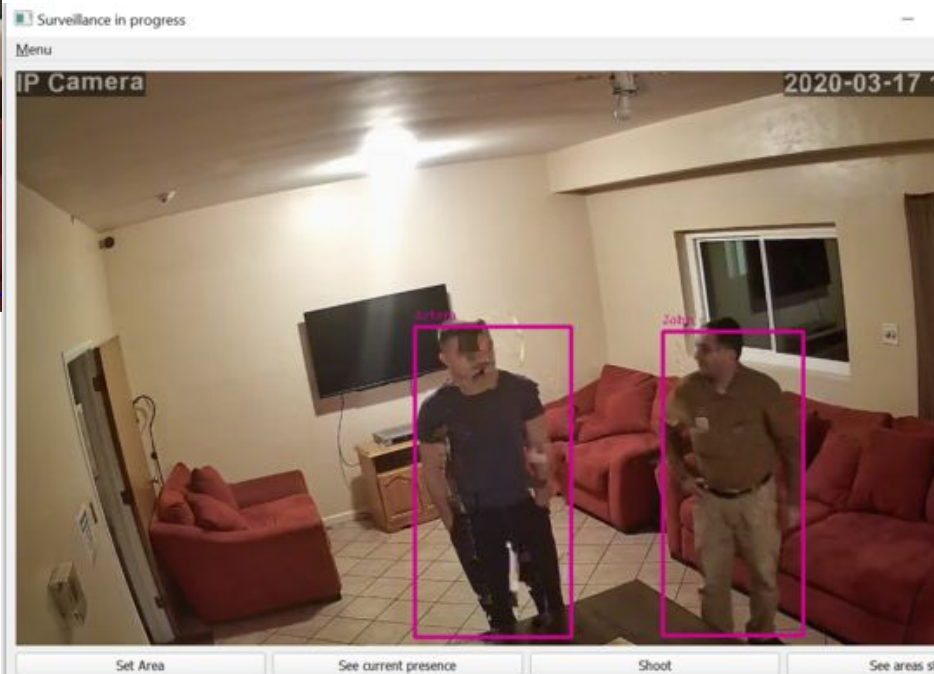


- **Time statistics test:**

- I requested the system to recognize me and a couple of other people (Picture 14).
- I split a testing room into two zones called “Rest” and “Work” in my software (Picture 13).
- I stayed in the “Work” zone for 10 minutes and stayed in the “Rest” zone for 3 minutes.
- Artem stayed in the “Work” zone for 3 minutes and stayed in the “Rest” zone for 10 minutes.
- John stayed in both zones for 6 minutes and 30 seconds. (Picture 14)

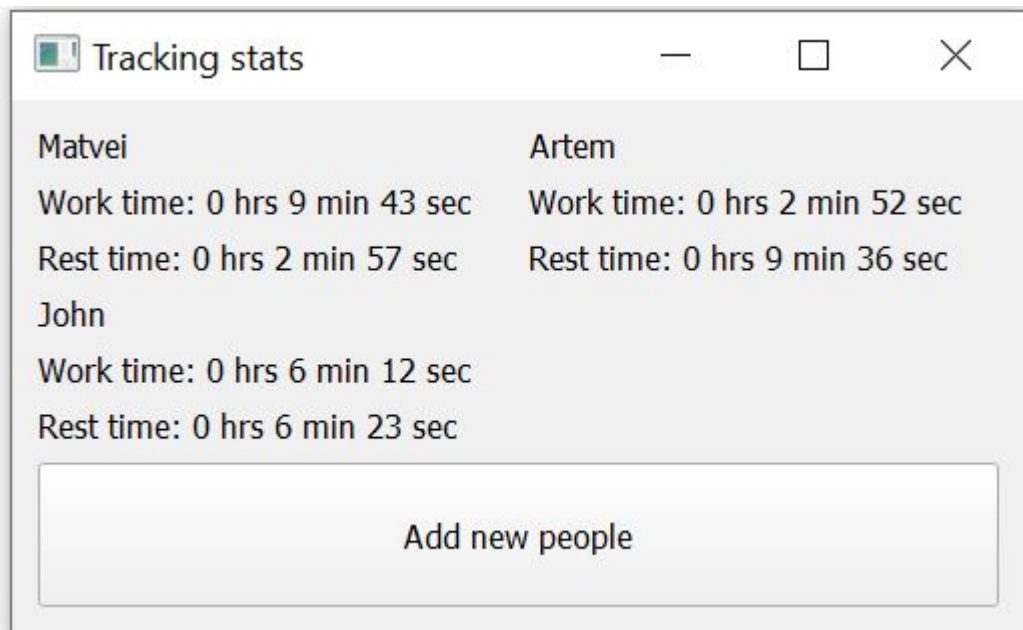


Picture 13



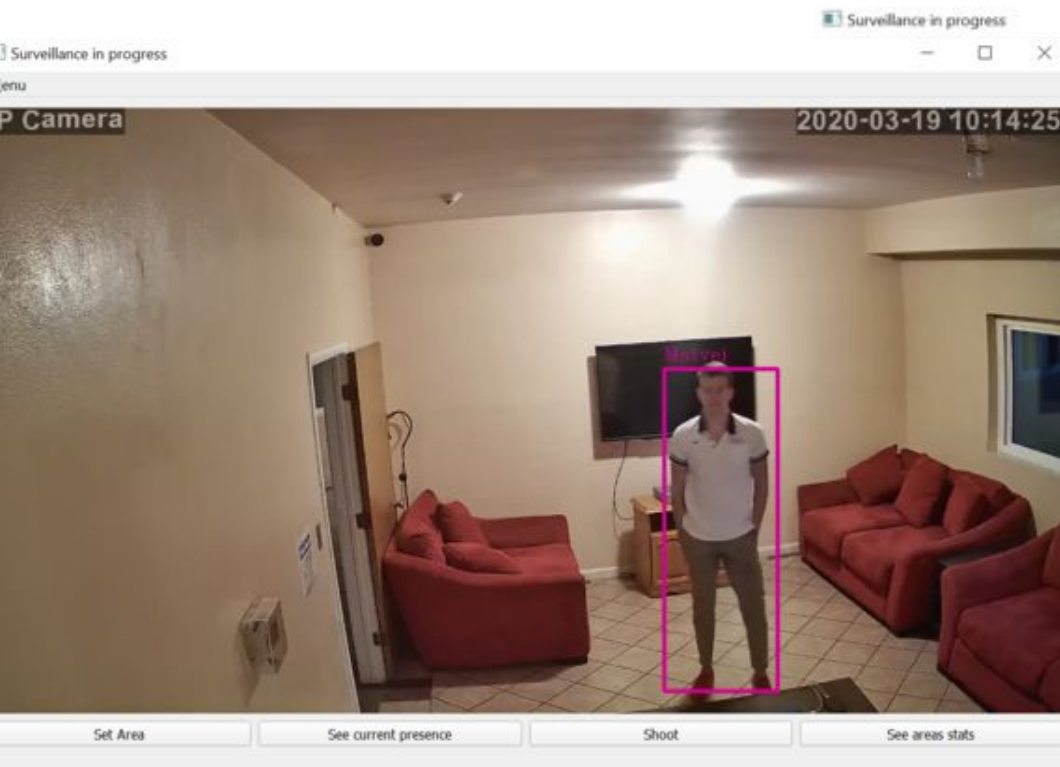
Picture 14

- The output of the system was very accurate, time statistics counted by system almost matched the reality (there was no time loss higher than 30 seconds). Time loss occurred due to instability of internet connection for IP camera, and rare inability of system to identify human objects, due to picture quality change (as in Picture 14). See the system output below:

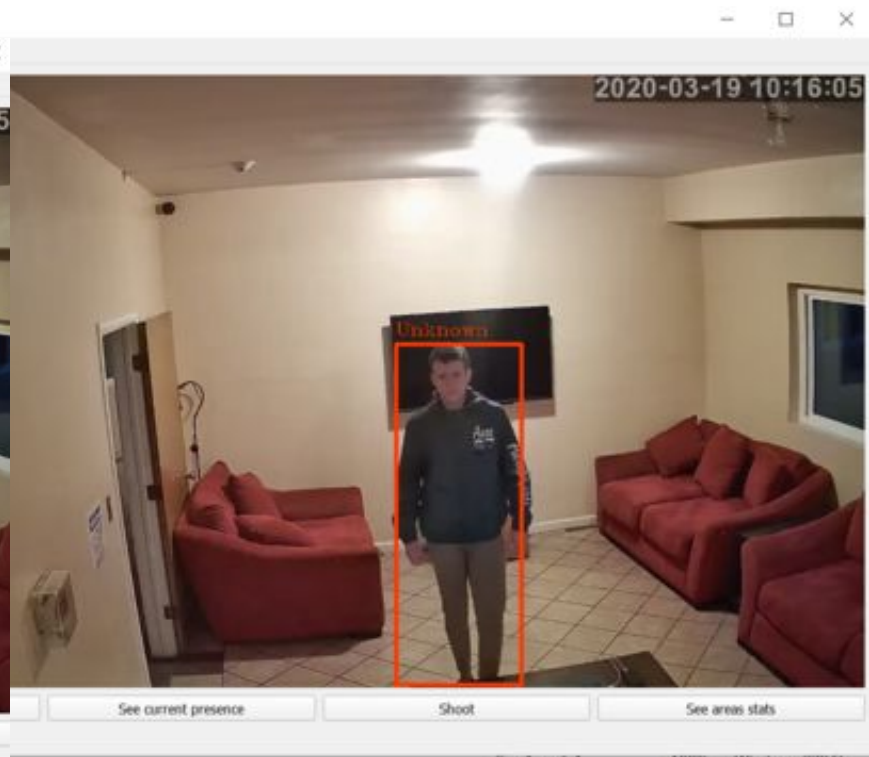


Reidentification of a human figure, with changed visual characteristics.

- In real life, human figures are changing their visual characteristics very often (for example people changing their clothes). This is a big problem for my system, because the **Reidentifier** network extracts very specific features of the person, from a picture that is used to learn. Clothes and other visual characteristics are a huge part of these features, if they are going to be changed, in most cases the network would not be able to identify a human object once again. Below is the example:
 - I trained the system to recognize me wearing a polo ((Picture 15).
 - However after I put a hoodie on the top of it, the system stopped recognizing me (Picture 16).



Picture 15

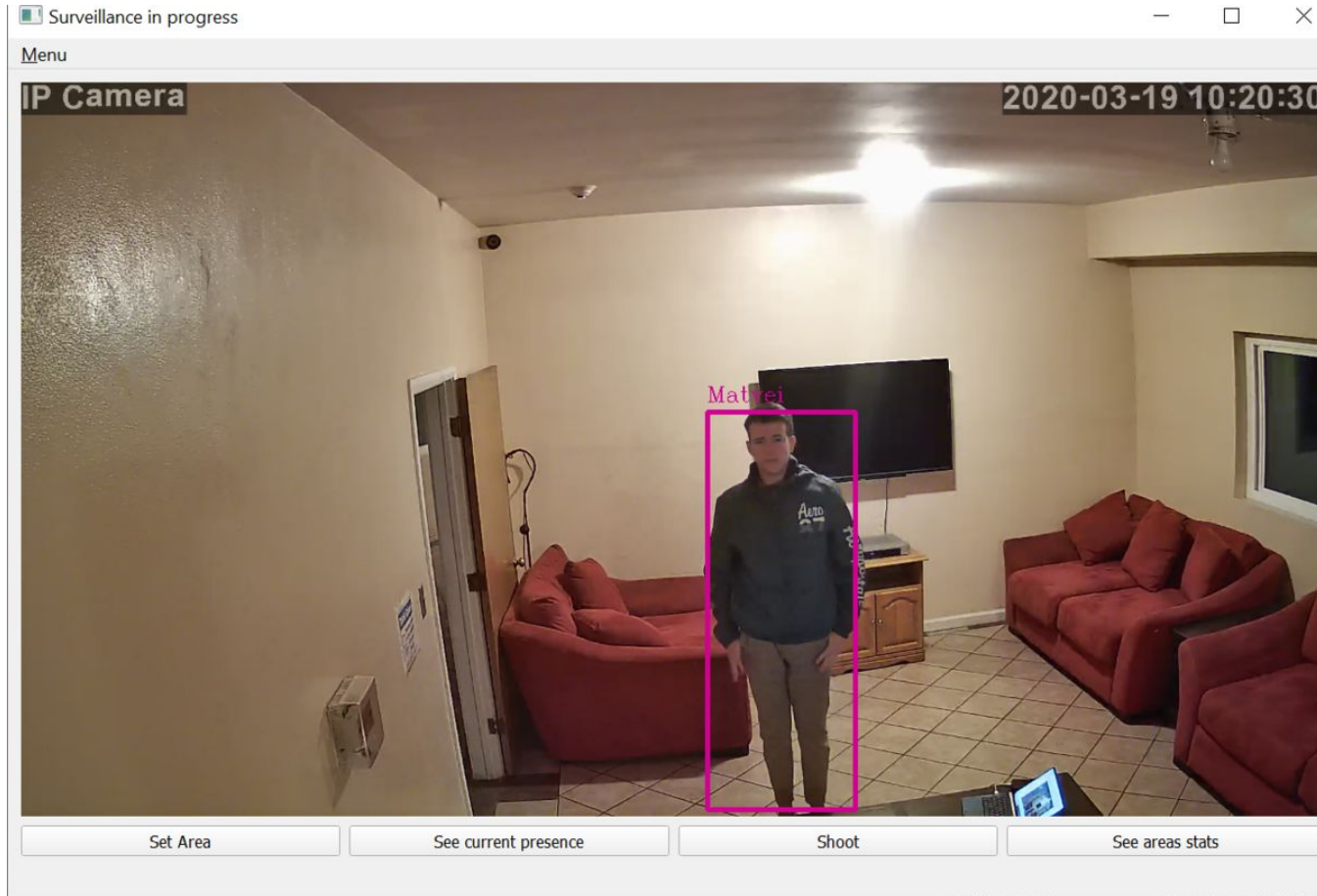


Picture 16

- The system requires a bit of a user's help, to solve this problem. Right after detecting an Unknown human object, the system will notify the user about it via Telegram Bot, so the user could manually make the human object recognizable. In this case, it took me exactly 4 seconds to make the system recognize me wearing a hoodie. See the process below:



- Now the system can recognize me wearing both hoodie and polo.



Accuracy vs Speed tradeoff



- This system has a clear tradeoff between its accuracy of people identification and the speed of its performance. The more pictures of a specific human figure user provide, the more accurate the system's predictions become (due to variability of data) however, the system's performance becomes slower. It happens due to an increase in the amount of data that the system needs to analyze every time it performs identification on a human figure. It is a user's choice on what to choose in this tradeoff. In the future, many optimization algorithms could be applied to make this tradeoff less visible for a user.

Possible improvements



- The list of potential improvements for this system is endless, here are some of them:
 - Make system be able to identify the gender, age of detected human objects, and use it for collecting statistics (Use neural networks for this).
 - Applying optimization algorithms all over the program, to increase its speed performance.
 - Make my project available and supportable for any operating system (OS).
 - Make the code more Object-Oriented and apply software engineering patterns to make it more readable and “beautiful”.
 - Develop a web-based version of this project.
 - Make design of the GUI and Telegram Bot better.

Conclusion



The successful test of the system proved that territory surveillance could be easily done with minimum human involvement and effort if we will use neural networks for that purpose. The accuracy of my software is close to the accuracy of a human. This software has the potential to increase humanity's productivity and safety in the long term. The amount of modifications and improvements that can be applied to this system is enormous, which is great. I am strongly motivated to keep developing this project, until one day it makes the lives of people better.

Bibliography



Intel OpenVINO - <https://software.intel.com/en-us/opencv-toolkit>

Keras Implementation of YOLOv3 - <https://github.com/ggwwwww/keras-yolo3>