Department of Information Engineering and Computer Science (DISI)

University of Trento, Italy

Computer Vision - AY 2020/2021

MMAnomaly module report

Students:
Davide Bulbarelli [215047]
Davide Piva [211585]

1. Introduction

The scope of this project was threefold: firstly, understand how the mmAnomaly [1] module works, then perform some code refactoring in order to improve the developer experience when configuring and running the project and, finally, test and evaluate its performances on video simulations generated by the Crowd-Simulator [2] based on Unity.
The main obstacle consisted in running the mmAnomaly module: the code base dates back to 2013 hence many library functions are deprecated nowadays, it lacks proper documentation and some code parts were affected by several bugs which compromised the final results in the first stage.

2. Related work

In the following sections is reported what has been found at the start of the project in the mmAnomaly module and in the Unity simulator.

2.1. MMAnomaly - module 2

This module takes in input a .avi file to find the anomalies on. The module operation is divided into five parts:

1. A grid of points is drawn, one every eight pixels and used as a starting point;
2. Then, the Lucas-Kanade Optical Flow [3] is computed on the scene using the previous grid and identifying the flow vectors between two frames;
3. After three frames, the displacement between the first and third frame points is computed if and only if the flow was identified for those specific points.
4. Using the Mixture of Gaussian method (where the number of gaussians is fixed to three), the background and foreground are separated for the selected points of the previous step. Only the foreground points are considered as anomalous and stored;
5. The first four steps are repeated until the video ends. If at a certain point of the video, the number of anomalous points overcomes a predefined threshold for a predefined number of frames, the anomaly is detected and kept until the number of occurrences is below the anomaly threshold for another predefined number of frames.

2.2. MMAnomaly - module 3

This module takes in input a video file to find the anomalies on. The goal is to observe the anomalies in the flow of the crowd, modeling its behavior on an energetic basis. A standard model of crowd motion is built in certain areas defined as rectangles. This behavior is taken as a reference to consider whether the energy computed in the subsequent frames is below the threshold (fewer people than expected) or above the threshold (more people than normal).

The module operation is divided into four parts:

1. Selection of the surveillance areas, identified with rectangles;
2. For each surveillance area is created a uniform grid of particles;
3. Accumulation phase: predefined observation period where particles are accumulated based on the crowd motion in order to build a benchmark behaviour model;
4. Evaluation phase: in this phase the energy of the particles is computed and compared with respect to the benchmark behaviour model. If the energy computed for a certain surveillance area overcomes a threshold, the relative rectangle will be red. Instead, if the energy computed for a certain surveillance area goes below a threshold, the relative rectangle will be blue. Otherwise, the rectangles will be green.

2.3. Crowd Simulation with Unity

The provided Crowd-Simulator [2] consists of two main part, the rendering part written in C# and the logical part written in Python. The two communicate with each other via a TCP channel.
In particular, the Python code is responsible for determining the state of the actors in the scene, including their velocity and direction.
On the other hand, the C# code is responsible for rendering the actors into the scene along with the various objects representing a city and the illumination depending on the time. Thanks to the Unity engine, it is possible to put various cameras and record the simulation frame by frame. The only user input to this simulator is a configuration file where the spawn areas and day time can be defined.
When the simulation starts, the rendering part will communicate the number of actors, their position and velocity, then the logical part will determine the next step for each of the actors along with the velocity.

3. Methodology

3.1. MMAnomaly module refactoring

The refactoring of the MMAnomaly module took some effort: some libraries have been excluded, some others lack of version in the original report and moreover the technical documentation was absent so we needed to figure out what this old code base was performing.
The first achievement was to correctly link the proper libraries and run the code with Eclipse IDE. The second was to discover what the code was performing and fix some bugs that were preventing the correct detection of the anomalies.
Finally we realized a simpler way to install the modules on the machine via bash script that is responsible for downloading the dependencies and building the source code. Also, the interaction part has been simplified thanks to the introduction of a configuration file and a single entry point that does not need to be recompiled each time as previously.

For the second module, the parameters that can be tuned in the csv config are: the video source path, the particle threshold to detect an anomalous behaviour, the number of frames for consequent anomalous behaviours before triggering the anomaly state and the number of frames to recover from it.

For the third module, the parameters that can be tuned in the csv config are: the video source path, the video's width scale, the video's height scale, the starting frame of the energy accumulation phase, the ending frame of the energy accumulation phase, the threshold for high energy, the threshold for low energy and the surveillance area rectangles (defined as a single parameter that will be parsed as an array where each entry defines a single rectangle and every value must be separated by ";". A rectangle is defined with a tuple *(Y;X;width;height)*).

3.2. Anomaly integration into Crowd Simulation with Unity

The original simulator lacks the implementation of anomaly simulation. Our main task was to understand how the rendering and logical part interacts and then integrate a system to simulate the anomalies. We considered anomalous the behavior of changing speed at a certain time *T* with a multiplier *M* of a number of actors *N* during the simulation.
With this new feature, through a XML configuration file it is possible to define the number of actors that will look anomalous, the speed multiplier and the number of steps before starting the anomaly.

4. Technical Documentation

4.1 Installation

In order to install all the dependencies required to run correctly the mmAnomaly module, a bash script (*setup.sh*) has been developed and it is located in the principal folder. To install the dependencies, the aforementioned script has to be run with root privileges through the command "*sudo ./setup.sh --install*". Note that the overall procedure may take quite a long time.

4.2 Configure and run

In order to run the mmAnomaly module, the C++ project needs to be built. In order to achieve this, the bash script *setup.sh* can be exploited. The project can be built in two ways:

- When "*sudo ./setup.sh --install*" is executed, the script will build the project automatically after having downloaded and installed all the required libraries;
- Executing the command "*./setup.sh --build*".

When the build is finished, an executable file "mmAnomalyExecutable" is generated. It can be run making explicit the module number that has to be executed (e.g., "*./mmAnomalyExecutable 2*" to run the second module). A flag "*-v*" or "*--verbose*" can be passed as the last parameter if more statistics are needed in the console output. Note that the configuration files inside the *config* folder need to be adapted before running the executable.

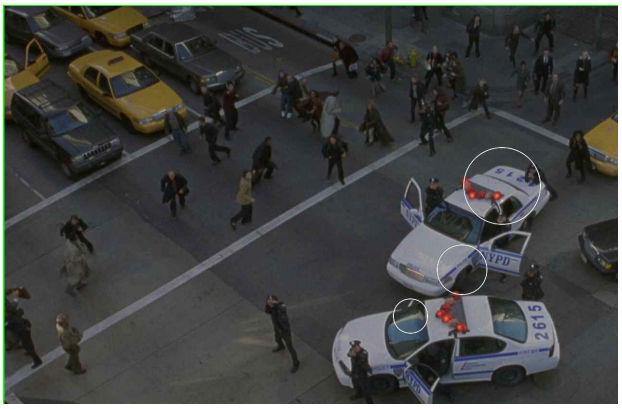
5. Results

In this chapter are shown the results obtained from both the modules using some given videos of real-life scenarios and some other videos generated in Unity with anomalies.

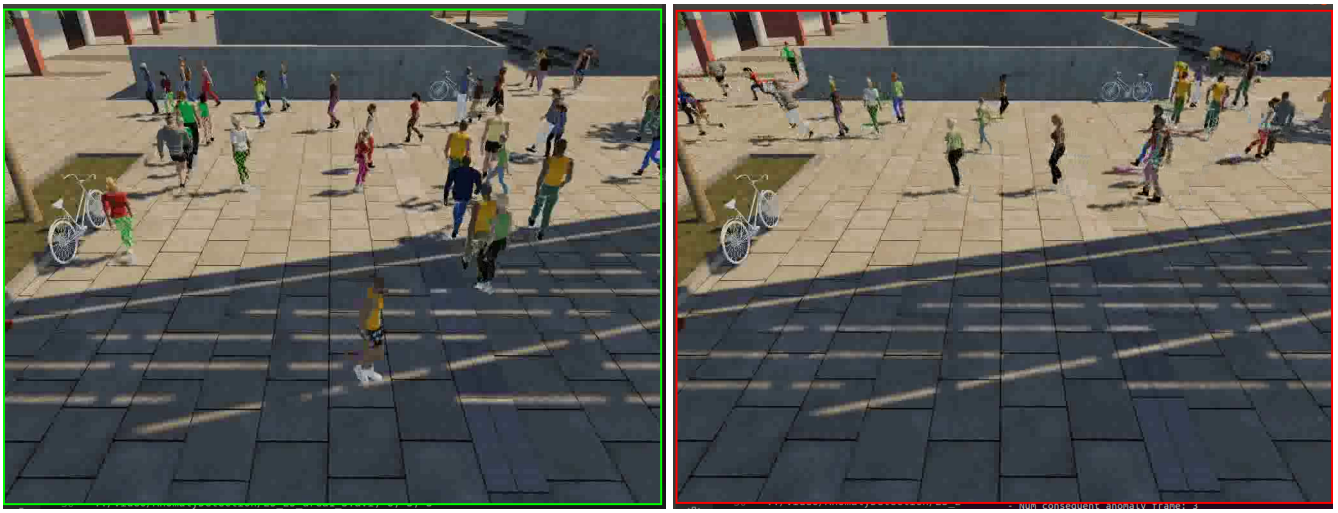
5.1 Module two results

The following screenshots will demonstrate how the second module performed both on real and simulated crowd anomalies.

The video "Crowd1.avi" [4], contains some real people walking around and then running away from the scene. With a threshold of 70 particles, 3 consecutive frames before the anomaly signal and 3 consecutive frame to go back to a normal state, the anomaly was successfully detected.



The video “police2.avi” [4], contains some real people in an urban street running away from a gun fight where police is involved. Here the anomaly was not detected: the cause may be an unsuccessful tracking by the Lucas-Kanade Optical Flow [3] since the colors are very similar, so the particles didn’t move enough.



The video “50_25_area3.avi” [4], contains 50 simulated actors walking through a direction, after some frames 25 of them start running away from the scene. With a threshold of 230 particles, 3 consecutive frames before the anomaly signal and 3 consecutive frames to go back to a normal state, the anomaly was successfully detected.

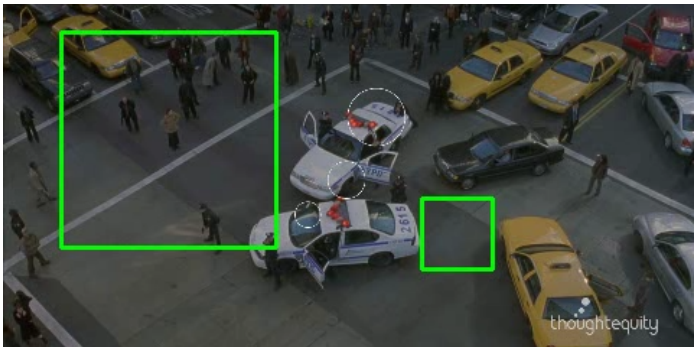
The video “100_50_area1_4.avi” [4], contains 100 simulated actors walking through a direction, after some frames 50 of them start running away from the scene. With a threshold of 100 particles, 3 consecutive frames before the anomaly signal and 3 consecutive frames to go back to a normal state, the anomaly was successfully detected.


The video “50_50_area7.avi” [4], contains 50 simulated actors walking through a direction, after some frames 50 of them start running away from the scene. The anomaly was not detected since the actors are too small with respect to the area observed by the camera.

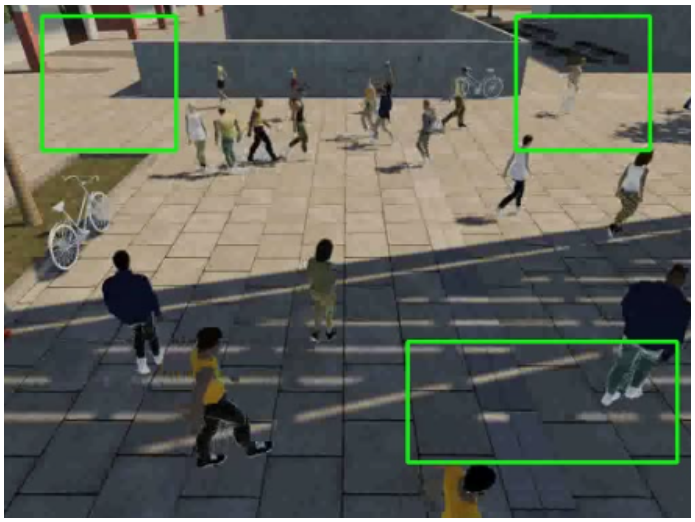

The video “50_25_area3.avi” [4], contains 50 simulated actors walking through a direction, after some frames 25 of them start running away from the scene. The anomaly was not detected since the actors are too small with respect to the area observed by the camera.

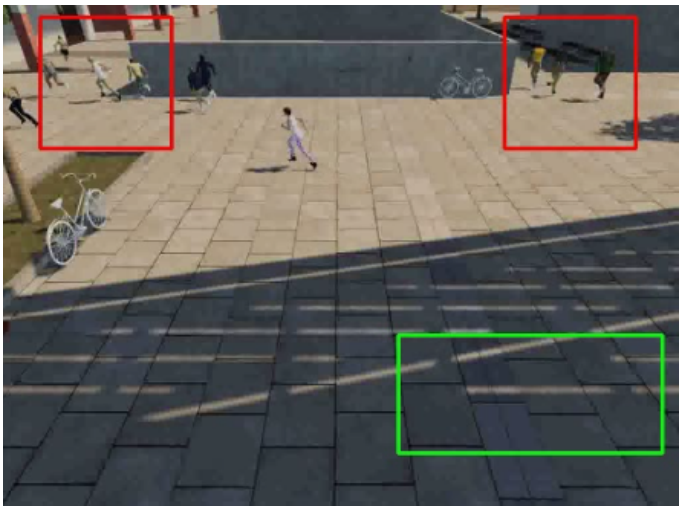
5.2 Module three results

The following screenshots will demonstrate how the third module performed both on real and simulated crowd anomalies.



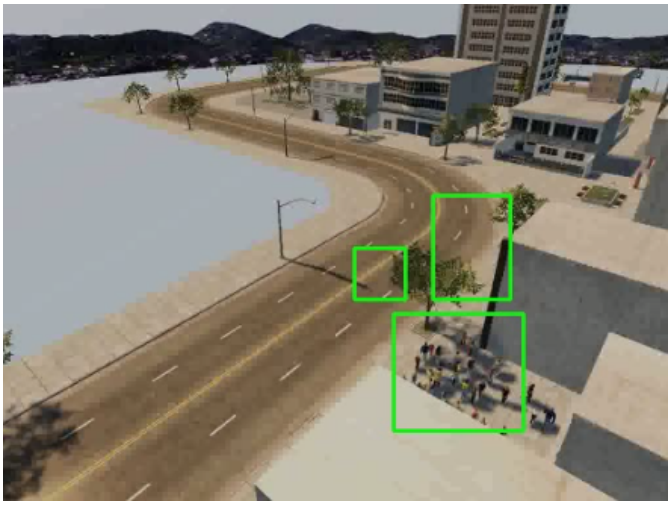
For the video “police2.avi” [4], the module is able to detect the anomaly using a *high_threshold* of 30 and the first 120 frames as accumulation phase.

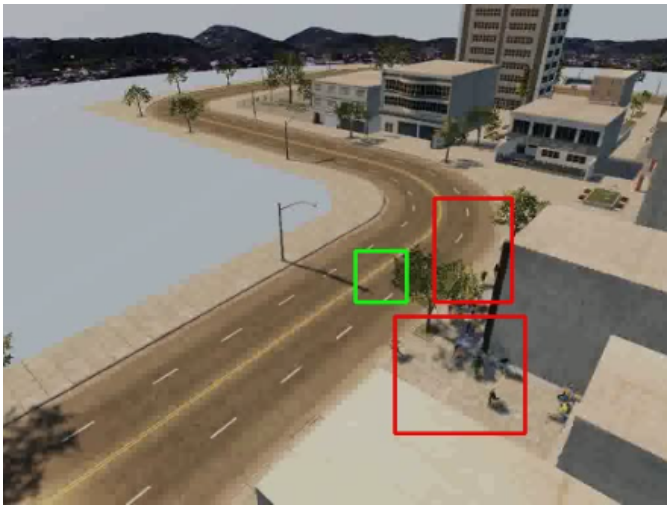




For the video “25_25_area3.avi” [4], the module is able to detect the anomaly using a *high_threshold* of 60 and the first 140 frames as accumulation phase.

For the video “movie8.avi” [4], the module is not able to detect the anomaly using a *high_threshold* of 1 and the first 140 frames as accumulation phase.





For the video “50_50_area7.avi” [4], the module is able to detect the anomaly using a *high_threshold* of 3 and the first 140 frames as accumulation phase.

6. Conclusions

6.1. Module two considerations

After having observed the results of module two, what is clear is that most of the anomaly detections depend on the camera quality and position, also on the colors present in the video. A far away and bad quality recording will not be useful to the module since it will not be able to track the movements and compute the number of moving particles used to detect anomalous behavior in the scene.

6.2. Module three considerations

After having observed the results of module three, it seems clear that it can perform quite well if the parameters are tuned properly (mostly the duration of the accumulation phase, the two thresholds and the surveillance rectangles). It is able to perform well even if the crowd is quite far away with respect to the camera but it is susceptible to small variations (e.g., shadows) when a small threshold has to be set. Due to this reason, a possible future work can be to set different thresholds for each surveillance rectangle.

6.3. Module two vs module three

Both modules have pros and cons: module two is simpler to implement and configure, also the shadows do not influence the detection too much. But, when the actors are small with respect to the observed scene, the module is unable to perform the anomaly detection properly. On the other hand, module three does not suffer from this problem since it monitors the energy of the particles in a portion of the image (surveillance rectangle), but it is more complex to configure and more sensible to variation of the scene lights and shadows.

Bibliography

1. https://github.com/dadebulba/ComputerVisionProject
2. MMLab private Crowd Simulator
3. https://docs.opencv.org/3.4.14/d4/dee/tutorial_optical_flow.html
4. Crowd Anomaly Videos Directory