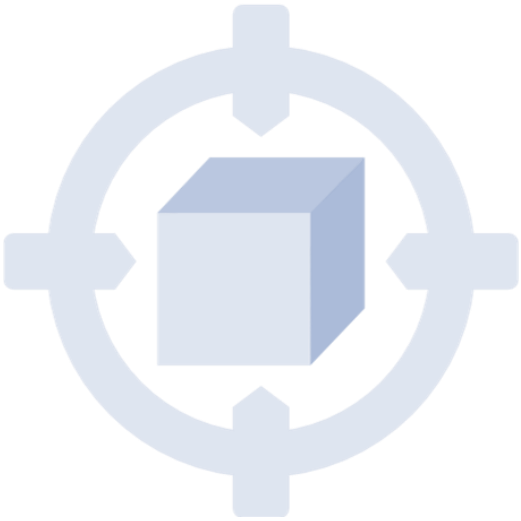
Collision Detection

University of Minho

Master in Informatics Engineering



Computer Graphics

Image Processing and Computer Vision

February 2016

Bruno Miguel da Silva Barbosa	67646
Carlos Manuel Magalhães Gonçalves	67650
Carlos Rafael Cruz Antunes	67711

Table of Contents

Table of Images

Introduction

Collision detection is a computational problem where the objective is to detect when two or more objects intersect. This topic is very often related with the areas of videogames and physical simulations as well. It also has a big importance on robotics because they need to know when they reached an object or even change direction when they face a wall. This technology is present in our lives too. A lot of people use their cars to go for work and there are some situations that is difficult to park. Nowadays cars come with an integrated rear gear camera that helps on the parking maneuver allowing people to see how far the car is to the obstacle and warning them when they are too close to it.

The objective of this work assignment was to detect general movement in a video, with one or more objects. This movement consists in travels from one side to the other of the screen, collision with the camera and approaching or receding objects. This is a real time application, so we will study the algorithm performance by measuring the time spent on each frame. The validation process is another essential point of this work assignment. We do not want to just develop an application: we want a robust application that can detect collision when it actually happened.

For this task, a script was developed in MatLab 2015 which reads a video, detects its background and starts detecting and tracking an object. We reused an already existent algorithm and we adapted it according to our preferences. Finally it studies its route do determine which type of movement described before is the object performing. The results are then printed on the console output, in real time.

Object Tracking

To track the various possible objects in a given video it was implemented a previously created algorithm. This motion-based multiple object tracking algorithm is available online [1]. As the name suggests this algorithm is capable of detecting and tracking multiple objects. It displays a mask in which we can see the objects detected, as well as the respective bounding box for each object. The centroid of each object is also calculated, but not shown in the original algorithm (the centroid is the arithmetic mean position of all the points in the shape). Later a function from our authoring was added and the centroid is also shown.

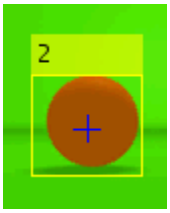


FIGURE 1 - OBJECT WITH BOUNDING BOX AND CENTROID

FIGURE 1 represents a ball moving towards the camera, the bonding box and the calculated centroid in one particular frame of the video. FIGURE 2 represents the same scenario, but with background noise. We can see by these two examples that the algorithm is precise in most tested cases.

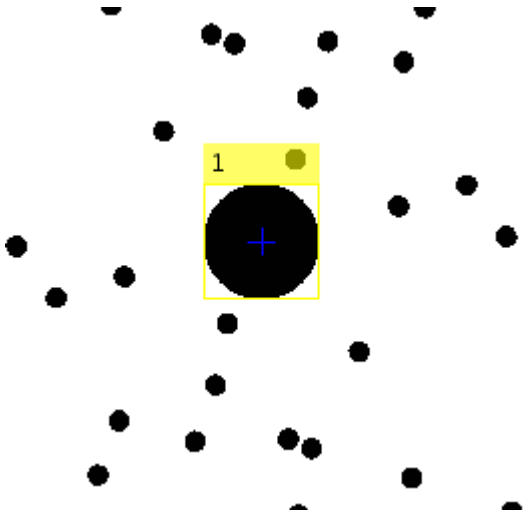


FIGURE 2 – OBJECT DETECTION WITH NOISE

With this stage concluded we were able to use the centroid and bounding box information to detect the direction of the object's movement, and consequently detect collisions.

Collision Detection

One of the objectives of this work assignment was to track the object and categorize its trajectory. Once the object is recognized, we can now analyze the way it moves. Our implementation considers that moving forward implies that the bounding box has grown. Otherwise, if the object is receding it means that the bounding box had decreased. The translations are detected through the last possible movements. So, if the bounding box is moving to the left or to the right, we say that occurred a translation, respectively.

We achieved collision detection be studying the centroid and bounding box of a certain object. For example, if the centroid is always in the center of the video, and the bounding box is progressively increasing, that means that the object is approaching, and if said bounding box occupies most of the screen we can conclude that there was a collision. By the other hand, it doesn't make any sense to detect collision when the object is receding.

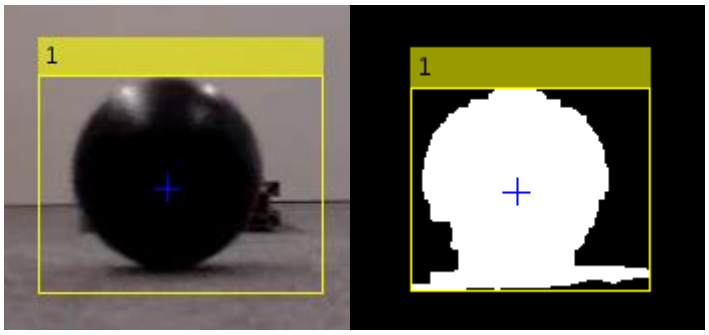


FIGURE 3 - DETECTED OBJECT AND ITS MASK

To tune the script in order to satisfy the most part of the test videos and avoid false positives, the script does not perform the calculations explained before every frame. Instead, it calculates the average centroid and bounding box throughout four frames, and only then classifies its movement.

The output for a uniformly approaching object throughout the length of the video is:

```
Object Approaching
Object Approaching
…
Object Approaching
Collision Detected! *beep sound*
```

Each new line is printed every four frames, as explained before, which gives us the real time movement detection. Another output for a translating and then receding object is:

```
Movement to the right
Movement to the right
Movement to the right
Object Receding
Object Receding
…
```

Unsuccessful Cases

The developed algorithm has two main problems. In some case test videos the algorithm was unable to identify any kind of movement correctly. In other cases with a uniformly approaching object an "Object Receding" message can be found on the output, along with the "Object Approaching" messages.

These problems are the result of a very low differentiation between the object and the video background or the reduced number of frames the videos, which does not allow a clear object detection and tracking.

For example, **Error! Reference source not found.** represents the first problem, and FIGURE 5 the second problem (lack of frames did not allow the algorithm to detect a collision).

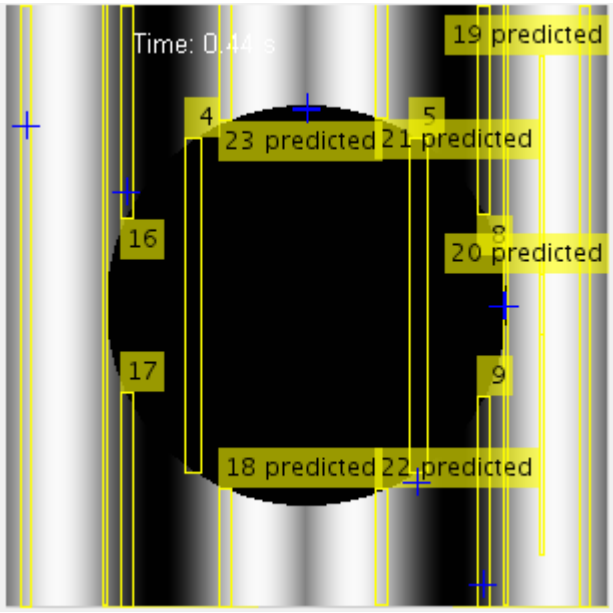


FIGURE 4 - EXAMPLE OF AN UNSUCCESSFUL CASE

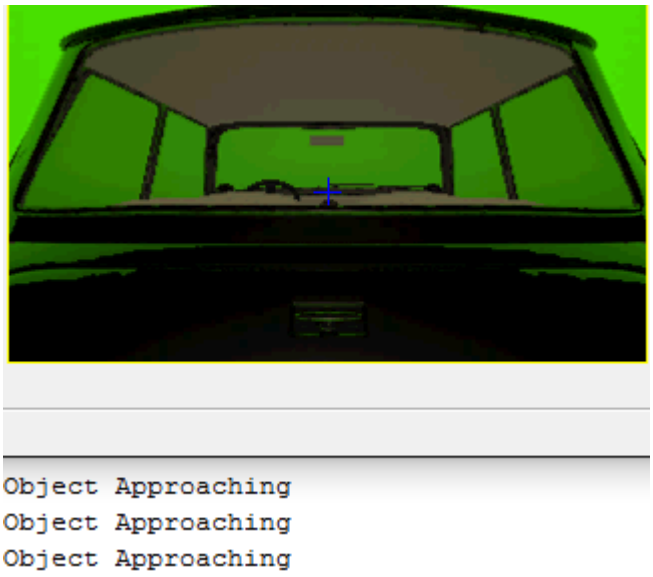


FIGURE 5 - COLLISION NOT DETECTED

Performance and Validation

In this section we evaluate the performance my measuring the time spent on each frame. The performance measures were made on a MSI GE60 2QE 1026XPT Apache Pro laptop with an i7-4720HQ processor (more information on Annex – Hardware Characterization). The next step we performed was the validation process that englobes a test for the robustness of the used algorithm by identifying the cases where we have a:

- True positive – When there is a collision and our program detects it.
- True negative – When there isn't a collision and our program knows that wasn't a collision.
- False positive – When the program detects a collision when, in fact, there wasn't any.
- False negative – When occurs a collision and the application isn't capable of classifying it.

VIDEO	ELAPSED TIME (S)	FRAMES	FPS	TIME/FRAME	COLISION	PROGRAM
1_BOLA	3,06	199	65	0,0154	Yes	Yes
2_BOLAS	3,48	199	57	0,0175	No	No
APPROACHING_25MS	1,87	80	43	0,0234	Yes	Yes
APPROACHING_25MS_TEXTURE	1,81	80	44	0,0226	Yes	No
APPROACHING_50MS	3,14	151	48	0,0208	Yes	Yes
APPROACHING_50MS_TEXTURE	3,07	151	49	0,0203	Yes	No
APPROACHING_CIRCLE_25MS	1,86	81	44	0,0230	Yes	No
APPROACHING_CIRCLE_50MS	3,16	152	48	0,0208	Yes	Yes
APPROACHING_FLOW_FIELD	4,58	177	39	0,0259	No	No
APPROACHING_SCATTERED	3,71	177	48	0,0210	No	No
APPROACHING_YESPLE	3,62	177	49	0,0205	No	No
CAR	1,66	90	54	0,0184	Yes	No
CAR2	1,67	90	54	0,0186	Yes	No
RECEDING_25MS	1,91	79	41	0,0242	No	No
RECEDING_50MS	3,37	150	45	0,0225	No	No
S_A_R	1,46	59	40	0,0247	Yes	Yes
S_N	1,45	59	41	0,0246	Yes	Yes
TRANSLATING_25MS	1,93	84	44	0,0230	No	No
TRANSLATING_25MS_0,5METERS	1,16	41	35	0,0283	No	No
TRANSLATING_50MS	3,54	170	48	0,0208	No	No
TRANSLATING_50MS_0,5METERS	1,94	84	43	0,0231	No	No
TRANSLATING_APPROACHING_25MS	2,63	116	44	0,0227	Yes	Yes
TRANSLATING_APPROACHING_50MS	4,77	223	47	0,0214	Yes	Yes
TRANSLATING_RECEDING_25MS	2,47	116	47	0,0213	No	No
TRANSLATING_RECEDING_50MS	4,25	223	52	0,0191	No	No
TRATADO_1	1,92	36	19	0,0533	No	No
UNIFORMLY_APPROACHING	3,65	150	41	0,0243	Yes	Yes

TABLE 1 – COLLISION DETECTION RESULTS

	VIDEOS WITH COLISION	VIDEOS WITHOUT COLISION	TOTAL
COLLISION DETECTED	9	0	9
COLLISION NOT DETECTED	5	13	18
TOTAL	14	13	27

TABLE 2 – COLLISION DETECTION RESULTS SUMMARY

Table 2 synthetizes Table 1 allowing us to understand more easily the results. Though, we can see that there are 5 videos where the program detects collision when there wasn't. The explanation was referred in the chapter related with the Unsuccessful_Cases.

Conclusions and Future Work

From this entire development process we have concluded that the developed algorithm works efficiently and accurately for the most cases tested, giving real time information on objects contained in the video. Although there are some known problems explained in Unsuccessful Cases. Some of this cases are due to the need learning frames, which increased the problem of low frames. In other circumstances, the video's quality didn't help either due to the background noise or even due to the poor light conditions.

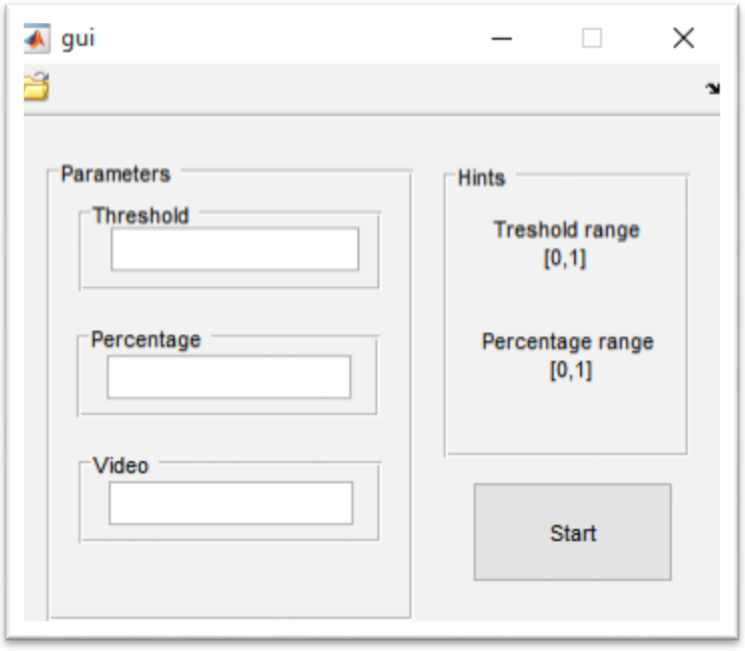
The results obtained could be better if the algorithm were focused in one particular case, or have one common background. This way the algorithm could be tuned appropriately. Another improvement could be to refine the collision detection in order to reduce the number of frames required to achieve an acceptable result with a low resolution video input.

As future work we would like to suggest the continuity of expansion of this work once we were a little bit limited by time and by other subject's commitments. Basically we think that would be a good idea to export this project to a portable device, like a smartphone, and experiment it on a real-world situation. Still, the algorithm needs to be tuned properly in order to give response to the adversity of possible scenarios. Our group is fully constituted by students that are learning concepts of parallel and distributed computing, so it would be a great opportunity to explore a parallel implementation of this program.

Instructions' Manual

In this section we present a few steps that help anyone who wants to try our application by itself. The user must have installed ®MatLab 2015, otherwise we cannot guarantee that the program will run on other versions. If you don't have ®MatLab 2015 installed we recommend to visit the website in order to know the system requirements.

1. Download the file to the desired location.
2. Open ®MatLab 2015 and navigate to the extraction folder.
3. Navigate to the *collision_detector* folder
4. Run the *gui.m* file.
5. It will open a window just like the figure below. There are three parameters:
    a. The threshold receives its level in a range of [0,1];
    b. The percentage receives the percentage of the screen that one object must occupy so that it is considered a collision. The value should be in range of [0,1];
    c. The video receives the path to the video to read. The user must write something like "*videos/filename.avi*".
6. Once this field have been filled, click start.



References

[1] MathWorks, "Motion-Based Multiple Object Tracking," [Online]. Available: http://www.mathworks.com/help/vision/examples/motion-based-multiple-object-tracking.html.

[2] MathWorks, "Computer Vision," [Online]. Available: http://www.mathworks.com/products/computer-vision/.

Annex – Hardware Characterization

- Manufacturer: MSI
- Model: GE60 2QE 1026XPT (Apache Pro)
- Motherboard: MSI MS-16GF
- CPU chip
    - Manufacturer: Intel
    - Model: Core i7-4720HQ
    - Reference: [1]
    - Number of Cores: 4
    - Number of Threads: 8
    - Base Frequency: 2.6 GHz
    - Maximum Turbo Frequency: 3.6 GHz
    - Instruction Set: 64-bit
    - Architecture: Haswell
    - 16 DP FLOPs/cycle
    - Theoretical Peak Performance (DP): 166 GFLOPS
    - Benchmark Peak Performance (DP): 128 GFLOPS
- Cache
    - L1 Data Cache Size: 4 x 32 KB
    - L1 Instructions Cache Size: 4 x 32 KB
    - L2 Unified Cache Size: 4 x 256 KB
    - L3 Unified Cache Size: 6144 KB
- RAM
    - Manufacturer: Kingston
    - Size: 8 GB
    - DDR3L
    - 1600 MHz
    - Latency: 11 clock cycles
    - Maximum Memory Bandwidth: 13 GB/s