Hes·so // VALAIS WALLIS
Haute Ecole d'Ingénierie
Hochschule für Ingenieurwissenschaften

Hes·so

Degree Programme
Systems Engineering

Major Infotronics

Bachelor's thesis
Diploma 2023

Rithner Aurélien

Real-time line crossing detection

Professor
Carrino Francesco

Expert
Donzé Célien

Submission date of the report
25.08.2023

HES-SO Valais-Wallis • rue de l'Industrie 23 • 1950 Sion
+41 58 606 85 23 • hei@hevs.ch • www.hevs.ch/hei

swissuniversities

EQUAL-SALARY CERTIFIED

Données du travail de diplôme
Aufgabenstellung der Bachelorarbeit

x

Filière / Studiengang **SYND**	Année académique / *Studienjahr* **2022-23**	No TB / *Nr. BA* **IT/2023/87**
Mandant / *Auftraggeber* ☒ HES—SO Valais ☐ Industrie ☐ Etablissement partenaire *Partnerinstitution*	Etudiant / *Student* **Aurélien Rithner** Professeur / *Dozent* **Francesco Carrino**	Lieu d'exécution / *Ausführungsort* ☒ **HES—SO Valais** ☐ Industrie ☐ Etablissement partenaire *Partnerinstitution*
Travail confidentiel / *vertrauliche Arbeit* ☐ oui / ja ☒ **non / nein**	Expert / *Experte* (données complètes) **Célien Donzé** Haute Ecole Arc – Ingénierie, Espace de l'Europe 11, 2000 Neuchâtel	

Titre / *Titel*

Real-time Line Crossing Detection and 2D/3D mapping [using a low-cost camera drone]

Description / *Beschreibung*

Ce travail de diplôme vise à mettre en place un système pour détecter automatiquement, à l'aide de caméras, le dépassement d'une ligne de marquage au sol par un véhicule. Le système doit fonctionner en temps réel.

Le cas d'application visé est la *Summer School I* de la HEI-VS dans laquelle des petits véhicules, dont la forme peut varier, doivent compléter un parcours sans dépasser certaines limites tracées au sol pour éviter d'encourir des pénalités.

Comme deuxième proof-of-concept (POC), ce projet investiguera également les possibilités et les contraintes offertes par l'utilisation d'un drone low-cost muni de camera (plutôt qu'une installation fixe).

Objectifs / *Ziele*

- Analyse de l'état de l'art des algorithmes pour la détection de dépassement
- Réalisation d'un POC avec des caméras fixes.
- Evaluation des performances
- *Objectif optionnel* : analyse-réalisation-évaluation POC avec drone « low-cost ».

Taches / *Aufgaben*

1. Analyse
   a. Spécifications du problème : contraintes et possibilités en termes d'installations, sécurité, bande passante, puissance de calcul, etc.
   b. Bref état de l'art des algorithmes pour la détection des lignes de marquage au sol et de dépassement.
   c. Prise en main des caméras [et du drone]
   d. Définition des métriques d'évaluation des performances
2. Conception
   a. Définition de l'architecture du système (hardware)
   b. Définition de l'architecture du système (software)
   c. Définition de 1-2 prototypes intermédiaires vers le POC final
   d.
3. Développement et évaluation
   a. Développement itératif (développement – test) vers la réalisation d'un proof of concept fonctionnel
   b. Evaluation du système en termes des métriques définie pendant la phase d'analyse.

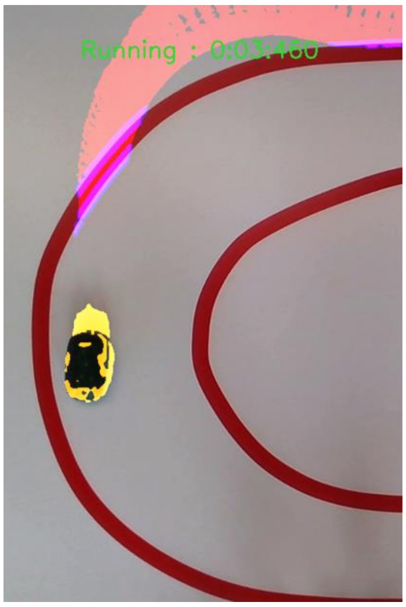
Rapport reçu le / *Schlussbericht erhalten am* ……….………. Visa du secrétariat / *Visum des Sekretariats* …………..

Signature ou visa / *Unterschrift oder Visum* Responsable de l'orientation / *Leiter der Vertiefungsrichtung:* [illegible] .... [1] Etudiant / Student: [illegible]	Délais / *Termine* Attribution du thème / *Ausgabe des Auftrags:* **15.05.2023** Présentation intermédiaire / *Zwischenpräsentation*: **19 - 20.06.2023** Remise du rapport final / *Abgabe des Schlussberichts:* **25.08.2023, 12:00** Expositions / *Ausstellungen der Diplomarbeiten:* **25.08.2023** – HEI **28.08.2023** – Monthey **31.08.2023** – Visp Défense orale / *Mündliche Verfechtung:* **Semaine/Woche 36 (04-07.09.2023)**

[1] *Par sa signature, l'étudiant-e s'engage à respecter strictement la directive DI.1.2.02.07 liée au travail de diplôme.*
Durch seine Unterschrift verpflichtet sich der/die Student/in, sich an die Richtlinie DI.1.2.02.07 der Diplomarbeit zu halten.

Hes·so VALAIS WALLIS

Haute Ecole d'Ingénierie
Hochschule für Ingenieurwissenschaften



Real-time line crossing detection

Graduate Rithner Aurélien

Bachelor's Thesis
| 2023 |

Degree programme
Systems Engineering

Field of application
Infotronics

Supervising professor
Prof. Carrino Francesco
Francesco.carrino@hevs.ch

Objectives

The aim of this bachelor's project is to provide a solution for automatically detecting, using cameras and in real-time, when a vehicle crosses the lines of a racetrack markings during the Summer School 1 module.

Methods | Experiences | Results

First, an analysis of the problem's specifications, the different hardware choices available and the different techniques used in this computer vision field was made.

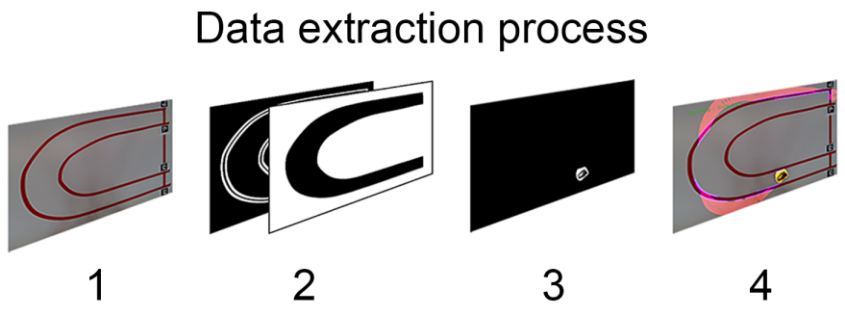
Then, using Python and OpenCV, a prototype was developed with two major parts. The back-end assuring image acquisition and processing as well as data extraction. The front-end is a Django based web server that allows to store information in a database and display them on a web page.

A mock-up was built to test the system under similar conditions to those of the actual race.

A machine learning based approach was explored as well. It required to generate virtual images of the race for the data set used to train the CNN model.

The prototype achieved timing and crossing detection using traditional methods.

The front-end web page allows users to monitor the race information and watch the best times saved on a scoreboard.



Initially, a median frame computes the static image portion (1). Two masks, generated by the Canny Edge detection algorithm, identify track limits (2). The following step involves detecting moving objects through background subtraction (3). Combining these masks highlights off-track areas (4).

swissuniversities EQUAL-SALARY CERTIFIED Valais excellence

Abstract

This report documents a project carried out as part of the "Summer School I" program within the Industrial Systems curriculum at HES-SO Sion. Over a three-week period, students are tasked with building remote-controlled vehicles for a race competition.

The objective is to develop a system that can automatically detect when a vehicle crosses track marking in real-time using cameras.

First, an analysis of the problem's specifications, the different hardware choices available and the different techniques used in this computer vision field was made.

Then, using Python and OpenCV, a prototype was developed with two major parts. The back end assuring image acquisition and processing as well as data extraction. The front end is a Django based web server that allows to store information in a database and display them on a web page.

Evaluations shows that the prototype achieved timing and crossing detection using traditional methods, while the front-end web page allows users to monitor the race information and watch the best times saved on a scoreboard.

A final proof-of-concept was designed and will be put to test at this year's summer school race. Therefore, a mock-up was built to test the system under similar conditions to those of the actual race.

Additionally, a machine learning based approach was explored as well. It required to generate virtual images of the race for the data set used to train the CNN model using Unity and Blender. With encouraging initial results, this approach was put on the side for time reasons.

Contents

1 Introduction *7*

1.1 Context of the project **7**

1.2 Aim of the project **7**

1.3 Goals **8**

1.4 Report structure **8**

2 Analysis *9*

2.1 Problem specification **9**

2.1.1 Project’s needs 9
2.1.2 Installation constraints 9
2.1.3 Time constraints 10

2.2 State of the art **11**

2.2.1 Introduction to line crossing detection 11
2.2.2 Image acquisition and pre-processing 11
2.2.3 Edge detection algorithms 11
2.2.4 Tracking and object detection 12
2.2.5 Challenges and limitations 13
2.2.6 Evaluation metrics and datasets 14
2.2.7 Conclusions and future directions 15

2.3 Hands on **16**

2.3.1 OpenCV using Python 16
2.3.2 Setting up the developing environment 16
2.3.3 Test program 17
2.3.4 Material used 17
2.3.5 Line detection 19
2.3.6 Edge detection 20
2.3.7 Contour detection 22
2.3.9 Basic object tracking 23
2.3.10 Particles filter tracking 24
2.3.11 Blob detection 25
2.3.12 Camera internal calibration 26
2.3.13 Camera external calibration 28

2.4 Camera **30**

2.4.1 Criteria 30
2.4.2 FOV & Height relation 31
2.4.3 Recapitulative table 32

2.5 Drone **33**

2.5.1 Concept 33
2.5.2 Pros and cons of using a drone 33
2.5.3 Options available 34
2.5.4 Conclusion 36

2.6 Evaluations metrics **36**

3 Conception *37*

3.1 Goal **37**

3.2 Actors **37**

3.3 Separation **38**

3.4 Parts role **39**
3.4.1 Front end 39
3.4.2 Back end 39

3.5 Data exchange **40**

3.6 Mock-up **41**
3.6.1 The need for a mock-up 41
3.6.2 Frame and camera support 41
3.6.3 Exploded view and parts description 42
3.6.4 Overview 43
3.6.5 Test vehicle 44

4 The prototype *45*

4.1 Prototype concept **45**

4.2 Back-end **45**
4.2.1 Creating a median frame 46
4.2.2 Defining the limits mask 47
4.2.3 Defining the track mask 49
4.2.4 Moving object detection 50
4.2.5 Line crossing detection 51
4.2.6 Race timing 52
4.2.7 Detecting start & finish line 54
4.2.8 State machine 55

4.3 Front-end **56**
4.3.1 Purpose 56
4.3.2 Tools choice 56
4.3.3 Web page 57
4.3.4 Video feed 57
4.3.5 Run data 57
4.3.6 Scoreboard 58
4.3.7 Back-end modifications 59
4.3.8 Two way communication 59

4.4 Results **60**

5 Final proof of concept *61*

5.1 Its goal **61**

5.2 Concept & installation **61**

5.3 Principal differences with the prototype **61**
5.3.1 Different web pages 61
5.3.2 Web page control 61
5.3.3 Tracking of the vehicle 61

5.4 Limitations **62**

5.5 Testing phase and results **62**

6 Machine Learning *63*

6.1 Context **63**

6.2 Dataset creation **63**
6.2.1 Setting up the virtual environment in unity 63
6.2.2 Random placing of the car 63
6.2.3 Automatic image classification 63
6.2.4 Convex mesh challenge 64

6.3 The training **66**
6.3.1 Kaggle 66

6.3.2 Model architecture ... 66
6.3.3 Iterations ... 67

6.4 Results ... 68
6.4.1 Learning Curves ... 68
6.4.2 Evaluations ... 69
6.4.3 Conclusion on using machine learning in this project ... 70

7 Evaluations ... 71

7.1 Goals & results ... 71

7.2 Tasks & planning ... 72

8 Durability ... 74

8.1 Introduction ... 74

8.2 Innovation ... 74

8.3 Fairness & Equity ... 74

8.4 Conclusion ... 74

9 Conclusion ... 75

10 References ... 76

11 Appendices ... 77

11.1 Project specifications ... 77

11.2 User Interface ... 80
11.2.1 Console output ... 80
11.2.2 Console & OpenCV window ... 80
11.2.3 OpenCV window only ... 80
11.2.4 Graphical UI ... 80
11.2.5 Front end for display & back end for input ... 80
11.2.6 Front end for display + input & back end for processing ... 81

11.3 Race track drawings ... 82

11.4 Cameras specifications ... 83

11.5 Cameras placement height estimations ... 88

11.6 Particles filter Tracking tests ... 90

11.7 Mock-up parts details ... 93
11.7.1 Wooden Frame ... 93
11.7.2 Handles ... 93
11.7.3 Base ... 93
11.7.4 Pole ... 94
11.7.5 Tube-pole connector ... 95
11.7.6 End cap ... 95

Figures

Figure 1 : Picture of the 2022 SS1 race [1]........ 7
Figure 2: Previous vehicles built in 2020 [3] 9
Figure 3 : Concept of CNN [5]........ 13
Figure 4: Confusion matrix description [6] 14
Figure 5 : PyCharm's logo [10]........ 16
Figure 6 : Première image de test........ 19
Figure 7 : Sobel edge detection on thread test image 20
Figure 8 : Canny edge detection on thread test image 20
Figure 9 : Canny edge detection on the circuit's virtual image with better parameters........ 21
Figure 10 : Contours detection after Canny edge detection........ 22
Figure 11 : Particles filter tracking test 24
Figure 12: Setup of a blob detector in OpenCV........ 25
Figure 13: Calibration plate with detected chessboard 26
Figure 14: YAML file containing calibration parameters........ 26
Figure 15: Test image before internal calibration........ 27
Figure 16: Test image after internal calibration........ 27
Figure 17: Perspective transformation example 28
Figure 18: Warped image........ 29
Figure 19: Result image with perspective correction applied........ 29
Figure 20: Extract from Intel D415's datasheet........ 31
Figure 21: Intel D415 placement for full track view 31
Figure 22: Representation of the difference in P.O.V between a camera and drone setup.. 33
Figure 23: Dji Tello Drone [14] 34
Figure 24: Parrot Anafi Ai Drone [15] 35
Figure 25: OpenCV drone diagram [16] 35
Figure 26: Evaluation metrics defined for this project 36
Figure 27: Decomposition in sub- tasks 37
Figure 28: Actors and their respective tasks........ 37
Figure 29: Actors and main system components interactions........ 38
Figure 30: Minimalistic components architecture........ 38
Figure 31: Sub-task repartition with blue for back-end and green for front-end 39
Figure 32: Content of the webpage with the nice-to-have ones in green 39
Figure 33: Representation of the data exchanged between the front and back end 40
Figure 34 : Printed board with racetrack........ 41
Figure 35 : Virtual overview of the mock-up 43
Figure 36: Overview of the mock-up built 43
Figure 37: Revell Mini Car........ 44
Figure 38: Activity diagram for the back end 45
Figure 39: compute_median_frame function 46
Figure 40: Process to create masks – Part 1........ 47
Figure 41: Process to create masks - Part 2 47
Figure 42: Binarization values adjustments window 47
Figure 43: Example of a bad values for binarization........ 48
Figure 44: JSON file with saved values........ 48
Figure 45: Segmentation of the test track........ 49
Figure 46: Segmentation of the actual track........ 49
Figure 47 : Detection of moving objects, the two cars in this case........ 50
Figure 48 : Limits crossing highlighted in red 51
Figure 49 : Time indication displayed in top left and........ 52
Figure 50: Timekeeper class UML diagram........ 53
Figure 51: Finish line marked with ArUco markers 54
Figure 52: RaceMgmt's class UML diagram........ 55

Figure 53: RaceMgmt’s state machine UML diagram.. 55
Figure 54: Prototype web page displaying race data.. 57
Figure 55: Database architecture.. 58
Figure 56: Extract from the API code 58
Figure 57: Flask implementation in the back end 59
Figure 58: Result frame with the crossings detected 60
Figure 59: Inside and outside image generated... 63
Figure 60: Convex VS Complex Mesh colliders [17]... 64
Figure 61: The "outside zone" object decomposed in Blender.. 65
Figure 62: Colliders highlighted in Unity.. 65
Figure 63: Model architecture definition using Keras.. 66
Figure 64: Optimizer settings 67
Figure 65: Initial planning... 72
Figure 66: Actual planning 73
Figure 67: Technical specifications for the Intel D145 83
Figure 68: Page 1 of the technical specifications for the RPI WWCAM 84
Figure 69: Page 2 of the technical specifications for the RPI WWCAM 85
Figure 70: Technical specifications for the Axis M2025-LE 86
Figure 71: Technical specifications for the DLink DCS-8627LH 87
Figure 72 : Height estimation for the Intel D415 88
Figure 73 : Height estimation for the RPI WWCAM.. 88
Figure 74 : Height estimation for the DLINK DCS-8627LH 89
Figure 75 : At first particles are distributed everywhere randomly...................................... 90
Figure 76 : Quickly they focus on the green object. In this example there are two identic cars and with the parameters used they followed only one 90
Figure 77 : With this algorithm the tracking continues after the bridge................................ 91
Figure 78 : During the 180 degree turn some of the particles drift away but the tracking continues 91
Figure 79 : Tracking all the way until the end 92
Figure 80 : 3D view of the frame 93
Figure 81 : 3D view of a handle... 93
Figure 82 : 3D exploded view of the base parts.. 94
Figure 83 : 3D view of the pole.. 94
Figure 84 : 3D view of the pole-tube connector... 95
Figure 85 : 3D view of the end cap... 95

1 INTRODUCTION

1.1 CONTEXT OF THE PROJECT

As part of the "Summer School I" of the Industrial Systems cursus at HES-SO Sion, students have three weeks to build a small remote-controlled vehicle. At the end of the project, the different teams compete in a race. The circuit is marked out on the ground using a wire and includes turns and a bridge. Each run will be timed, and the fastest team will win. Time penalties are applied if a vehicle crosses the boundaries of the circuit during a run.

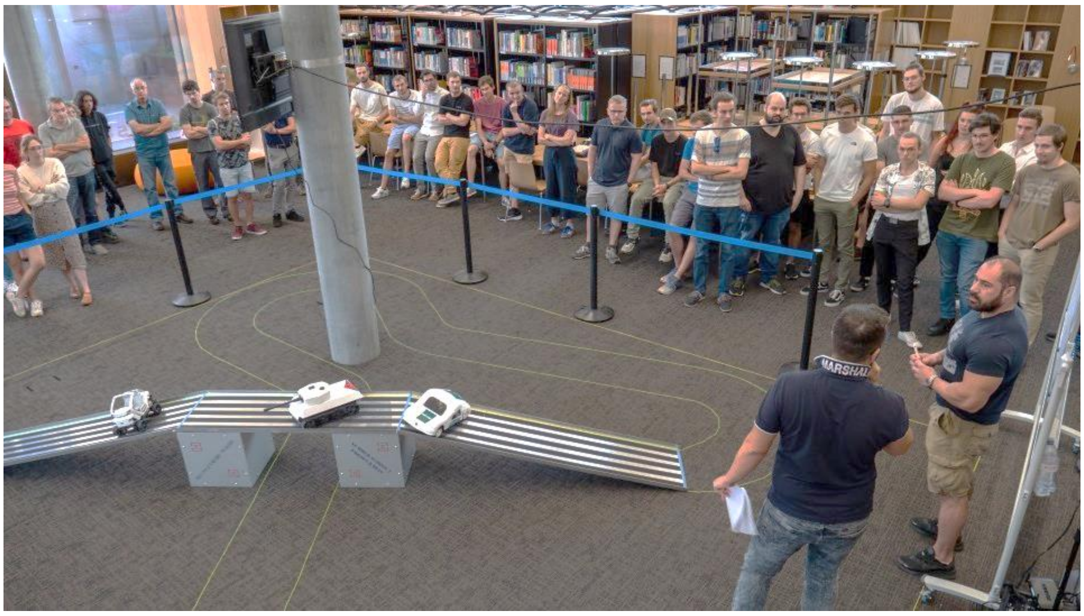


Figure 1 : Picture of the 2022 SS1 race [1]

1.2 AIM OF THE PROJECT

The aim of this bachelor's project is to provide a solution for automatically detecting, using cameras and in real-time, when a vehicle crosses the lines of the track markings. This project will also examine the possibilities and constraints of using a low-cost drone instead of a fixed camera.

Nowadays computer vision is an increasingly studied subject.

"..., the research in the field of computer vision purports to develop machines that can automate tasks that require visual cognition. However, the process of deciphering images, due to the significantly greater amount of multi-dimensional data that needs analysis, is much more complex than understanding other forms of binary information." [2]

This thesis will give an overview of the available strategies in computer vision from basic techniques to the more advanced ones and experiments which one works the best for this application.

1.3 GOALS

As the end of this thesis work coincides with the summer school's race, the system will be tested at this year's event. The project will then not only be focused on lines crossing detection but will cover everything related to hardware installation, user interface and user experience as well to ensure it can be tested live during the actual race.

According to the specifications defined at the start and available in the appendix, here are the different objectives of this project.

- **Analysis:** Establish the state of the art in algorithms for detecting overruns.
- **Implementation:** Design and implement a Proof-Of-Concept with fixed cameras.
- **Evaluation:** Evaluate the system's performance
- **Optional:** Analysis-Realisation-Evaluation with a drone

1.4 REPORT STRUCTURE

In the upcoming sections of this report, the different phases of this project will be dissected and presented in their order of execution. Except for the analysis part, each section follows a similar format to allow a comprehensive coverage of this project.

1. **Introduction**
   To assert their purpose, this part provides context about the section, an overview of what can be find in it as well as the metrics to evaluate the results.
2. **Conceptualization**
   It provides an explanation of the ideation process, the tools available, etc.
3. **Applied Efforts**
   It shows the conducted experiments and concrete work produced.
4. **Results and Forward Trajectory**
   The outcomes will be presented along the potential for future improvements.

In particular, Section 2 is the analysis part that discusses the problem specifications, presents a state-of-the-art, it shows the test made during the Python/OpenCV hands-on and also presents the different choices in camera and drones.
Section 3 presents the global system conception with architectures diagram, prototype definition and it also shows how a mock-up was built to test the system.
Section 4 discusses the prototype and the work achieved on its back and front-end part.
Section 5 presents the final proof-of-concept with its use during this year summer school race.
Section 6 explain the tests made to train a machine learning model.
Section 7 presents and discusses the results of this project.
Section 8 concludes this report and discusses future improvements.

2 ANALYSIS

2.1 PROBLEM SPECIFICATION

After discussing with the professors related to this project, Prof. François Corthay referred as the “client” who had the idea of this project and Prof. Gabriel Paciotti who is the manager of the Summer School 1, here is the information gathered:

2.1.1 PROJECT’S NEEDS

On the operator side, except for a technically working system, a basic and easy to use interface and installation is needed. The actions required to run the program should be minimal and straightforward.

The public and competitors should be able to watch the information displayed live during the race. The information will contain things such as timing of the progressing run, number of penalties detected and total time at least. Ideally, the position of the cars and the position of the crossed limits will be displayed on a map too.

And the end of a run, the interface must provide a way to manually correct the number of penalties in case of a misdetection. The time will then be registered and added to a scoreboard.

2.1.2 INSTALLATION CONSTRAINTS

The race will take place at the HESSO Valais-Wallis Sion, in the library. A way must be found to place and fix the camera(s) above the track. The ceiling is made of concrete, and it won’t be possible to fix anything directly on it as there are lights and sound reduction panel hanging under it. The cameras can’t be placed above the lights to avoid obstructing the field of view. A mobile lift platform can’t be used to avoid damaging the floor. There are large concrete columns where something like straps could be attached (see Figure 1).

The goal is to keep the maximum freedom in the cars design but modifications of the track such as changing the material used to define the limits are possible.

Figure 2: Previous vehicles built in 2020 [3]

2.1.3 Time constraints

The goal is to test a proof-of-concept (POC) during this year's summer school. The race will happen on September 8th.

The installation of the system will only be possible a few days before the race. Due to the event happening in the library, no tests will be possible earlier than this.

2.2 STATE OF THE ART

2.2.1 INTRODUCTION TO LINE CROSSING DETECTION

Nowadays, line crossing detection has become a pivotal topic in computer vision applications due to its wide range of practical implementations, ranging from security systems to self-driving cars. While utilizing out-of-the-box algorithms for line detection is relatively straightforward with current technologies, the primary challenge lies in developing a robust and reliable system that can withstand environmental variations.

This project necessitates the use of both line detection and tracking methods. This section will provide an overview of the different techniques currently employed in line detection and tracking.

2.2.2 IMAGE ACQUISITION AND PRE-PROCESSING

To detect lines or track an object on an image, first the image needs to be acquired. There is a wide range of cameras that can be used to capture footage.

Some of these camera's specifications will impact the results such as the field-of-view (FOV), resolution, framerate, etc.
Cameras are not perfect, internal calibration is needed to correct deformation caused by the lens. Then external calibration is used to correct positioning.

If the application requires to acquire live video frame it means a video stream is used. Live streaming protocols are vulnerable to the network quality. Since protocols prioritize framerate over image quality, it can lead to altered image being received when the network speed in not good.

Once an image or a video frame is acquired, pre-processing is applied to increase the quality of the result.

2.2.3 EDGE DETECTION ALGORITHMS

Edge detection is a fundamental process in image processing. Its goal is to identify boundaries and transitions between different regions in an image. An edge in an image is represented as significant shift in intensity or colour between adjacent pixels. The primary goal of edge detection is to highlight these boundaries, making them more distinct and facilitating subsequent analysis and feature extraction.

a) SOBEL EDGE DETECTION

Sobel Edge Detection is one of the most used algorithms for edge detection. It consists of spotting sudden changes in pixel intensity.
It will sweep the image in both x and y axis and detects the changes of intensity and computes the gradient. Then it will combine both axis and compute the gradient's magnitude and angle.

Edges can be extracted by applying a threshold to the gradient magnitude. [4]

b) CANNY EDGE DETECTION

This one is certainly the most popular as it is very robust and flexible. It's a four-step process:

- Noise Reduction
- Intensity Gradient of the image
- Suppression of false edges
- Hysteresis Thresholding

This method uses Sobel Edge detection to compute the intensity gradient, but it is composed of several other steps that enable it to produce precise and well-connected edges. [4]

2.2.4 Tracking and object detection

Object tracking in computer vision involves following and monitoring the movement of objects within a sequence of images or videos. The primary goal of object tracking is to maintain continuous identification of a specific object's location and trajectory as it navigates through frames.
There are a lot of techniques used to detect and/or track an object in an image. Here are a few of them listed.

a) Template Matching

This method consists of having a reference image of an object and to search for it in another image by comparing pixels.

Pros: Efficient
Cons: Subject to changes in rotation, scale, lightning conditions

b) Contour detection

This method consists of analysing the contours of an object and searching for a similar contour in another image. Contours can be defined by identifying the boundaries of an object using a method like Canny Edge Detection.

Pros: Efficient
Cons: Not great with objects that are complexly shaped or without clear contour

c) Optical Flow

This method consists of estimating the motion vectors of pixels between a series of frames.

Pros: Can be used for detection and tracking of object over time
Cons: Subject to occlusions, fast motion, complex scenes

d) Machine learning approaches

There are different types of neural network used in machine learning.

- Multi-Layer Perceptron (MLP)
- Convolutional Neural Networks (CNN)
- Recurrent Neural Networks (RNN)

The most common type used in computer vision is CNN.

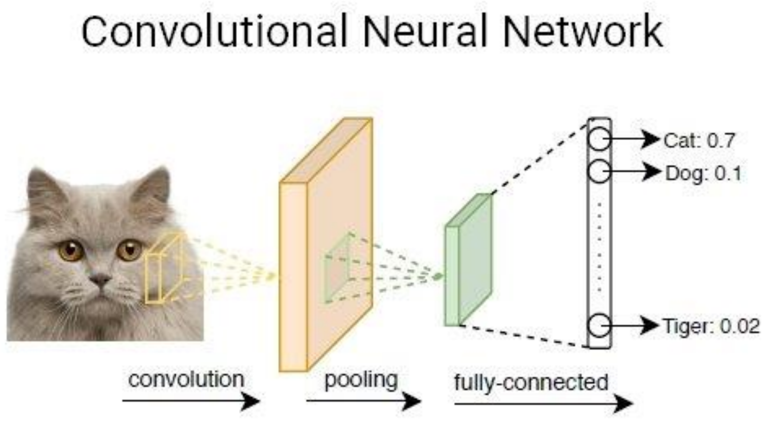


Figure 3 : Concept of CNN [5]

Main applications for CNN are image classification, object detection.

A convolutional layer applies filters to input data, extracting specific features and patterns by convolving the filters across the data. This process enables the network to detect edges, textures, and other significant characteristics in the input, contributing to the network's ability to recognize complex patterns and objects.

Pros: More robust and resistant to environmental changes
Cons: Requires loads of data and processing power to train a model

A more detailed and concrete machine learning application can be found in the section 6 - Machine Learning.

2.2.5 CHALLENGES AND LIMITATIONS

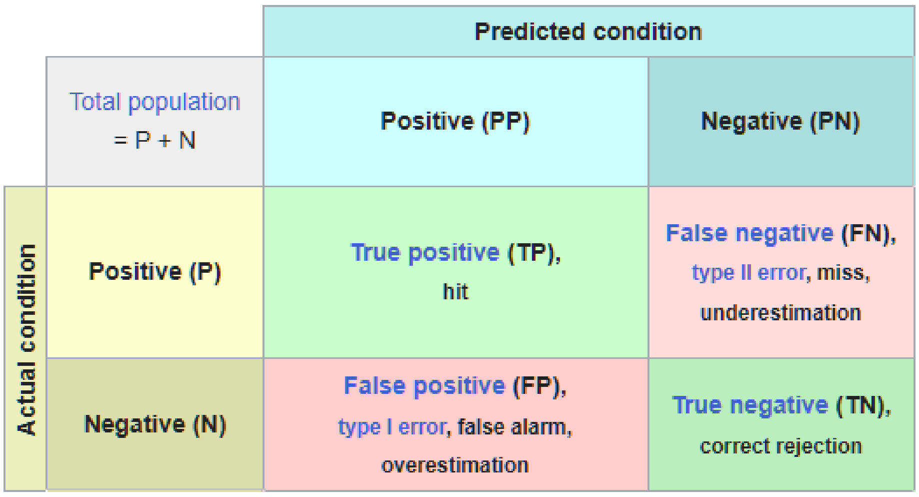
Despite the great advancements achieved in this field over the past years, several challenges ant limitations persist.

While the simpler approaches are easy to use "out-of-the-box" and can work great on simulation cases they tend to struggle to function properly in the real word. Most of these are not reliable enough when subject to varying light conditions, shadows, occlusions, image noise, etc. That's why the more complex techniques using machine learning are great as they can handle these external changes far better. But their limitations come from the size of the data needed and the processing power needed to train the models.

The same applies when it comes to the adaptability of the algorithms to diverse object shapes, sizes, and orientations. While some techniques excel in detecting simple shapes, they might struggle when confronted with irregular or complex objects, demanding more sophisticated approaches for robust detection and tracking.

2.2.6 EVALUATION METRICS AND DATASETS

The evaluation of line crossing detection and tracking algorithms necessitates the utilization of appropriate metrics and datasets. Common evaluation metrics for binary classification similar to this application includes accuracy, precision, recall, F1-score.



		Predicted condition	
	Total population = P + N	Positive (PP)	Negative (PN)
Actual condition	Positive (P)	True positive (TP), hit	False negative (FN), type II error, miss, underestimation
	Negative (N)	False positive (FP), type I error, false alarm, overestimation	True negative (TN), correct rejection

Figure 4: Confusion matrix description [6]

These metrics are based on the confusion matrix, here are their equations [6].

$$accuracy = \frac{TP + TN}{P + N}$$

$$recall\ or\ TPR = \frac{TP}{TP + FN}$$

$$precision\ or\ PPV = \frac{TP}{TP + FP}$$

$$F_1\ score = 2 * \frac{PPV * TPR}{PPV + TPR}$$

“Accuracy is defined as simply as the number of correctly categorized examples divided by the total number of examples. […] The accuracy has the advantage that it is very easily interpretable, but the disadvantage that it is not robust when the data is unevenly distributed, or where there is a higher cost associated with a particular type of error.” [7]

On the other hand, F1 score is a combination of precision and recall and provides a balanced measure of the accuracy, particularly in situations where there is an uneven distribution of classes or imbalanced datasets.

Concerning machine learning, the models training is often evaluated using learning curves or ROC curve. These metrics provide quantitative measures of the algorithm performance, enabling objective comparisons between the different methods. The results produced by a model are evaluated with the same metrics presented above.

2.2.7 Conclusions and future directions

In conclusion, exploring the different line crossing detection and tracking techniques reveals and dynamic landscape of algorithms and technologies.
Going from traditional methods using simple math functions to the highly complex model pretrained, there is a lot of ground to cover. But as always, each method comes with its set of advantages and limitations. And it is rare to find an existing technique that perfectly suits a specific application. It may be either specific to a project but hardly adaptable or easily modifiable but too generic, demanding significant effort to adapt it.

The goal of this project will be to test a selection of these techniques and choose one or multiple ones to be further developed and adapted to the specific needs of this system.

2.3 HANDS ON

2.3.1 OPENCV USING PYTHON

If you ask a developer what comes to his mind when image processing is mentioned, he will probably say OpenCV.

OpenCV (Open-Source Computer Vision) was initially developed by Intel and is a free cross-platform computer vision library for real-time image processing. [8]

"The OpenCV software has become a de-facto standard tool for all things related to Computer Vision." [8]

Its first release was at the beginning of the 2000s and even nowadays this library is still highly popular with about 29'000 downloads per week.

"The goal of OpenCV is to provide an easy-to-use computer vision infrastructure that helps people build sophisticated vision applications quickly by providing over 500 functions that span many areas in vision." [8]

It is written in C and C++ and compatible with most popular operating systems.

A fun fact about OpenCV: it even has been used in sound and music recognition where visual recognition techniques have been applied to sound spectrograms images. [8]

The primary interface for OpenCV is C++ [9], but interfaces have been created so it can be easily used with Python, Java and even MATLAB.

Since most examples and samples projects found online are coded in Python it was decided to use it.

The goal of this hands-on is to discover the variety of functions offered by OpenCV and to learn using it as well as familiarizing with the Python environment.

2.3.2 SETTING UP THE DEVELOPING ENVIRONMENT

JetBrains's PyCharm IDE is specifically designed for Python programming. It was the IDE used for this project as it offers a lot of tools that boosts productivity such as code completion, syntax highlighting, intelligent code analysis and refactoring. It also supports plugins like GitHub Co-pilot and has built-in version control systems.

Figure 5 : PyCharm's logo [10]

One of the most practical features is the support of virtual environment. It enables each project to have its own packages and dependencies installed without affecting the global Python environment of the PC. It makes it easy to browse through the different package's version and install the one needed.

Version control has been used to save snapshots of the code during the different phases.

An online repository is available on Gitlab, it also contains more information about the project's setup in the Read.me file:

https://gitlab.hevs.ch/SPL/bachelorthesis/2023-relicrode/opencv_intro

2.3.3 TEST PROGRAM

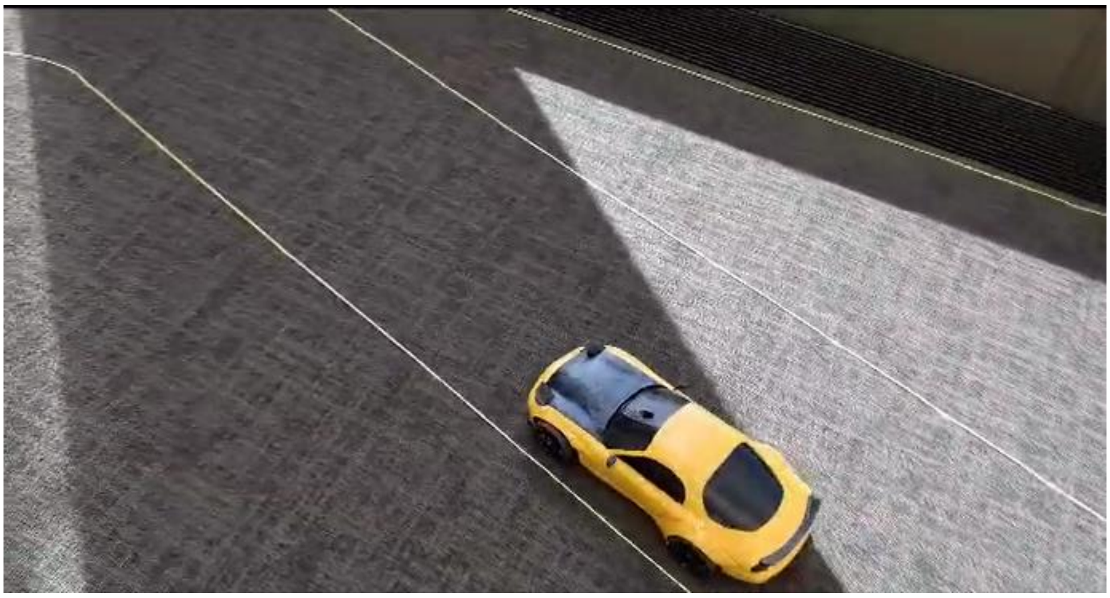
The methods mentioned in the state-of-the-art section will be tried out using Python scripts and the OpenCV library. These tests won't have a simple "pass" or "fail" result, but they will help in deciding which technique to use in the prototypes.

It must be noted that the following tests are not exhaustive. The results could be improved by applying pre- or post-processing such as contrast or luminosity modifications or by further tweaking some of the parameters.

2.3.4 MATERIAL USED

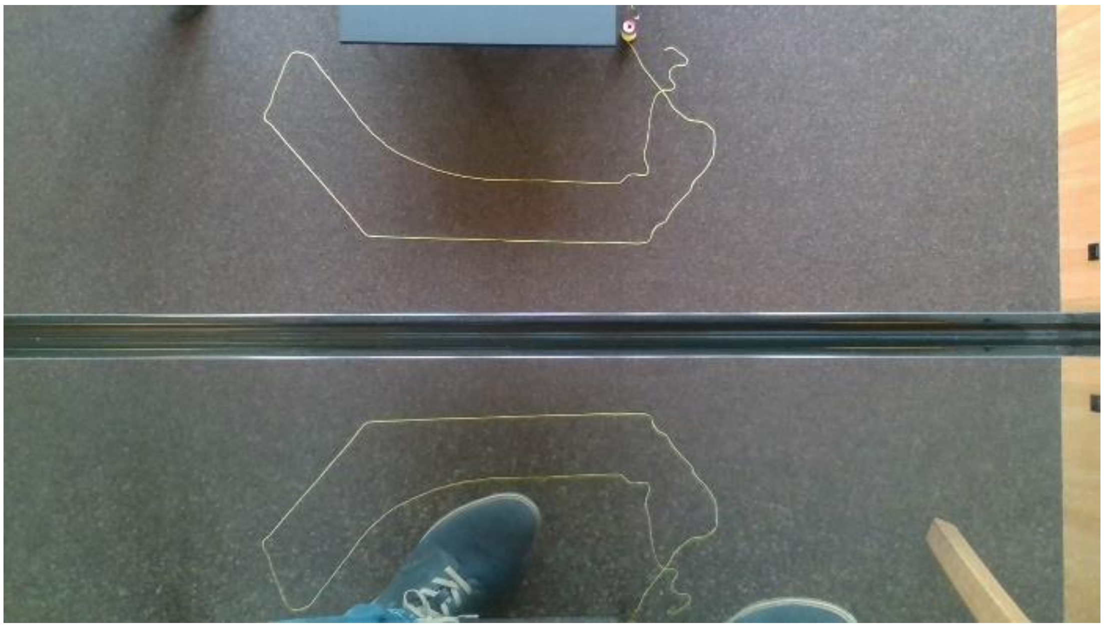
Testing image processing functions obviously requires some images. While the exact setup of the race was not possible to reproduce, here's the attempts to approach it that were made.

a) STILL IMAGE FROM LAST YEAR'S RACE



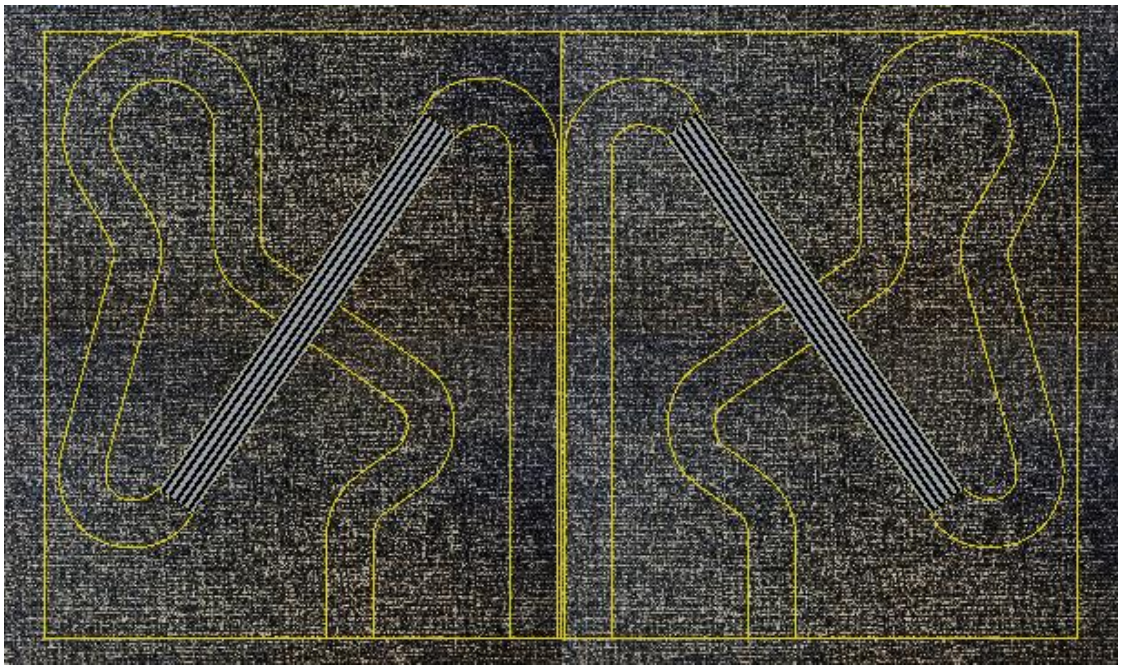
This image is a screenshot extracted from a video shot during last year's training.
It shows the exact floor that will be present this year too, has some challenging lighting conditions because of the window's shadows and enables to test the detection of this light-yellow thread used to mark track limits. But it is not exactly a top-down view and is taken too close from the ground.

b) Simple thread on the floor



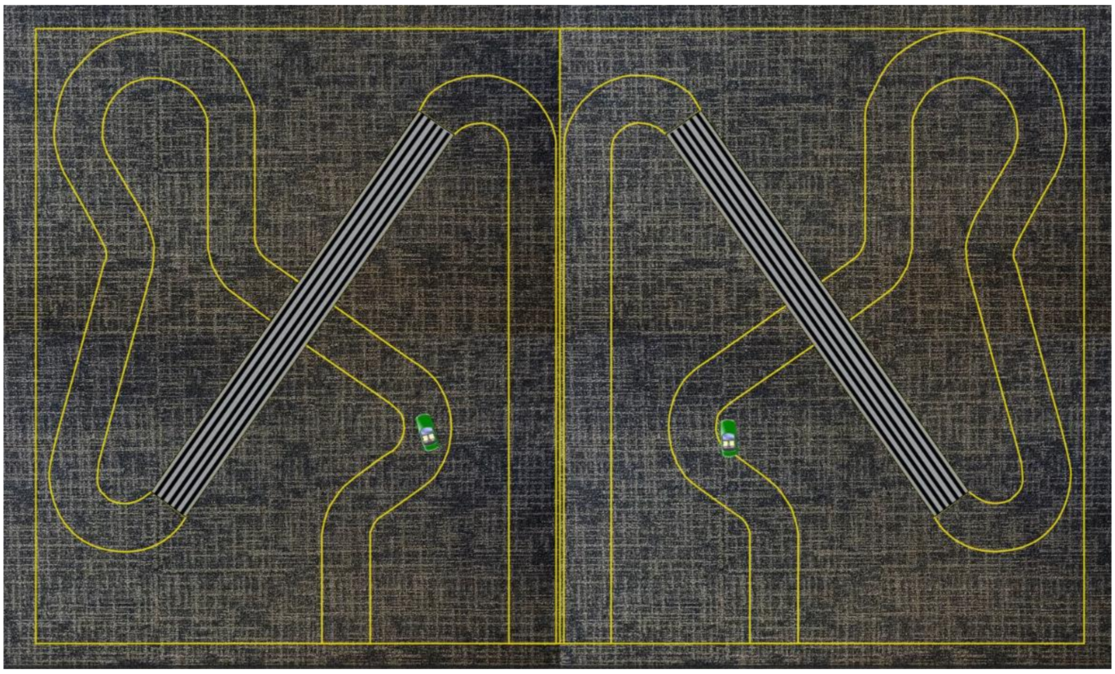
This shot tries to better recreate the angle and height of the camera. Even if it is not the exact floor of the library it gets close enough. The thread is simply laid on the floor with some curves and straight lines.

c) Virtual top-down view of the track



Using the track drawings and measurements this image was created. The floor was made by stitching multiple times a picture of the library floor.
This image solves a lot of the precedent picture's problems. But its downside is the fact that it is virtual and does not reflect a camera's noise and imperfections. It could be solved by virtually adding noise but wasn't necessary at this point. The thread's size is also a bit bigger than in real size.

d) VIRTUAL RACE ANIMATION



Later, a race animation was made. It shows a car on the left perfectly following the track. But on the right side, the car crosses the limits a few times.

2.3.5 LINE DETECTION

As the title of this thesis suggests, the first method that was tested was the line detection algorithm that uses Hough Transform. But this algorithm detects geometrical lines not lines in the sense of lanes. Therefore, it's hard to extract useful data in our case since the track is principally made of curves.

The function used is cv.HoughLines and it is described in the official OpenCV documentation. [11]

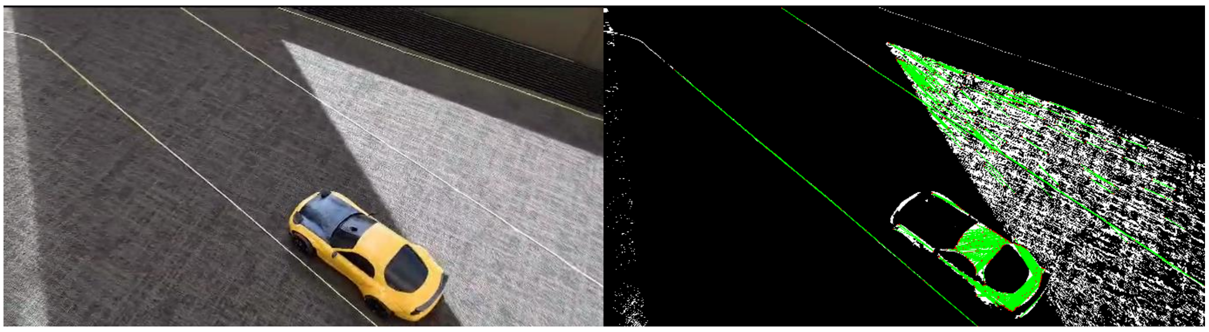


Figure 6 : Première image de test

This function is applied on a binarized image, and it showed the difficulty to select a good threshold value. It was hard to isolate the thread only.

2.3.6 EDGE DETECTION

Now let’s try to detect edges and contours instead of straight lines only.

First using the Sobel Edge Detection algorithm:

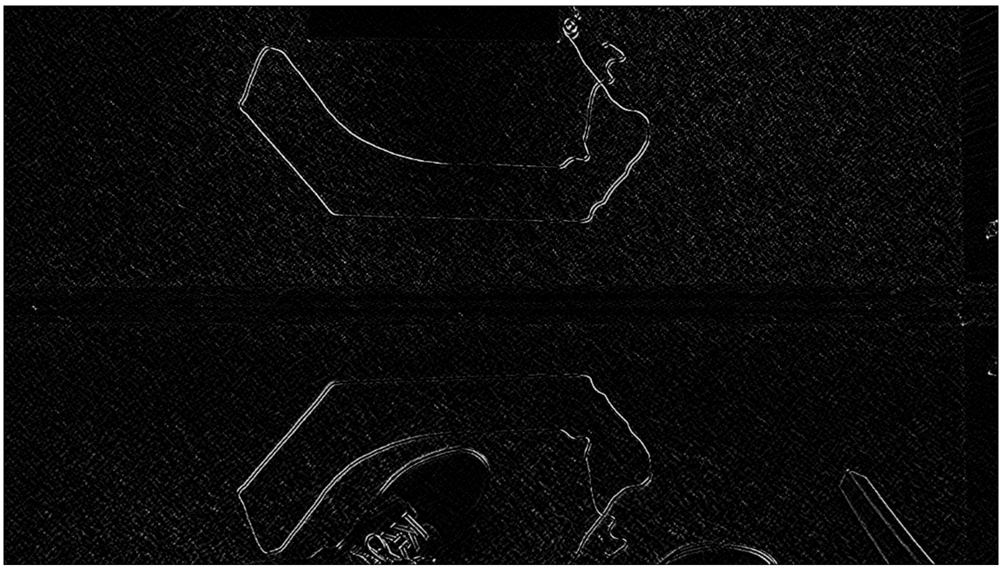


Figure 7 : Sobel edge detection on thread test image

Now using the Canny Edge detection:

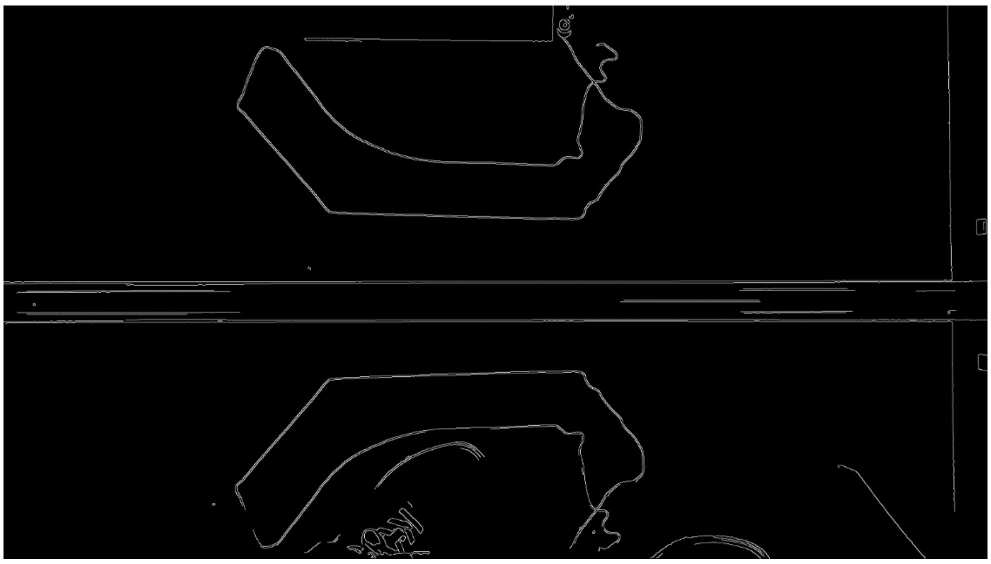


Figure 8 : Canny edge detection on thread test image

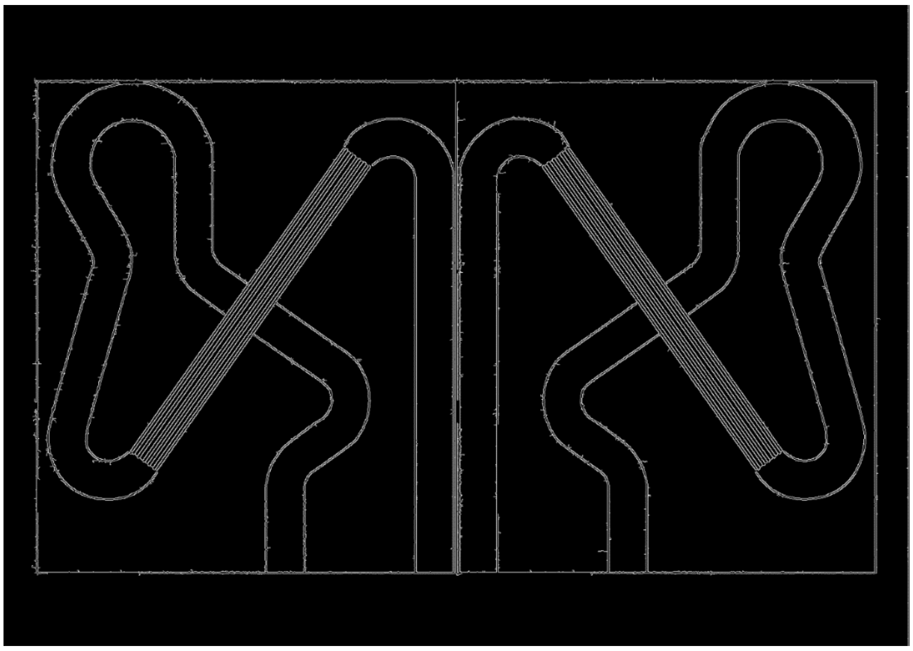


Figure 9 : Canny edge detection on the circuit's virtual image with better parameters

As seen in the state-of-the-art part, the Canny edge detection uses Sobel edge detection but adds pre- and post-processing steps, giving much cleaner results as shown with those tests. This method suits this application a lot better than the line detection one.

2.3.7 CONTOUR DETECTION

Let’s try applying contours detection on the detected edges.

The edge detection functions gives as a result another image with edges highlighted. Whereas the goal of the contour detection function is to classify those edges.

The function cv.findContours returns a binary image, an array with the detected contours represented as a vector of points and an array with each contour’s hierarchy.

Hierarchy is represented by an array of four values: **[Next, Previous, First_Child, Parent]**

Next: denotes next contour at the same hierarchical level.
Previous: denotes previous contour at the same hierarchical level.
First_Child: denotes its first child contour.
Parent: denotes index of its parent contour.

This hierarchy classification will be used later in the project.

OpenCV also provides a very handful function that allows to draw the detected contours called cv.drawContours. The results can be seen on the next image.

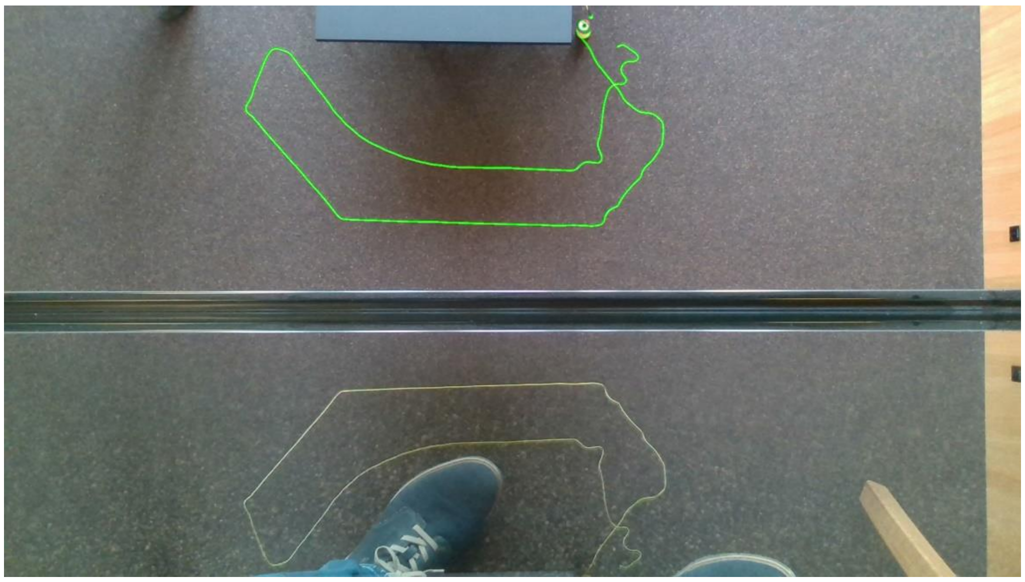


Figure 10 : Contours detection after Canny edge detection

2.3.9 BASIC OBJECT TRACKING

OpenCv provides an object tracker, and you can choose between different algorithms.

More details on these algorithms can be found online but here is a list with their pros and cons found on the LearnOpenCV website [12].

e) **BOOSTING TRACKER**

   Pros: None. This algorithm is a decade old and works ok, but other advanced trackers (MIL, KCF) based on similar principles are available.
   Cons: Tracking performance is mediocre. It does not reliably know when tracking has failed.

f) **MIL TRACKER**

   Pros: Performance is pretty good. It does not drift as much as the BOOSTING tracker and handles partial occlusion.
   Cons: Tracking failure is not reported reliably. Does not recover from full occlusion.

g) **KCF TRACKER**

   Pros: Accuracy and speed are both better than MIL. Reports tracking failure better than BOOSTING and MIL.
   Cons: Does not recover from full occlusion.

h) **TLD TRACKER (TRACKING, LEARNING, AND DETECTION)**

   Pros: Works best under occlusion over multiple frames. Tracks well over scale changes.
   Cons: Lots of false positives making it almost unusable.

i) **MEDIANFLOW TRACKER**

   Pros: Excellent tracking failure reporting. Works well when motion is predictable and there is no occlusion.
   Cons: Fails under large motion.

j) **GOTURN TRACKER**

   Pros: Robust to viewpoint changes, lighting changes, and deformations. Operates based on Convolutional Neural Network (CNN).
   Cons: Does not handle occlusion well. Requires the presence of Caffe model files.

k) **MOSSE TRACKER**

   Pros: Robust to lighting, scale, pose, and non-rigid deformations. Fast operation at a higher fps. Easy to implement.
   Cons: Lags behind deep learning-based trackers in performance.

l) **CSRT TRACKER (DISCRIMINATIVE CORRELATION FILTER WITH CHANNEL AND SPATIAL RELIABILITY)**

   Pros: Adjusts filter support for non-rectangular regions or objects. Higher accuracy for object tracking.
   Cons: Operates at a lower fps.

They have been tested with the virtual animation described earlier.
Some of them are deprecated, and only a few worked in this case. But none of them could track the car when it drove under the bridge. While some detect the failure, others don't and just keep tracking the same spot at the entry of the bridge.

2.3.10 Particles filter tracking

The goal with a particle filter is to have a large amount of small independent particles under some basic physic rules. Each particle will be tested and evaluated. For object tracking, they will be tested on their absolute position over the tracked object. Based on these tests different weights will be assigned. The good particles will be multiplied, and the bad ones removed. The cycle will repeat for each frame of the footage.

A test script was written following this online project [13].

The result depends greatly on the parameters of the particles such as the reward for being at the right place or the noise amplitude applied to them. After a bit of tweaking the particles tracked the green car on the right all the way even under the bridge or during the 180° turn. More screenshots of the tests can be found in the appendices.

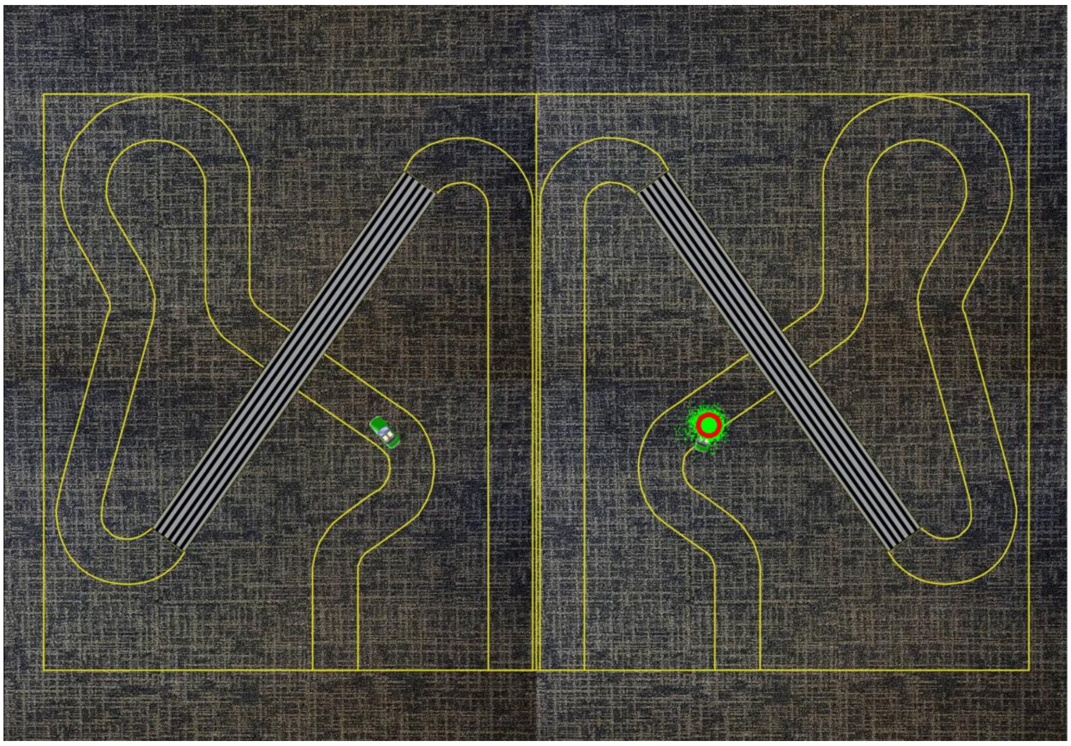
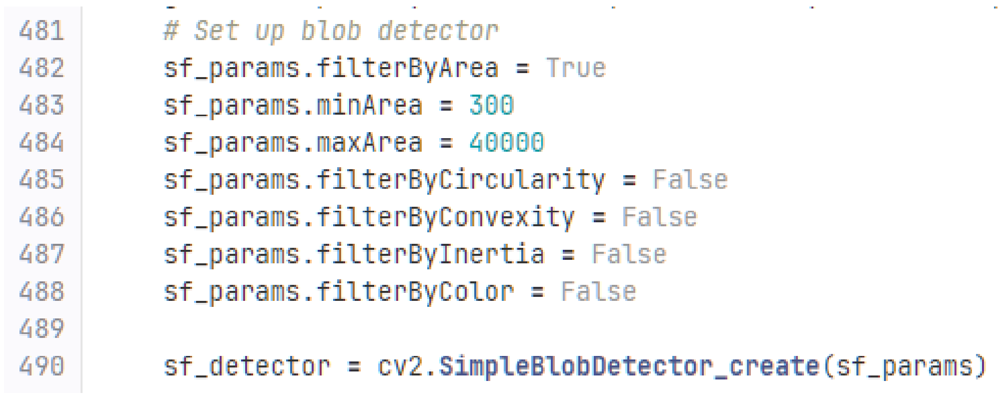


Figure 11 : Particles filter tracking test

2.3.11 Blob detection

OpenCV offers a feature called blob detection. Its purpose is to locate the areas of an image that share common the same properties such as colour or intensity.

There are a lot of detection parameters that can be set to configure the detected objects requirements such a minimum or maximum area, a specific colour, convexity, etc.



```python
# Set up blob detector
sf_params.filterByArea = True
sf_params.minArea = 300
sf_params.maxArea = 40000
sf_params.filterByCircularity = False
sf_params.filterByConvexity = False
sf_params.filterByInertia = False
sf_params.filterByColor = False

sf_detector = cv2.SimpleBlobDetector_create(sf_params)
```

Figure 12: Setup of a blob detector in OpenCV

It could be used to detect a specific object like the car, but the tests were not convincing. But it can be very useful to count or sort some objects.

2.3.12 CAMERA INTERNAL CALIBRATION

Most cameras are exposed to lens distortion. For everyday photography it can easily be overlooked but in computer vision it can affect the system performance.

Since it is a common problem, it has already been studied and well documented in the OpenCV documentation.

“Radial distortion causes straight lines to appear curved. Radial distortion becomes larger the farther points are from the center of the image. […] Similarly, tangential distortion occurs because the image-taking lens is not aligned perfectly parallel to the imaging plane. So, some areas in the image may look nearer than expected.” [13]

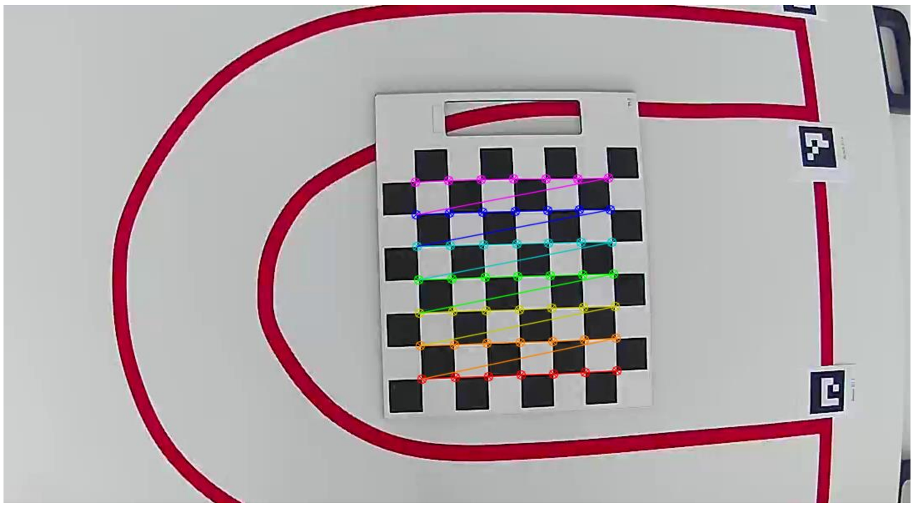
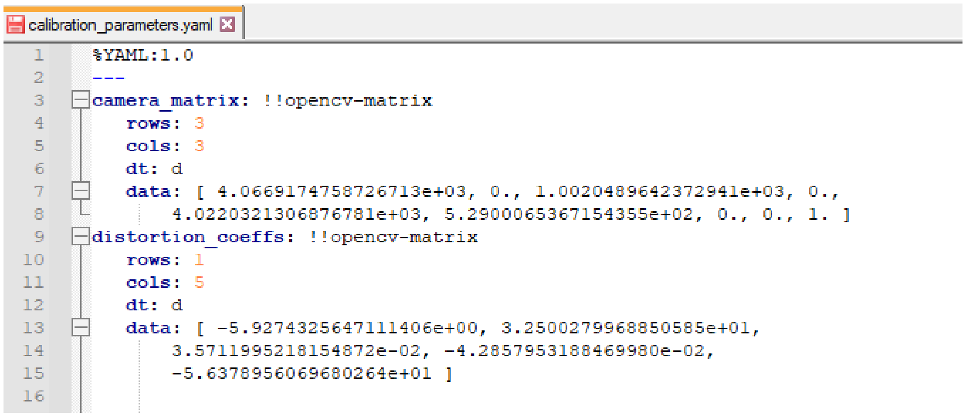


Figure 13: Calibration plate with detected chessboard

To calibrate the camera, a set of images with a calibration plate in the frame must be collected. Then the script will detect the chessboards and collect the points needed to calculate a matrix.


calibration_parameters.yaml

```
%YAML:1.0
---
camera_matrix: !!opencv-matrix
   rows: 3
   cols: 3
   dt: d
   data: [ 4.0669174758726713e+03, 0., 1.0020489642372941e+03, 0.,
       4.0220321306876781e+03, 5.2900065367154355e+02, 0., 0., 1. ]
distortion_coeffs: !!opencv-matrix
   rows: 1
   cols: 5
   dt: d
   data: [ -5.9274325647111406e+00, 3.2500279968850585e+01,
       3.5711995218154872e-02, -4.2857953188469980e-02,
       -5.6378956069680264e+01 ]

```

Figure 14: YAML file containing calibration parameters

This matrix and distortion coefficients are saved in a YAML file and can then be reused for any picture taken with the same camera.

Here is the before and after comparison:

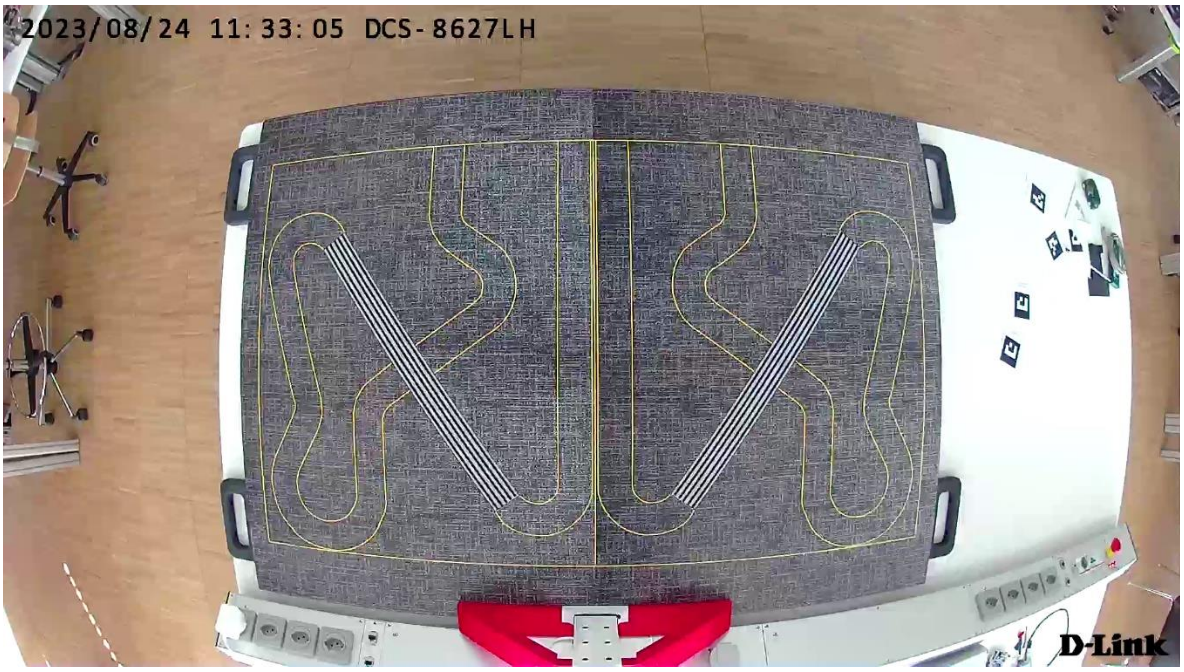


Figure 15: Test image before internal calibration

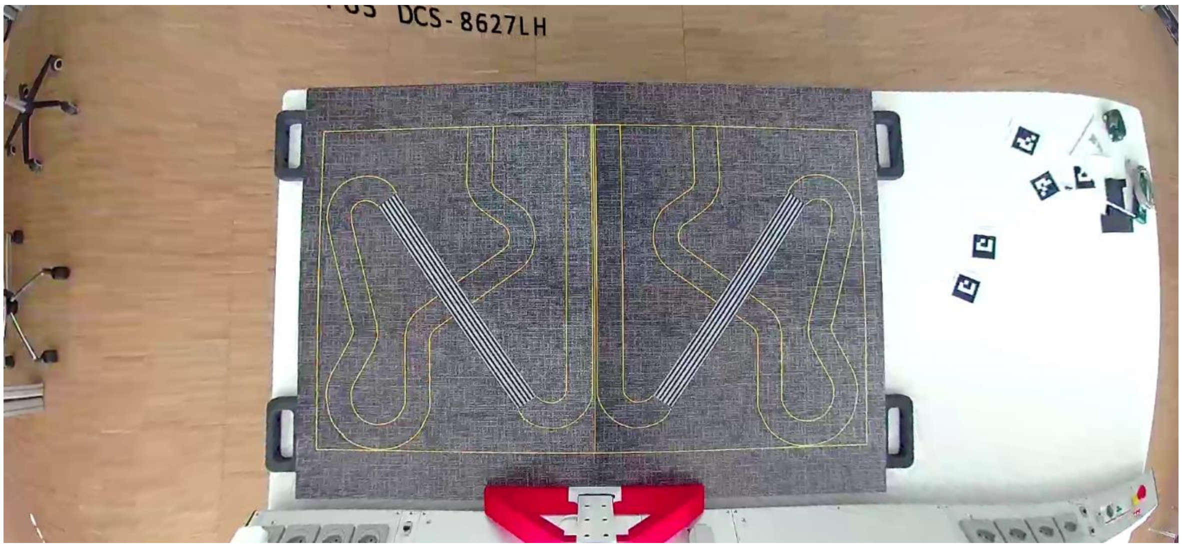


Figure 16: Test image after internal calibration

The difference before and after for this image is clear as this test's camera has a lot of distortion due to its wide field-of-view. The borders of the image are still distorted but its center is much straighter.

2.3.13 CAMERA EXTERNAL CALIBRATION

Now that the lens distortion problems are fixed, it is time to address the external calibration. This will correct the perspective deformation in case of the camera not pointing straight down.

The concept is to have both a reference and the deformed image, to select four points and get their respective coordinates on both images.

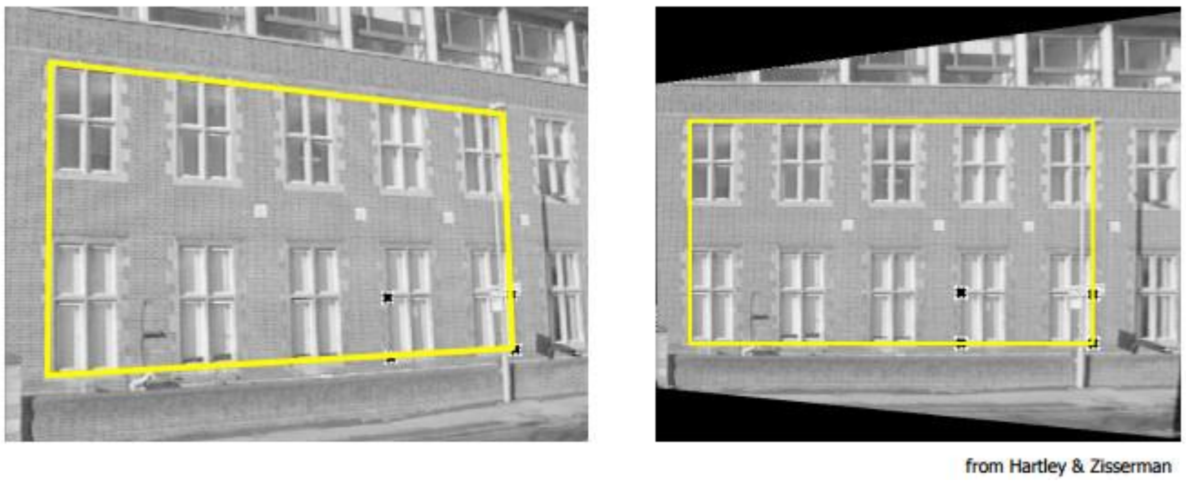


Figure 17: Perspective transformation example

The cv2.getPerspectiveTransform function will then compute a 3x3 matrix for the perspective transform. The matrix can then be used with the cv2.warpPerspective function to correct an image.

To collect coordinates on both reference and the image to correct, the plan is to use ArUco markers (a simplified version of QR codes). These are distinctive black-and-white square patterns commonly used in computer vision. Each marker can have its own ID.

To test this idea, four markers were added in the corner of the racetrack and a warped image was created. On the image below you can see more than four markers, but the script will only use four.

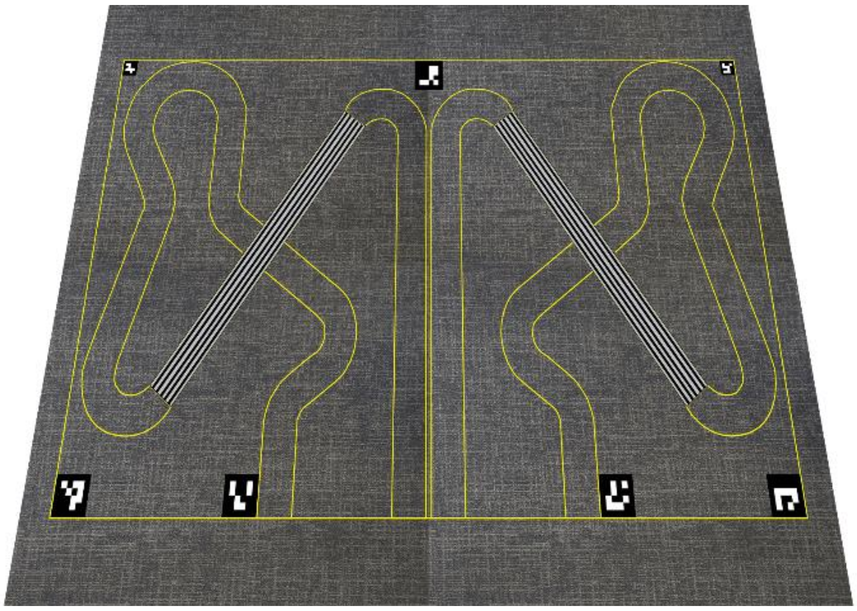


Figure 18: Warped image

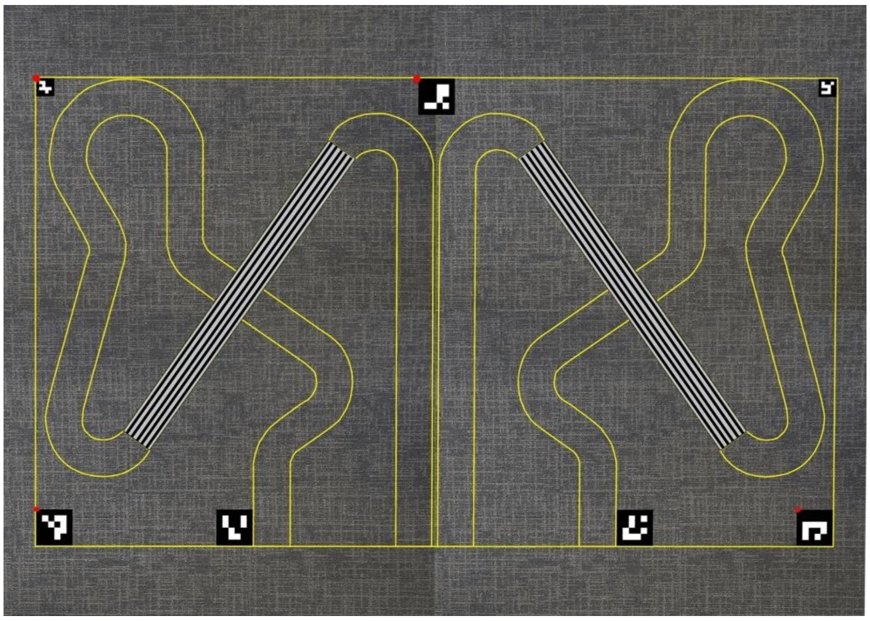


Figure 19: Result image with perspective correction applied

The results are good. This process can be done once and then the matrix can be reused while the camera position is not changed.

2.4 CAMERA

In this digital age, the choice of camera available is massive but not all will suit this project's needs.

For the initial tests an Intel Real Sense Depth Camera D415 was used. This camera provides 3D scanning features, but they were not used. It served as a point of reference for the future search.

If no camera with the right specifications is found there is the possibility to install two of them to cover all the track but that would require substantial additional work to make them work together.

2.4.1 CRITERIA

Here are the search criteria used.

a) **FIELD OF VIEW (FOV):**

Some manufacturers only share one value for FOV, and it corresponds to the diagonal FOV which is harder to work with as it depends on image format. But in most datasheets both horizontal and vertical view angle are shared. Those values are needed to make sure all the track area can be covered in the frame. There is a direct relation between the camera's elevation above the ground and the area in the FOV. The higher the FOV values are, the lower the camera needs to be placed, but with higher FOV comes higher lens distortion.

Lens distortion can be fixed in software so the limiting factor will be the maximum height placement. Therefore, the goal is to find a camera with a relatively high FOV.

b) **HEIGHT PLACEMENT**

As explained in the previous part there is a corresponding height placement for each FOV value. This height can't be greater than the library setup allows. Anything above six meters will not work.

The next section explains briefly how the FOV/Height relation was computed.

c) **CONNECTIVITY**

There are multiples ways to retrieve the live data from a camera. The most common would be to have it directly connected to the PC via USB. But that would not be practical as it would require a long cable which will decrease the video resolution. Another solution would be via any streaming protocols over internet either via Wi-Fi or ethernet cable.

d) **POWER SUPPLY**

A battery powered device would mean less cables to route up in the air above the track but would be less practical if it needs to be charged too frequently.

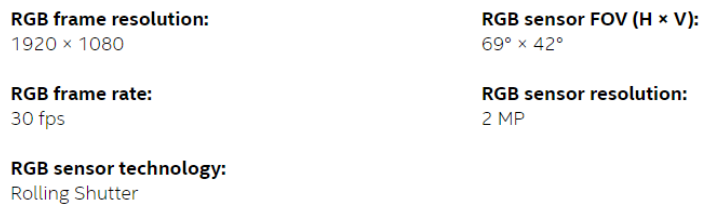
e) **RESOLUTION**

Nowadays most cameras sold on the market provides a decent resolution, but the camera should at least record in Full HD. We can then reduce the quality or frame per second if needed.

2.4.2 FOV & Height Relation

Using Onshape (more details on this website in the Section 0 - Mock-up) to draw in 3D the whole installation allows to then make drawings in 2D and measure the angles.

Let’s take the Intel D415 as an example.



RGB frame resolution:
1920 × 1080

RGB frame rate:
30 fps

RGB sensor technology:
Rolling Shutter

RGB sensor FOV (H × V):
69° × 42°

RGB sensor resolution:
2 MP

Figure 20: Extract from Intel D415's datasheet

The complete track layout with measurements can be found in the appendices. It has a length of 11 meters and a width of 6.5 meters.

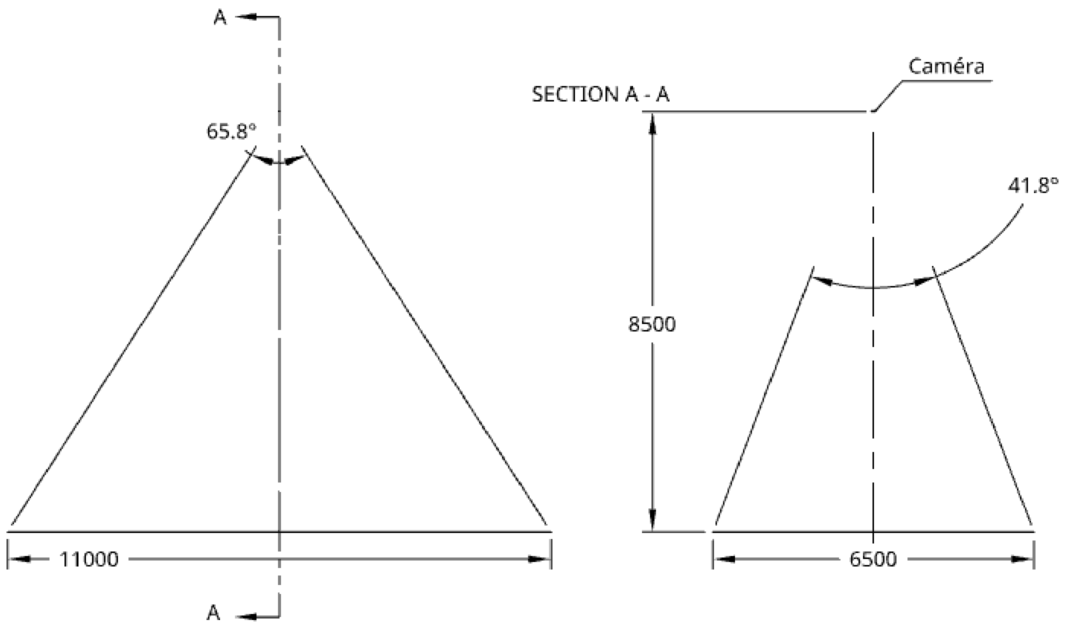


Figure 21: Intel D415 placement for full track view

With an FOV of 69°x42° the camera needs to be placed at least 8.5 meters above the ground to have the whole track in the frame.

2.4.3 RECAPITULATIVE TABLE

Name	FOV	Height	Connectivity	Power supply	Resolution @ 30fps	Price [chf]
Intel D415	69°x42°	8.5 m	USB	USB	1920x1080	250
Raspberry WWCAM	122°x89.5°	3.4 m	Direct to Raspberry	Raspberry	1920x1080	25
Axis M2025-LE	115°x64	5.5 m	Ethernet cable	POE	1920x1080	660
DLink DCS8627-LH	123.8°x65.4°	5.5 m	WiFi	Power cable	1920x1080	109

The Axis camera would be a great option if only the lens had not been replaced reducing significantly the FOV. Using a Raspberry powered on a battery bank with a camera module would be a very flexible solution but would require some work to setup the video stream transmit. Connecting a webcam like the D415 with a USB cable is not possible due to the loss of quality caused by the length of the cable required.

The most suitable camera is the DLink DCS-8627LH, its downside come from the fact that is not a camera made for industrial or professional use. The features available to configure it are limited.

2.5 *DRONE*

2.5.1 CONCEPT

Stated in this thesis description, there is the optional goal to use a drone for this project instead of a fixed camera. This section is an analysis of the potential to integrate a drone.

2.5.2 PROS AND CONS OF USING A DRONE

The idea behind the use of a drone is to make it fly directly over and made it follow the car as it runs down the track. There wouldn't be a good enough reason to have a drone flying still above the track as it would give the same result than a fixed camera.

The advantage of a drone flying directly over the car is that it would be more precise. It would greatly reduce the problem of markings being obstructed due to the height of the vehicle (see Figure 22). It would avoid interferences from other persons or object on the track as the view will be focused on the car and its proximity.

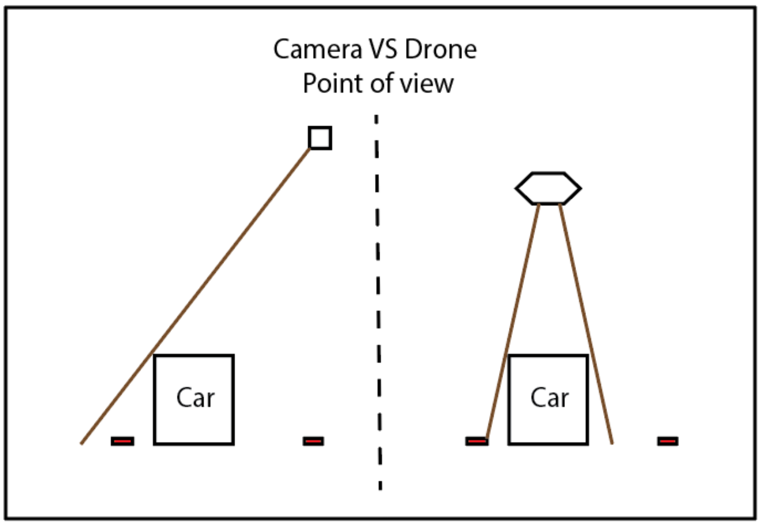


Figure 22: Representation of the difference in P.O.V between a camera and drone setup

However, flying a drone inside a close space such as the library bring some safety concerns. The drone would be flying quite low, making a lot of noise. Most camera drone have an autonomy between 15 to 30 minutes which would require frequent battery changes during the race.

2.5.3 OPTIONS AVAILABLE

Here are the required specifications the drone must have:

a) **TRACKING CAPABILITIES**

Either with an integrated feature or custom flight controller, the drone must be able to follow the car.

b) **DOWN FACING CAMERA**

The camera must be able to point down.

c) **OPEN VIDEO FEED**

There must be a way to acquire the video feed and transmit it to the image processing unit of this project.

d) **DECENT SIZE**

The library is quite small, the smaller the drone can be the better.

e) **DECENT FLIGHT TIME**

If the autonomy is too short, it would require too many battery swaps.

f) **LOW PRICE**

The budget is limited to maximum 500 CHF.

After some research online, here are the different options found.

First, there is the DJI Tello Drone.

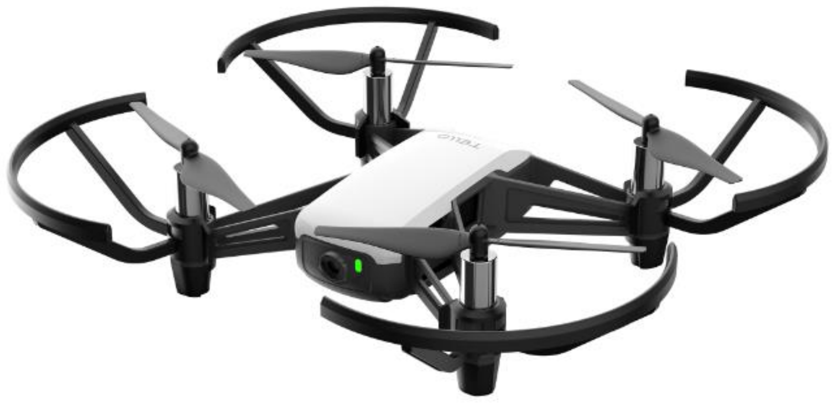


Figure 23: Dji Tello Drone [14]

The DJI Tello drone is a compact, beginner-friendly quadcopter with a user-friendly app, stable flight, basic camera capabilities, and programmable features. It offers a Software Development Kit that allows to create custom applications and control the drone's flight, camera, and other functionalities. With a low price of 115 euros, it would be the ideal choice. But the camera is fixed and pointing straight at the horizon.

There is also the Parrot Anafi Ai:

Figure 24: Parrot Anafi Ai Drone [15]

The Parrot Anafi Ai is an advanced foldable drone equipped with AI-powered features, 4K HDR camera, thermal imaging capabilities, and a developer-friendly SDK for creating specialized applications. As an industrial drone it matches almost all criteria except for the cost, with a selling price of 4000 euros, it won't fit in this project's budget.

An alternative is to build a custom drone. It something quite common in the RC drone world. It would be a great solution as it would be possible to build one that matches exactly all criteria.

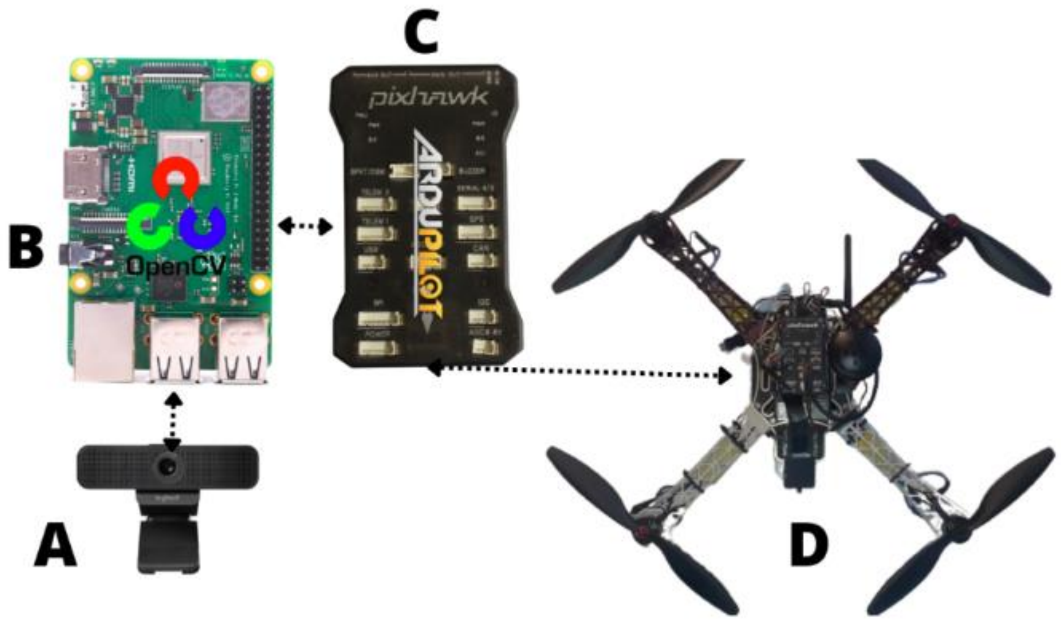


Figure 25: OpenCV drone diagram [16]

Once the drone is assembled form each individual components such as motors, flight controller, frame, battery, it then requires programming the flight controller with the functionalities desired.

Like what has been done in this project [16] found online, a custom flight software can be implemented, to track markers that would be on top of the cars during the race.

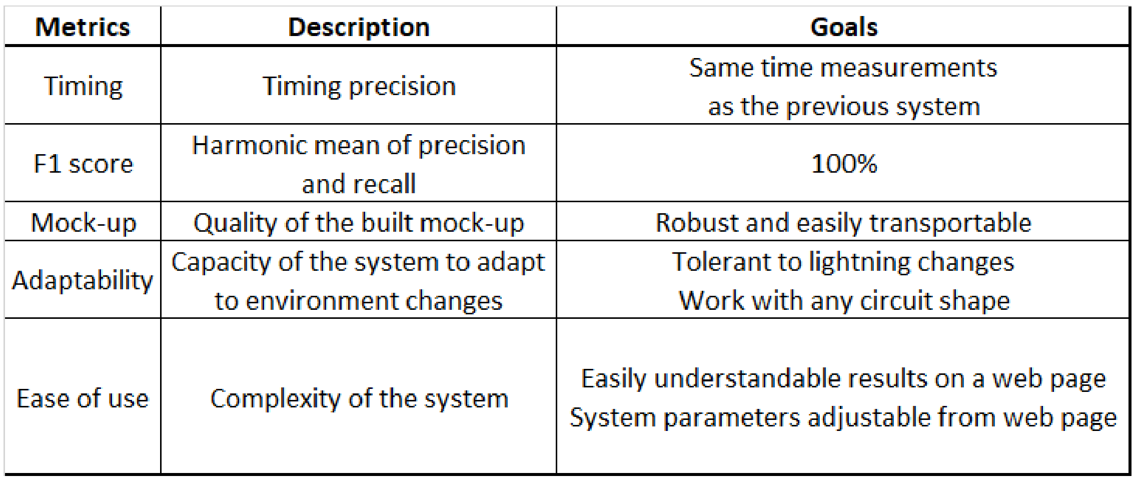
The downside is that it would be really time consuming as the price of individual components is traded with the time needed to build and program the drone.

2.5.4 Conclusion

It's hard to find a drone that matches all requirements. The most suitable idea is to build a custom drone, but it would involve a lot of building, programming, and testing given it would only provide the images needed to do the line crossing detection.

Given the time and resource constraints of this project, we decided to put aside the drone implementation.

2.6 *Evaluations Metrics*



Metrics	Description	Goals
Timing	Timing precision	Same time measurements as the previous system
F1 score	Harmonic mean of precision and recall	100%
Mock-up	Quality of the built mock-up	Robust and easily transportable
Adaptability	Capacity of the system to adapt to environment changes	Tolerant to lightning changes Work with any circuit shape
Ease of use	Complexity of the system	Easily understandable results on a web page System parameters adjustable from web page

Figure 26: Evaluation metrics defined for this project

Based on the analysis made, here is the evaluation metrics chosen for this project. They will be used to evaluate the results of the prototypes or final proof-of-concept.

Note the 100% F_1 score goal may be too ambitious but for this application, no error can be tolerated as all the teams needs to have fair judgment. This is a long-term goal while the system can be improved and manual correction is applied to detected errors.

3 CONCEPTION

3.1 GOAL

Now that the different tools and techniques available are clearer, it's time to move towards a functional proof-of-concept.

The Figure 27 shows the four global tasks to achieve.

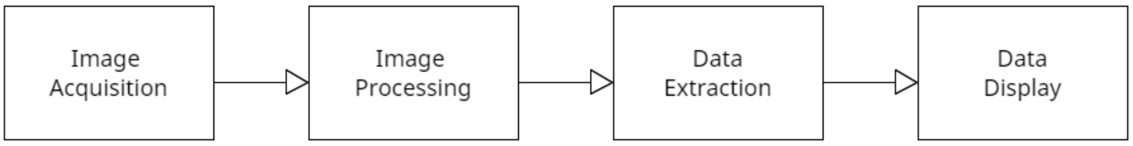


Figure 27: Decomposition in sub- tasks

The main challenge of this project is the image processing and data extraction part. But since the system will be tested live during the race, a decent user experience enabling data visualization and parameters control must be present.

This section discusses the initial design phase but as this is an iterative project process, the different elements presented will evolve.

3.2 ACTORS

To start the conception of this system, it is essential to define the different actors that will interact with it. Here is an UML diagram that represents them.

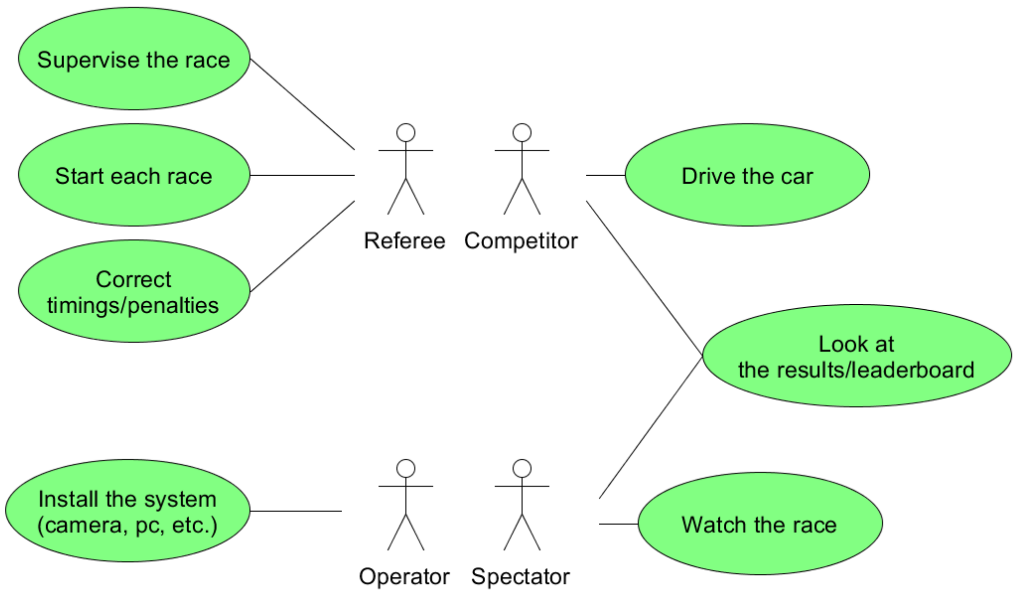


Figure 28: Actors and their respective tasks

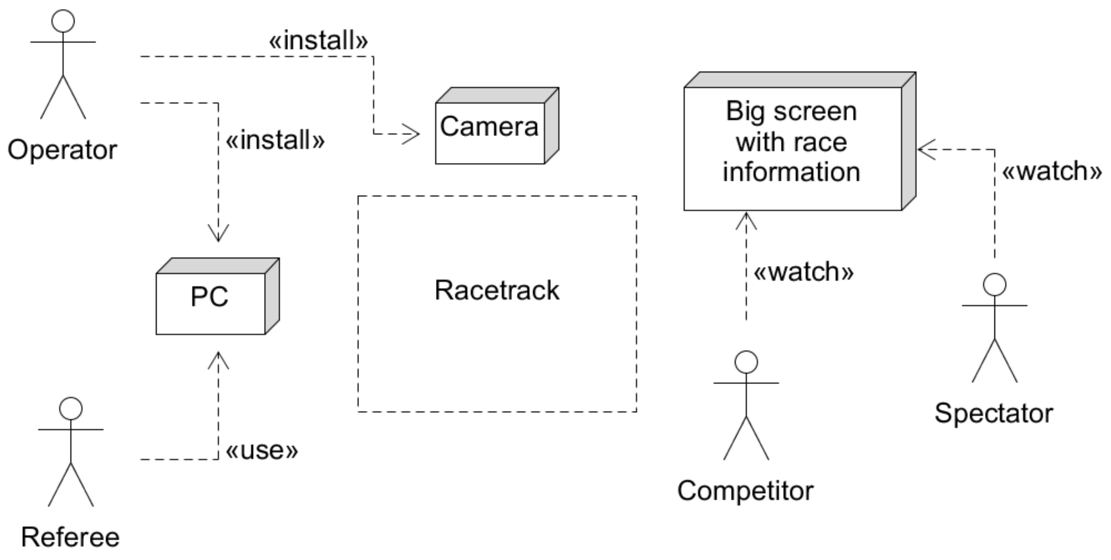


Figure 29: Actors and main system components interactions

The two main sides of this project are well reflected on these diagrams, some actors directly interact with the system to control it, manage the race while others only want access to the information produced.

3.3 SEPARATION

Following discussions with the "client" professors it appeared clear that it would be nice to have information displayed on a webpage.
Therefore, the choice is made to build an architecture with a separate data processing unit (the back-end part) working with a web server that will display information (the front-end part).

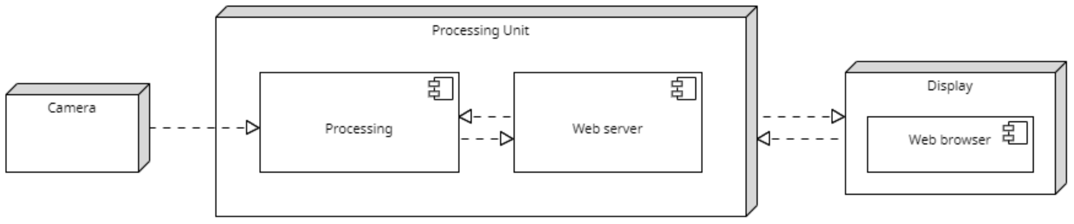


Figure 30: Minimalistic components architecture

This design allows sub-systems to be developed and tested independently.

3.4 Parts Role

Now that the system is divided into two main parts, it is time to define each one's role and tasks as well as the data needed and produced for each block.

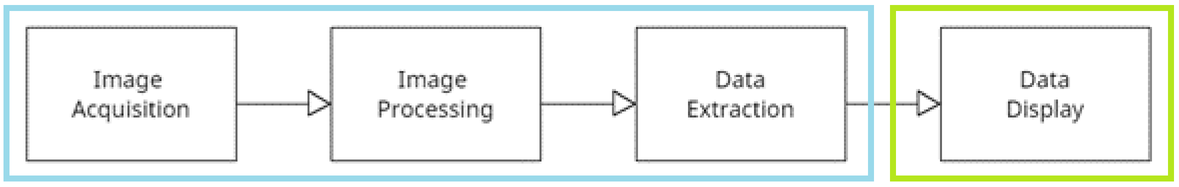


Figure 31: Sub-task repartition with blue for back-end and green for front-end

3.4.1 Front End

The main goal and purpose of the front-end is to provide a user-friendly and interactive interface for users to access, visualize, and interact with the processed data acquired from the back end. It should facilitate the system management so anyone even if not familiar with the project can understand and use the system during the race.

Displaying data will involve a web server that hosts a web page. The content of the web page is established based on the discussions with the "client".

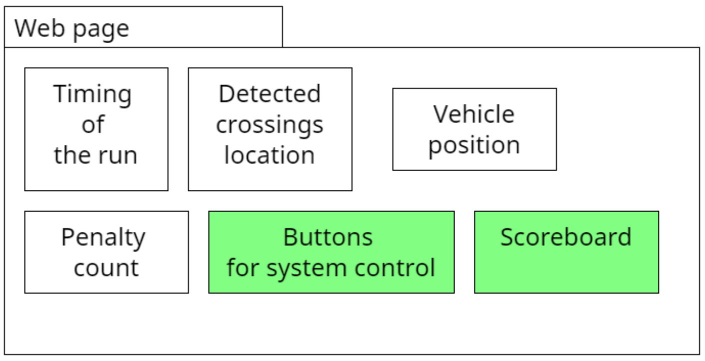


Figure 32: Content of the webpage with the nice-to-have ones in green

3.4.2 Back End

Most of the information displayed by the front end, is produced by the back end. Therefore, its role is to acquire images from the camera, to process them and use the different techniques tested previously to extract data out of them.

The next section discusses what data are exchanged between these two parts.

3.5 DATA EXCHANGE

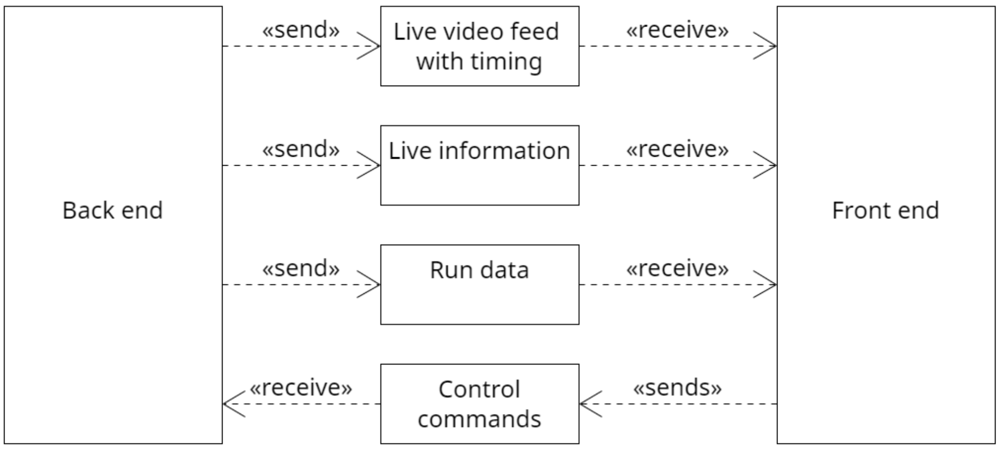


Figure 33: Representation of the data exchanged between the front and back end

By transmitting a live video feed, it enables to have multiple elements exchanged in one transaction. A frame contains the live position of the vehicle, the area where crossings have been detected can be superimposed on it as well as the timing.

Live information contains data about the current run, like penalty count, run status, participant id, etc.

The run data element is a summary of all useful information about a run and will be transmitted at the end of the run to be saved. This allows to have an history of all recorded run that can be displayed in the scoreboard.

3.6 *MOCK-UP*

3.6.1 THE NEED FOR A MOCK-UP

As the system's installation will not be possible before a few days prior of the race, it is necessary to design a way of testing the system with similar conditions.

The goal is to test the behaviour of the system using a real camera. It will show problems related to lighting conditions, shading, camera noise, etc.

A board with a top-down view of the track printed on it was ordered. Everything is at 1/10 scale.

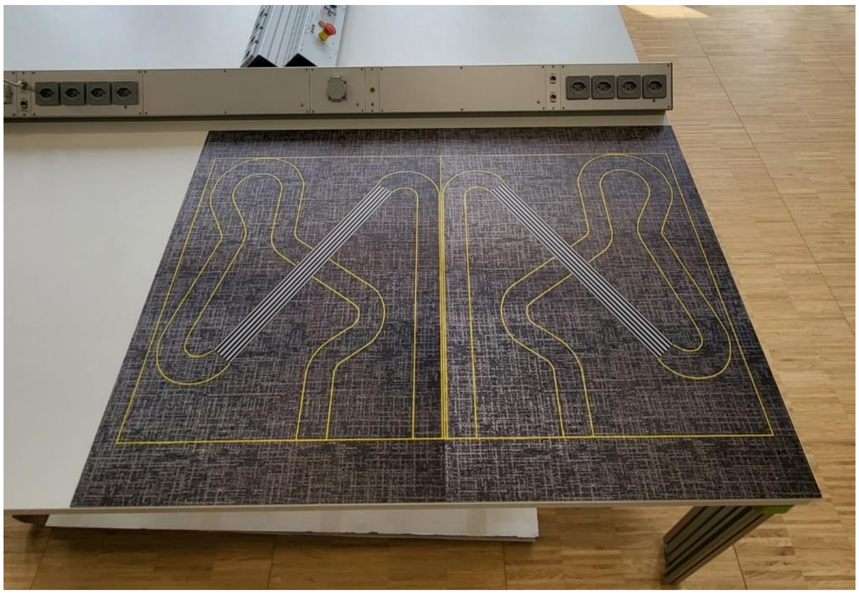


Figure 34 : Printed board with racetrack

3.6.2 FRAME AND CAMERA SUPPORT

The goal is to have a mock-up that is easily transportable and reusable for presentation or promotion.

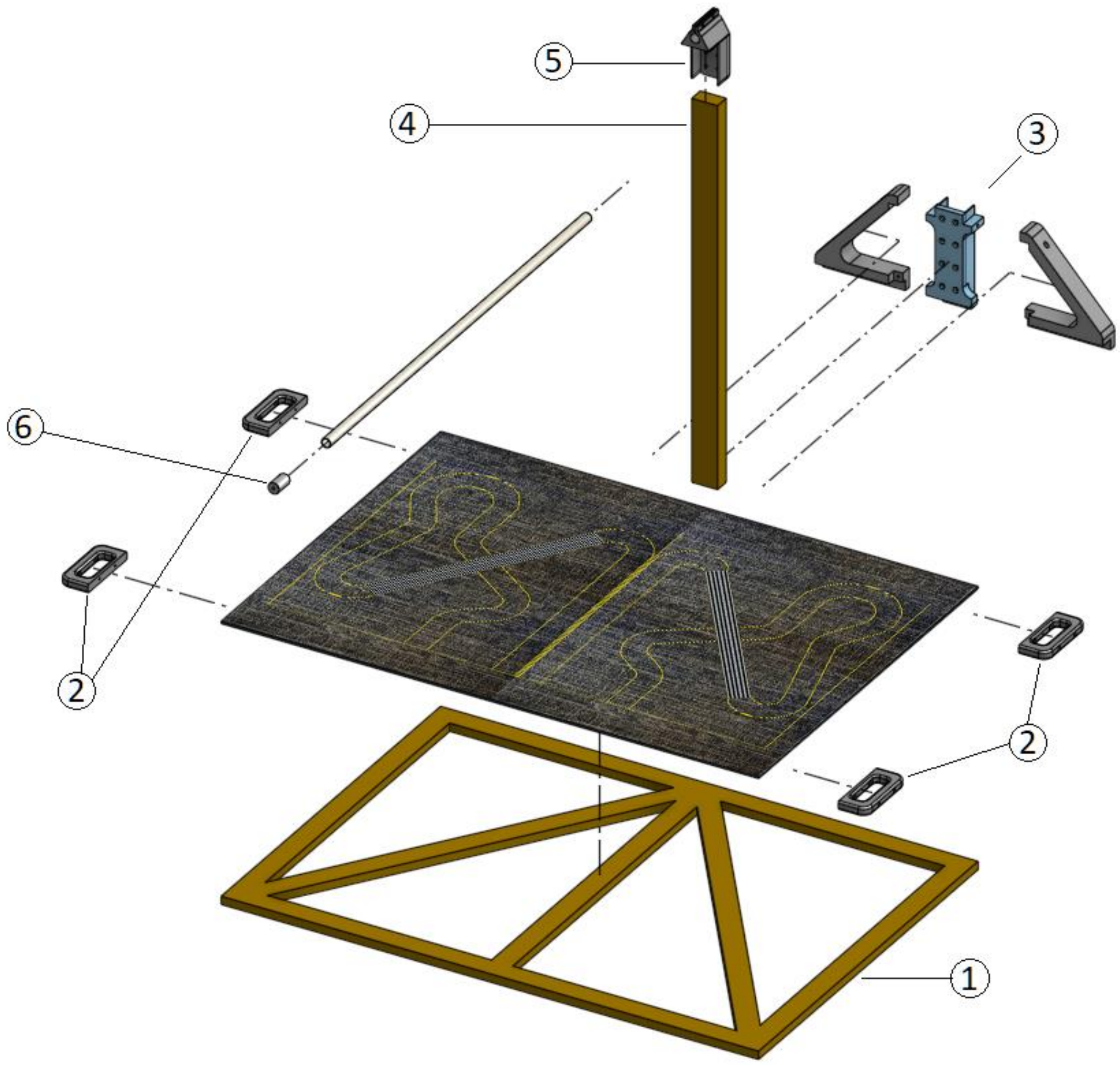
There are two main issues to address: rigidifying the panel and building a camera mount.
As the printed board is not stiff enough on itself, a wooden frame is placed underneath.
Handles are mounted on the side of the frame allowing to move the mock-up easily.
The camera will be mounted at the end of a steel tube that can slide on top of a wooden pole connected to the frame.

Most of the parts are be 3D printed.

The parts were designed using Onshape. Onshape is a web-based CAD platform that provides accessibility and version control. It allows multiple users to create, edit, and share 3D models and drawings in real time, eliminating the need for complex installations or local file storage.

Here is a link to access the Onshape documents with all parts of this project.
https://tinyurl.com/rtlcdCAD

3.6.3 EXPLODED VIEW AND PARTS DESCRIPTION



1. **Wooden frame:** is made of 18mm x 47mm lumbers. Its dimensions match those of the printed board.
2. **Handles:** are 3D printed and have two holes so they can be screwed directly on the side of the wooden frame.
3. **Base:** is 3D printed and will connect the pole to the frame. As the first design was too big to fit on the bed of the printer, the second is made of three parts. The two side arms are fixed to the middle part using screws and threaded inserts. The pole and frame are secured using wood screws.
4. **Pole:** is a 33mm x 57mm lumber. It will be cut to the right size depending on what height the camera must be positioned.
5. **Tube-Pole Connector:** This is a 3D printed part that joins the steel tube and the pole together. It also enables the tube to slide back and forth so the camera position can be adjusted. Once the camera is set the tube can be locked in place using two bolts and butterfly nuts.
6. **End cap:** is 3D printed and will be placed at the end of the steel tube. The camera will be screwed on it.

3.6.4 OVERVIEW

Here is what the complete mock-up drawing looks like and how it turned out.

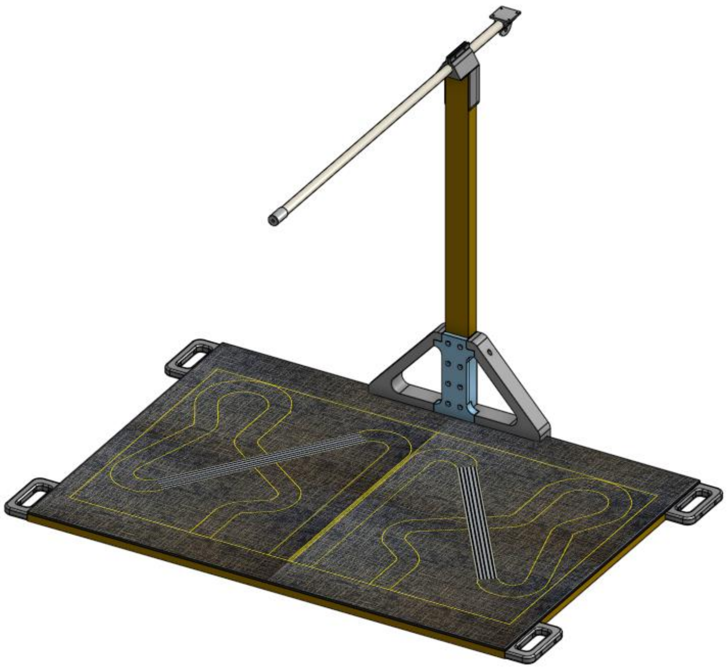


Figure 35 : Virtual overview of the mock-up

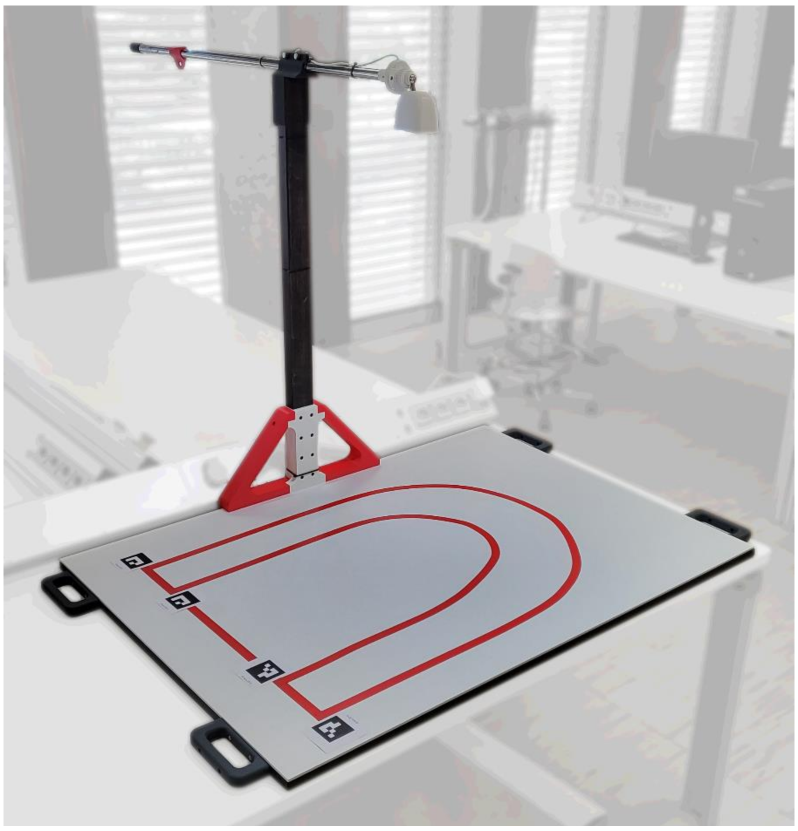


Figure 36: Overview of the mock-up built

3.6.5 TEST VEHICLE

Since the mock-up is at 1/10 scale, it was pretty hard to find a RC car small enough. This Tiny Revell Mini Car was bought.

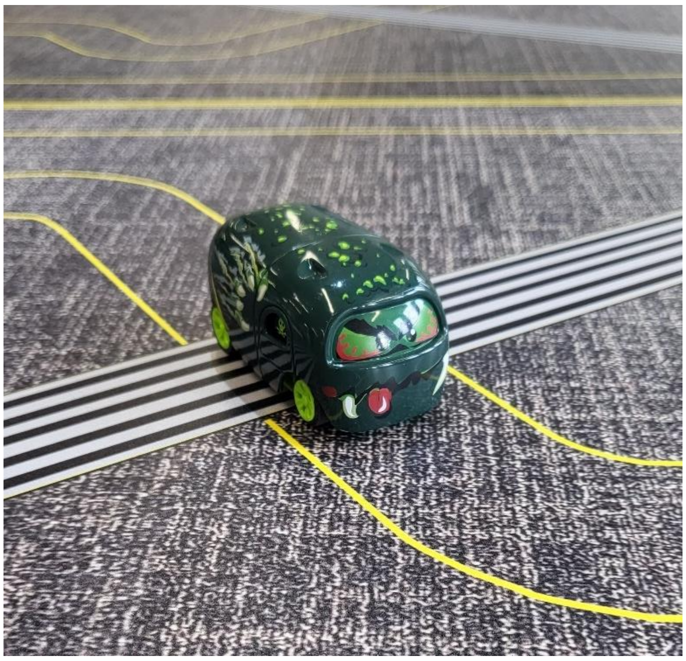


Figure 37: Revell Mini Car

It fits inside the track but it is slightly too big and its turn radius does not allow it to stay inside the limits during turns. Even if this mini car does not permit to simulate an entire run, it still allows to do some basic tests.

4 THE PROTOTYPE

4.1 PROTOTYPE CONCEPT

The goal is to develop a prototype to validate the design choices made during the initial conception phase. As a first prototype, it will focus on essential features proofing while the final proof-of-concept will focus on the adaptations needed to use the system during the actual race.

4.2 BACK-END

The back-end part is a Python project that consists of the main.py script and a JSON file to save different parameters values. It was first developed as a standalone app without front end communications. The changes made for it to work together are described in the section 4.3.7-Back-end modifications.

Here's an overview of the different steps from image acquisition to extracted data.

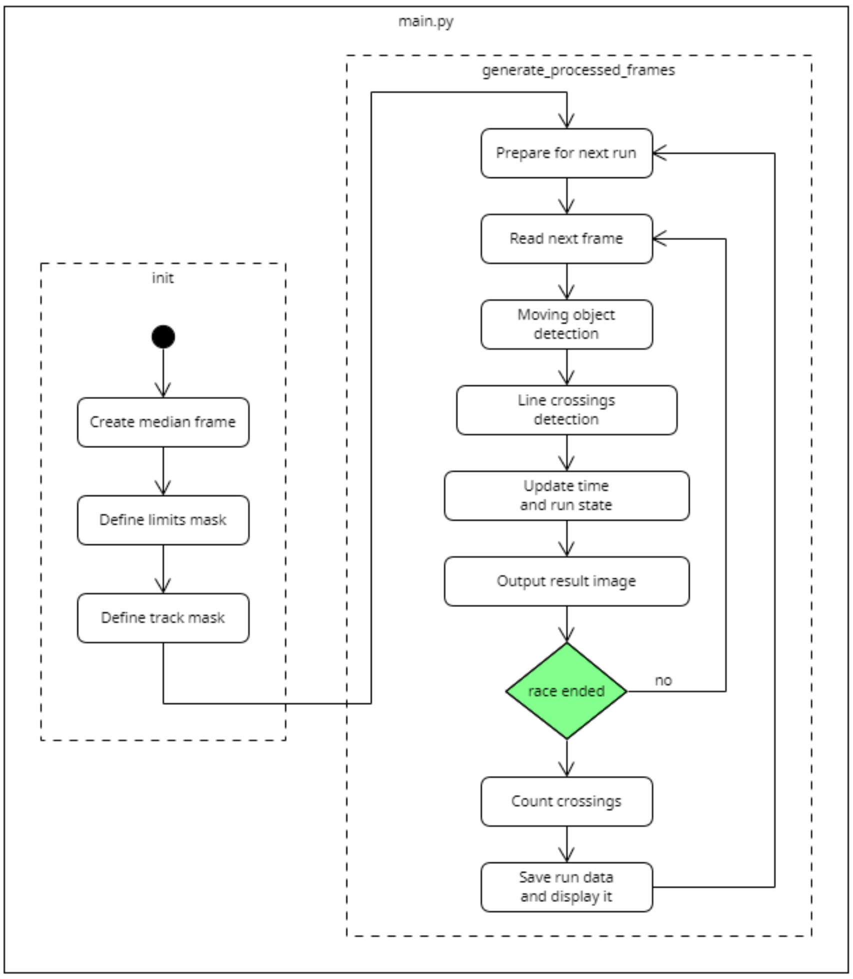


Figure 38: Activity diagram for the back end

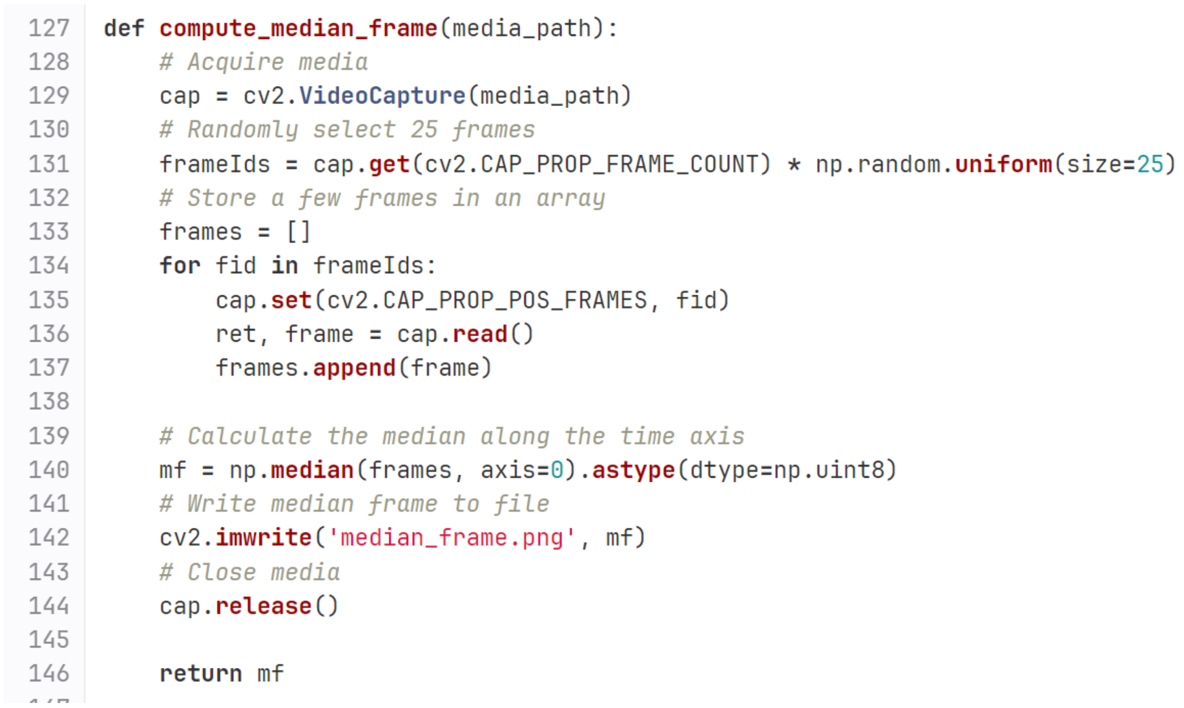
The next sections give more details about these steps.
More details on the project's configuration can be found in the Read.me file on the Gitlab repository:

https://gitlab.hevs.ch/SPL/bachelorthesis/2023-relicrode/rtlcd/

4.2.1 CREATING A MEDIAN FRAME

The chosen method to detect the car is the background subtraction technique and it requires a reference image.

To create the reference image, the script will collect a few frames from the video stream and then compute the median value for each pixel. This way the frame will contain only the static part of the image.



```python
def compute_median_frame(media_path):
    # Acquire media
    cap = cv2.VideoCapture(media_path)
    # Randomly select 25 frames
    frameIds = cap.get(cv2.CAP_PROP_FRAME_COUNT) * np.random.uniform(size=25)
    # Store a few frames in an array
    frames = []
    for fid in frameIds:
        cap.set(cv2.CAP_PROP_POS_FRAMES, fid)
        ret, frame = cap.read()
        frames.append(frame)

    # Calculate the median along the time axis
    mf = np.median(frames, axis=0).astype(dtype=np.uint8)
    # Write median frame to file
    cv2.imwrite('median_frame.png', mf)
    # Close media
    cap.release()

    return mf
```

Figure 39: compute_median_frame function

4.2.2 Defining the limits mask

One of the key elements of this process is to detect and isolate the track limits on the image. This allows to create masks that will define the limits and the outside of the track.

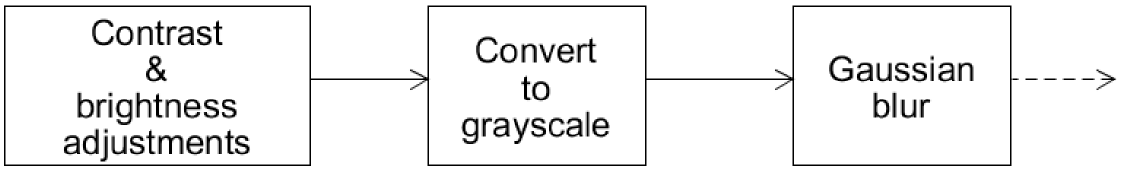


Figure 40: Process to create masks – Part 1

Using the median frame, it starts by applying contrast and brightness adjustments that help to make the track limits stand out more. Then it is converted to grayscale and a gaussian blur filter is applied to have better edges detection.

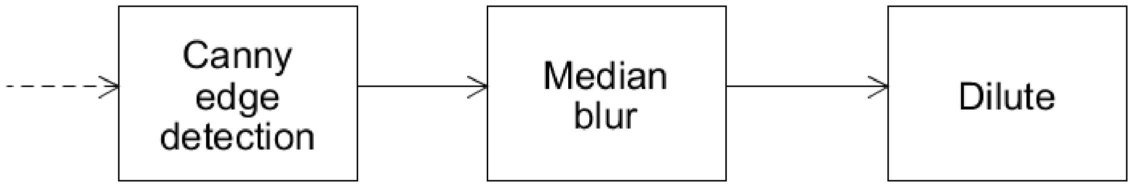
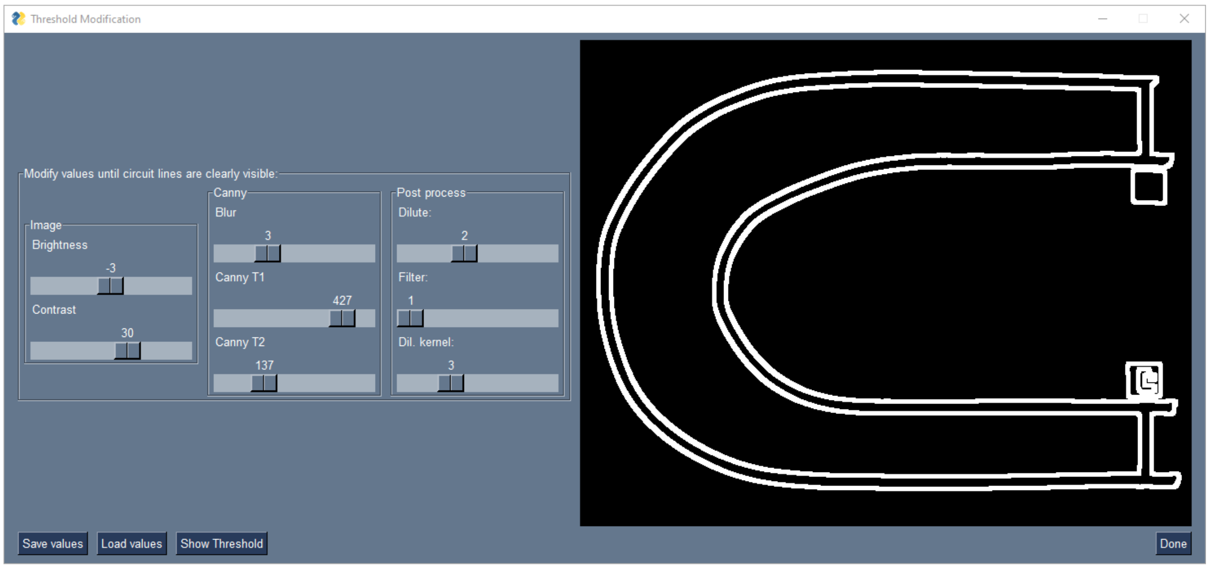


Figure 41: Process to create masks - Part 2

Then comes the Canny edge detection function that will output a binarized image of the detected edges. The median blur and dilute parts are some kind of post-processing to further modify the results if needed.

A graphical user interface (GUI) allows the user to tweak the settings used in those different steps until the limits are clearly defined on the image.



Modify values until circuit lines are clearly visible:

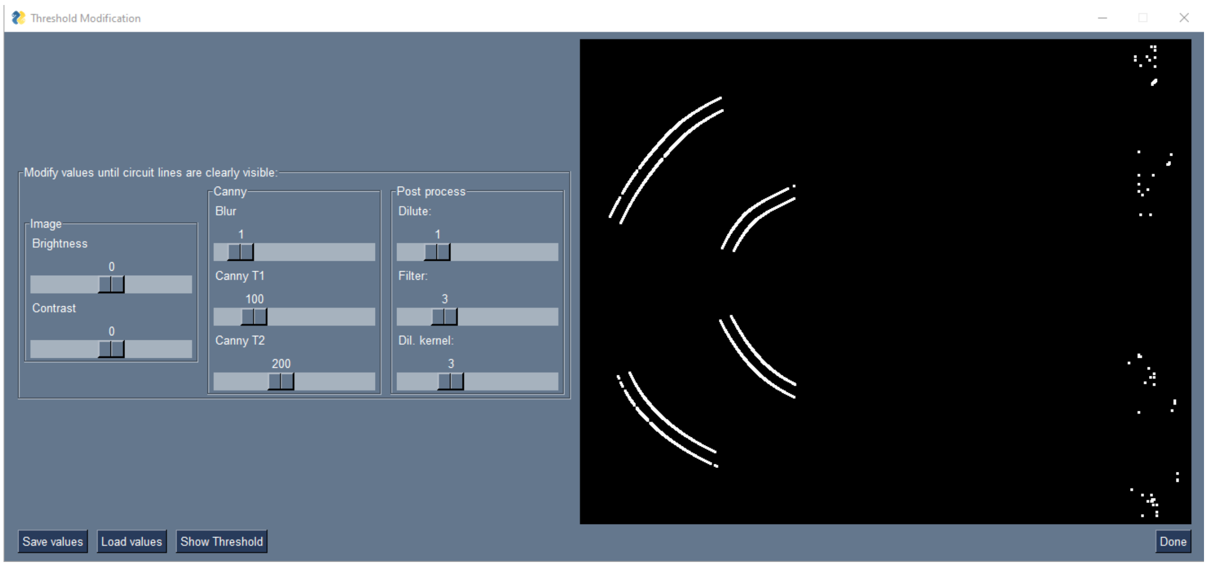
Image
Brightness -3
Contrast 30

Canny
Blur 3
Canny T1 427
Canny T2 137

Post process
Dilute: 2
Filter: 1
Dil. kernel: 3

Figure 42: Binarization values adjustments window

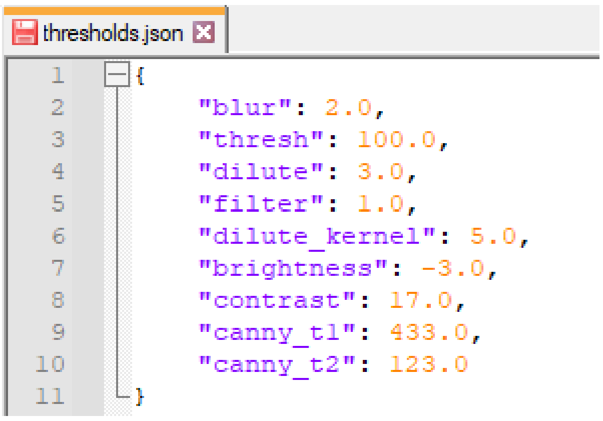
The so-called limits mask is now saved for further use. A good example can be seen in Figure 42, as well as a bad example in Figure 43.



Brightness 0
Contrast 0
Blur 1
Canny T1 100
Canny T2 200
Dilute: 1
Filter: 3
Dil. kernel: 3

Figure 43: Example of a bad values for binarization

The slider values can be saved in a JSON file and loaded the next time using the corresponding buttons.



```
{
    "blur": 2.0,
    "thresh": 100.0,
    "dilute": 3.0,
    "filter": 1.0,
    "dilute_kernel": 5.0,
    "brightness": -3.0,
    "contrast": 17.0,
    "canny_t1": 433.0,
    "canny_t2": 123.0
}
```

Figure 44: JSON file with saved values

4.2.3 DEFINING THE TRACK MASK

Based on the limits mask, the goal is now to segment the image to classify areas either as inside or outside the track.

The base idea is that the track has the same width all along its path (note that this is the case for the real race track but not for the test track used as example in this section) Therefore, by finding the contours that have the same width, it can be assumed they belong to track and can be classified as inside.

To do so, the code detects contours on the binary image using the cv2.findContours function. Then it asks the user to draw a box inside the track to approximate the track's width based on the box dimensions. Then for each contour, the distances of points in the image to the detected contours are computed. The furthest point to a contour defines the center of its inscribed circle which radius corresponds to the width of the contour.

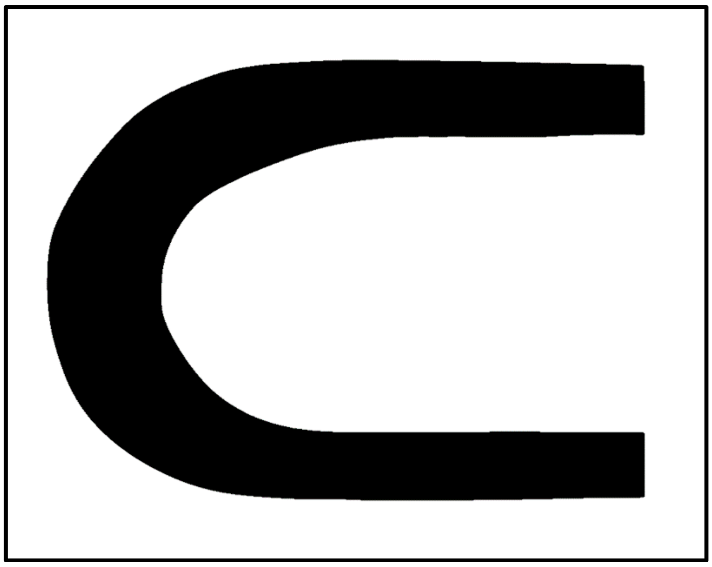


Figure 45: Segmentation of the test track

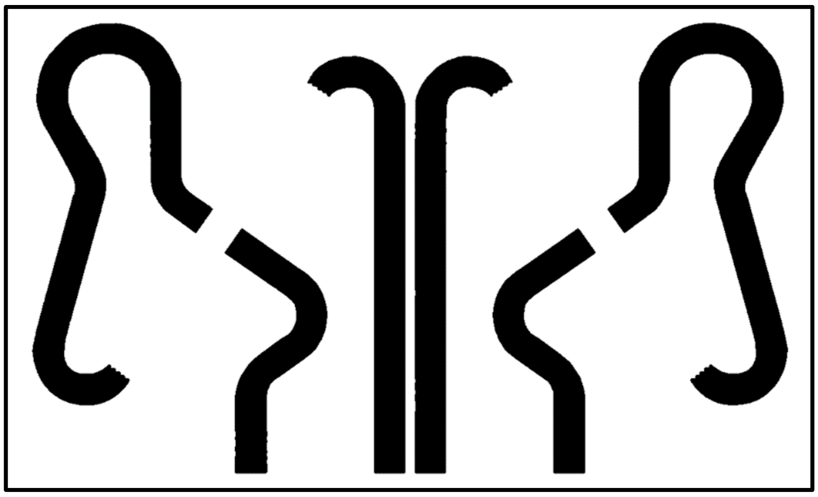


Figure 46: Segmentation of the actual track

The results show that this technique works well with complex track shape and can work too even if track width is not perfectly the same all track long.

4.2.4 Moving object detection

The background subtraction technique works by taking the current frame and highlighting all pixels that are different from the reference frame. As the background will not have moved, only the moving object's pixel will be highlighted.

Technically it is quite simple as it only requires computing the absolute difference between the two images pixels values. This results in a gradient map where the brightest spots are the pixels that changed the most.

Then, by applying a binarization using a customizable threshold value, the moving objects can be clearly defined.

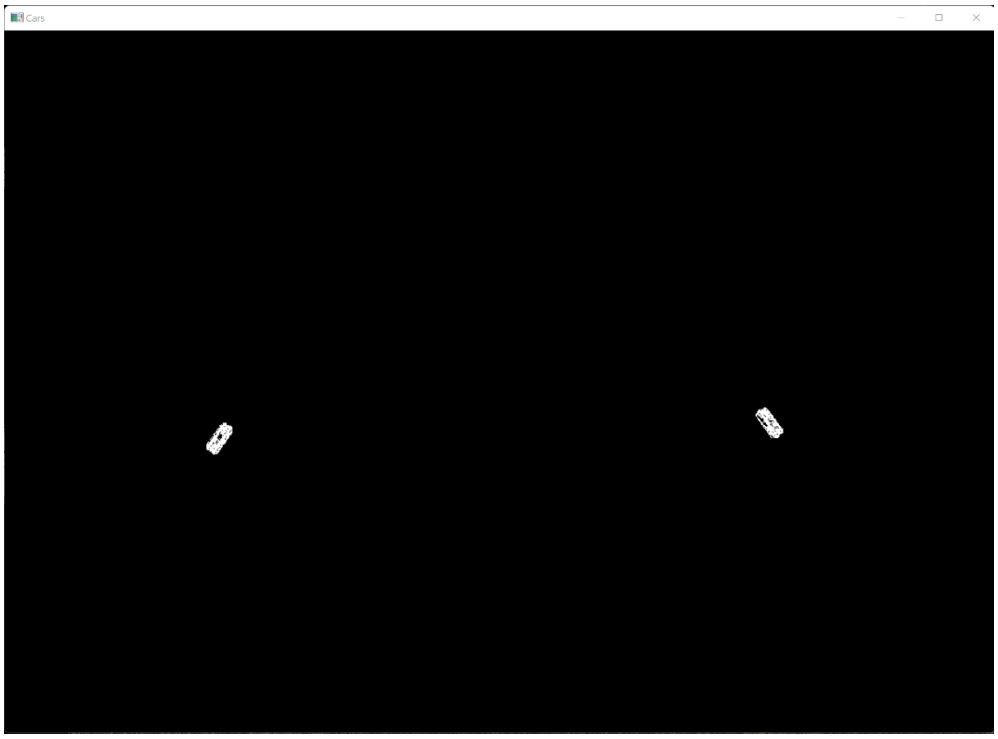


Figure 47 : Detection of moving objects, the two cars in this case

With the virtual video used for the tests the results are good as there is no lights effect or noise in the image. The results with a real camera feed were not good at first. The solution was to eliminate groups of pixels that had a too small area.

4.2.5 LINE CROSSING DETECTION

Now that the circuits contour and the moving objects have been detected, it requires only a bitwise AND mask to reveal where the line has been crossed. On the test video used, the car on the left stays inside the limits. The one on the right crosses it on multiples occasions.

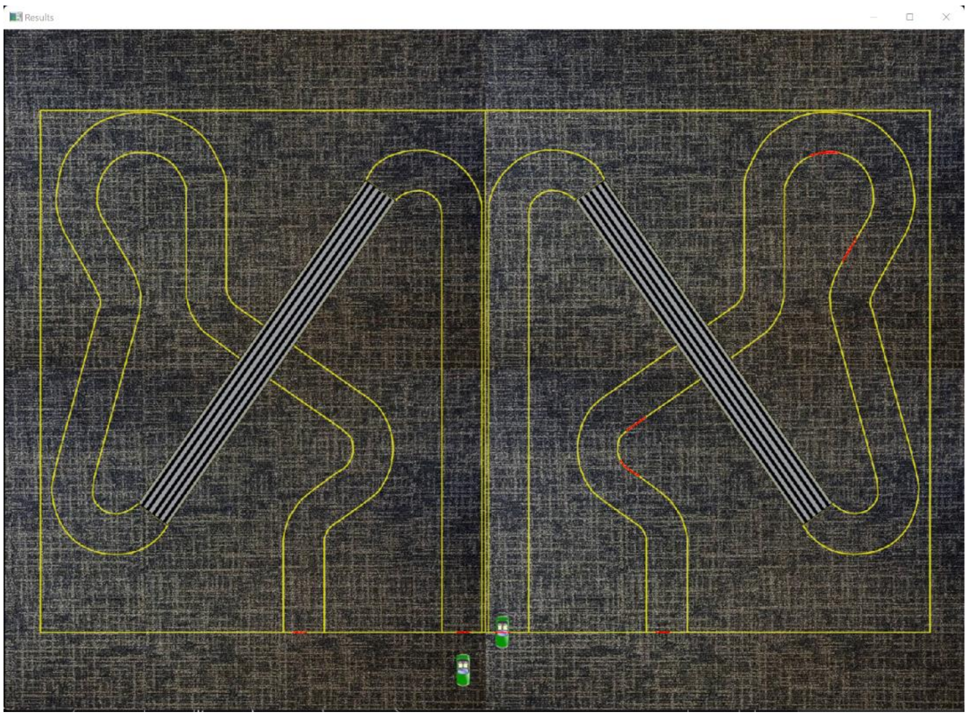


Figure 48 : Limits crossing highlighted in red

The advantage of this method is that except the reference frame and contours detection computing that are done in advance, the operations done in real time are simple. They mostly use B/W images and bitwise operations that are fast.

Now that the borders crossings are detected, there is still the need to extract useful information out of it. This is when the blob detector comes into play. By setting the right detection parameters it allows to count the number of crossings.

4.2.6 RACE TIMING

The goal is to display a stopwatch as the car runs down the track. It will require the detection of the start and finish line crossing.

The way it's been achieved during testing is by manually selecting the start and finish zone by drawing a box around it. Then, a blob detection is executed exclusively in this zone. At first, it assumed that if a blob is detected that means the car has crossed the start line and when a second one is detected, it means it has crossed the finish line.

The time is computed using Python's time library and displayed with OpenCV.

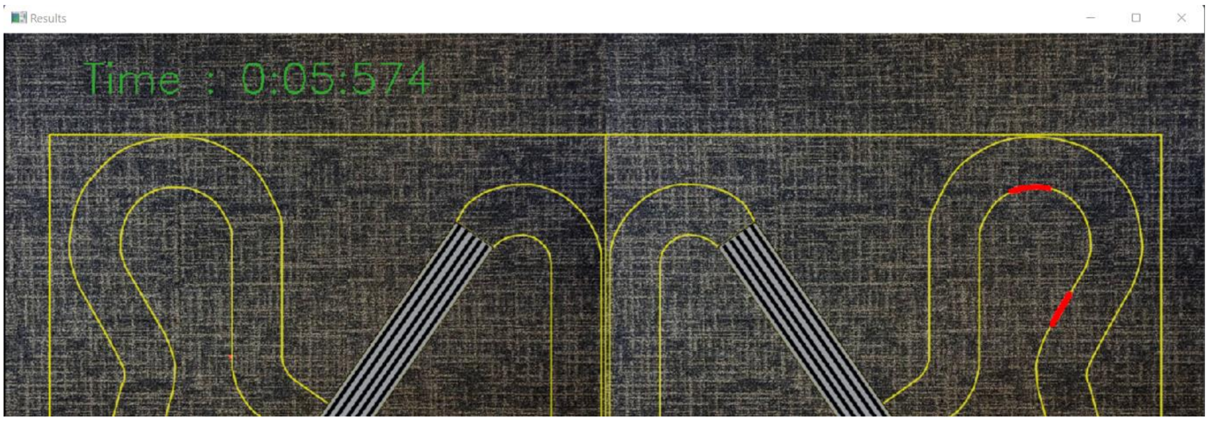


Figure 49 : Time indication displayed in top left and

By the end of the project, this part has evolved quite a lot. First the start and finish zone have been split up and previously the time was computed by saving the time at the start and at the end of the run with the time.time_ns function. It meant the time was depending on the execution time for all the frames. This problem had noticeable effects.

The way chosen to solve this issue is to use the cap.get(cv2.CAP_PROP_POS_MSEC) OpenCV function that returns the time (in milliseconds) elapsed since the opening of the media file.

$$time\ [ms] = \frac{frame_{index}}{frame_{rate}} * 1000$$

This way the time measured is directly linked to the frames and is not affected by execution time. The measurement precision is directly defined by the framerate.

Framerate [FPS]	Maximum achievable precision [s]	Maximum achievable precision [ms]
15	0.067	67
24	0.042	42
30	0.033	33
60	0.017	17
120	0.008	8

Any standard frame rate over 15 fps can provide half a tenth precision. If the need to go under a hundredth or a thousandth of as second presents itself, that would require using a high-speed camera providing a lot more frames per seconds.

A Timekeeper class has been developed to easily track and manage times intervals.

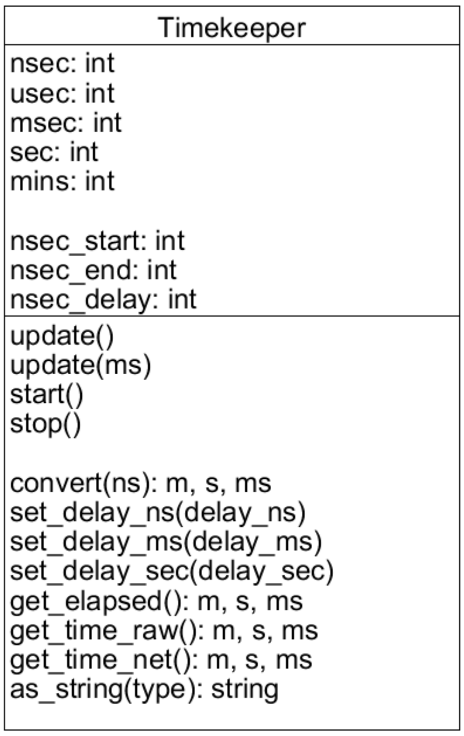


Figure 50: Timekeeper class UML diagram

This class works the following way, it stores a certain number of nanoseconds that needs to be updated with the provided methods, then it provides different methods such as retrieving the time interval in different formatting, starting, or stopping an interval. More specific to this application, is the possibility to add a delay to the interval, that translates to a penalty in a race context. The interval can then be retrieved with or without the penalty added.

The timing is displayed on the frame as a text and the information differs depending on the race state.

4.2.7 DETECTING START & FINISH LINE

To automatically detect where the start and finish lines are located, ArUco markers are used.

The start line is marked out by two markers with ID 0 and the finish with ID 1.

The markers are placed at both extremity of the line just as shown in the Figure 51.

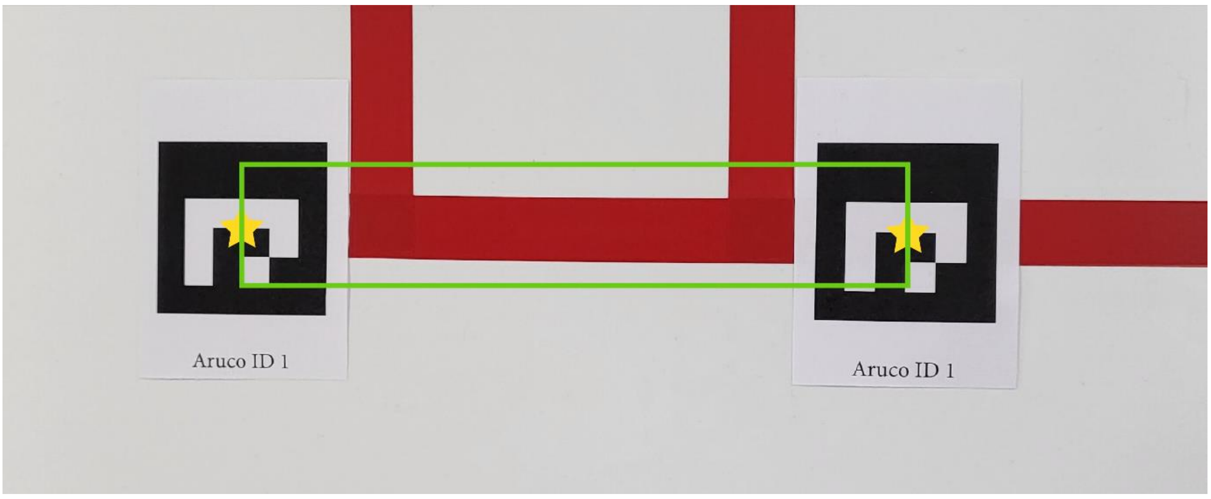


Figure 51: Finish line marked with ArUco markers

The function that detects the markers return their ID and the four corners' coordinates. The code computes their average to find the middle (shown in Figure 51 as yellow stars). Then, it detects the orientation axis of the line by comparing the coordinates difference between the two points. If delta X is smaller than delta Y, then it assumes the line is placed along the Y axis.

The line's dimensions are defined by the length between the two markers' center and by a defined pixel width.

It then prepares the data needed to crop the original image into two parts where the blob detection will be used.

If one of the markers pair is not detected, the script will prompt a GUI that asks the user to manually select the start or finish line.

4.2.8 STATE MACHINE

At first the different states of the system were managed by a pseudo state machine using if statements. This method was not ideal but served to prove the concepts. The creation of the RaceMgmt class that acts as a proper state machine, makes managing the different states a lot easier and the code much cleaner.

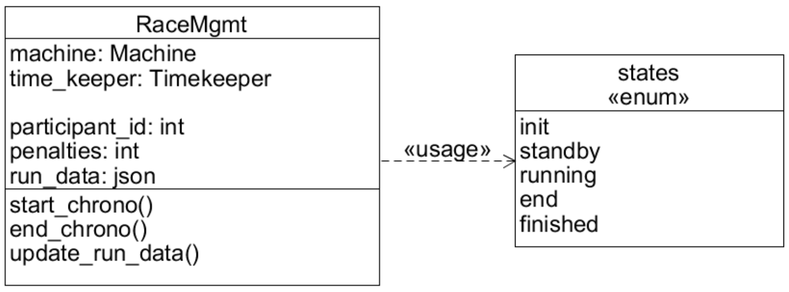


Figure 52: RaceMgmt's class UML diagram

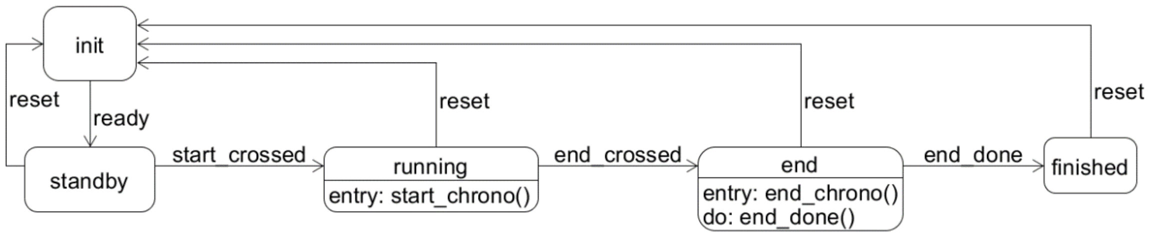


Figure 53: RaceMgmt's state machine UML diagram

4.3 FRONT-END

4.3.1 PURPOSE

In the appendices can be found a list of the different user interface ideas that were thought of. It starts with the most minimalistic one and ends with the “nice-to-have” more complex version.

This prototype focus on validating design and technologies choices, setting temporarily aside the separate use cases considerations, to produce a single web page with all elements to be tested.

4.3.2 TOOLS CHOICE

After searching online, here is a selection of web frameworks that exists.

a) FLASK

Flask is a lightweight micro web framework for Python. It provides the essentials to build a web application.

b) DJANGO

Django is a Python web framework that offers a lot of features for rapid development, including authentication, admin interface, database, etc. And it is ideal for projects that requires a full-featured framework.

c) FASTAPI

Fast API is a modern and high-performance web framework designed for building APIs.

d) TORNADO

Tornado is a scalable and non-blocking web framework made for handling real-time applications. It is particularly good in long-lived connections such as web sockets or streaming.

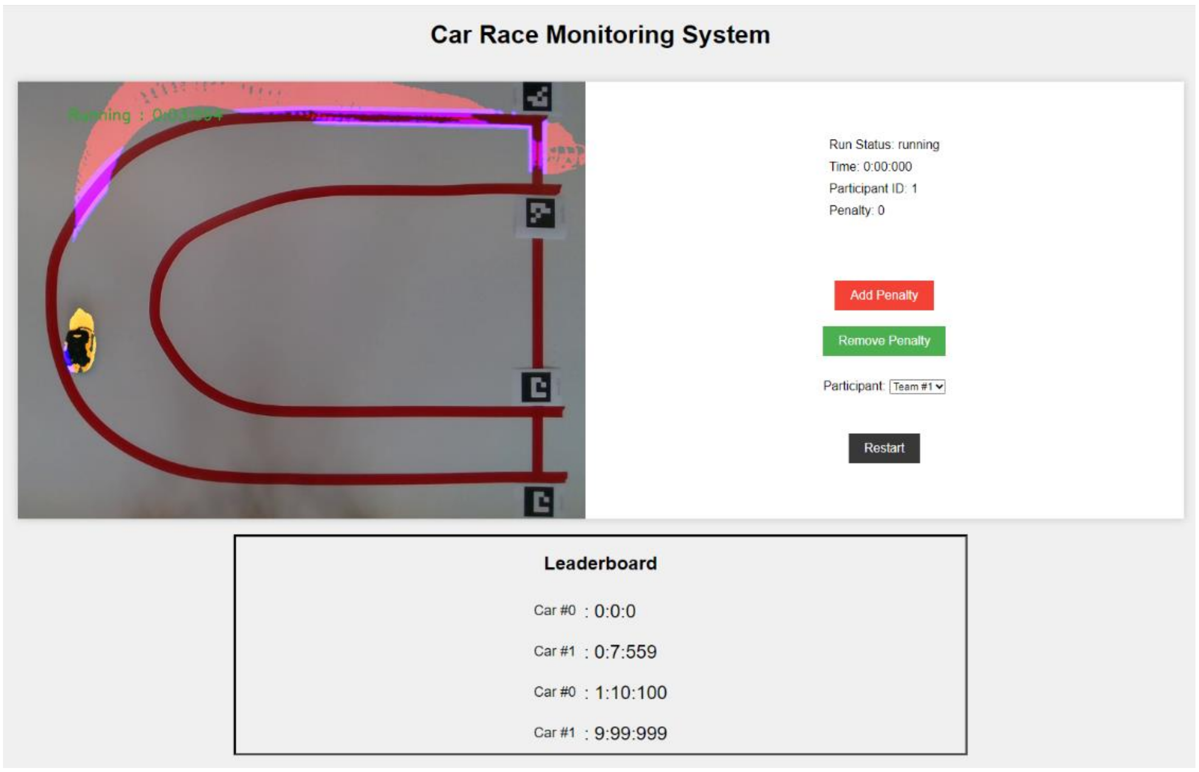
e) BOTTLE

Bottle is a minimalistic web framework focused on simplicity and it is designed for small scale application and prototyping.

Django was chosen as this project’s web framework. Without much experience in web development, it seemed that most of these frameworks would offers enough features for this project. The main reason in choosing Django was the fact it had extensive documentation and a lot of learning or troubleshooting resources.

4.3.3 Web page

Let's start with the end. Here is a screenshot of the web page created. Each part will be explained in the next sections.



Car Race Monitoring System

Run Status: running
Time: 0:00:000
Participant ID: 1
Penalty: 0

Leaderboard

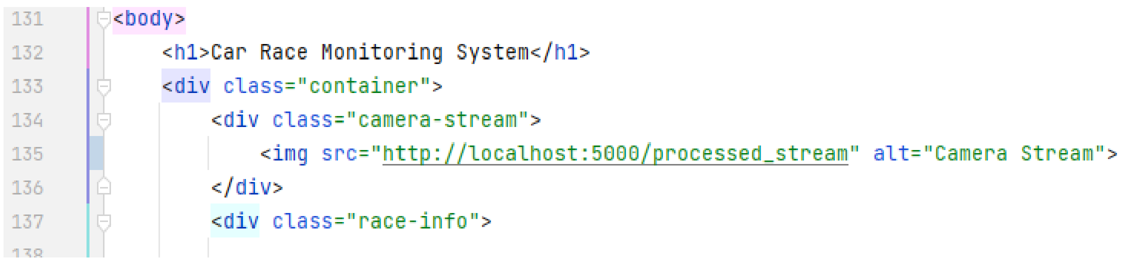
Car #0 : 0:0:0
Car #1 : 0:7:559
Car #0 : 1:10:100
Car #1 : 9:99:999

Figure 54: Prototype web page displaying race data

4.3.4 Video feed

This video feed offers a live view of the racetrack, showing the timing and where the vehicle drove off-track.

Those images are directly streamed from the back-end part. The back-end modifications are explained in the section 0 – Back-end modifications.



```html
<body>
    <h1>Car Race Monitoring System</h1>
    <div class="container">
        <div class="camera-stream">
            <img src="http://localhost:5000/processed_stream" alt="Camera Stream">
        </div>
        <div class="race-info">
```

4.3.5 Run data

This data is also directly streamed as JSON from the back-end part, and it shows information about the current run. When a run has ended, it is possible to modify the penalty count if needed and to start a new run.

4.3.6 Scoreboard

The scoreboard displays the participant ID and the time. These elements are pulled in increasing order directly from the database.

When creating a Django project, an SQLite database is automatically available. It will be use to store the store run data.

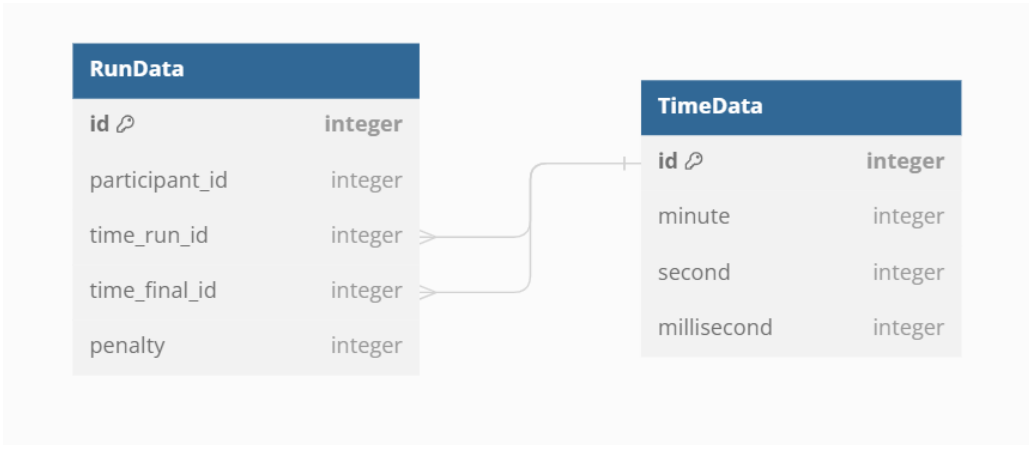
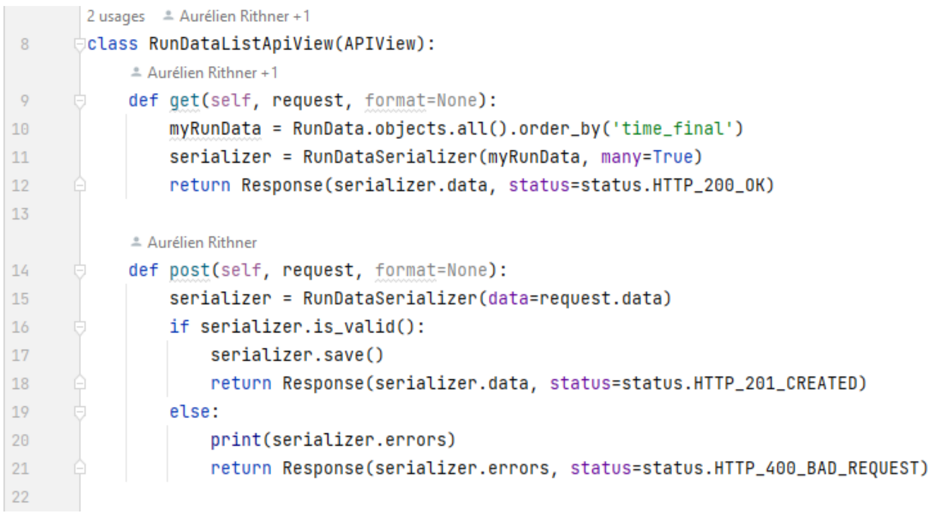


Figure 55: Database architecture

An API with GET and POST methods was created to enable adding elements in the database from the back end.



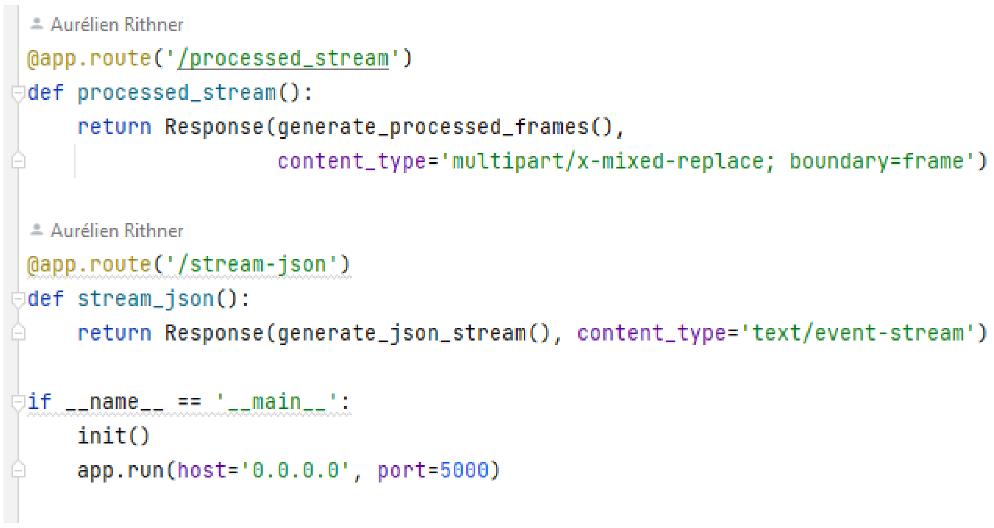
```
2 usages  Aurélien Rithner +1
class RunDataListApiView(APIView):
    Aurélien Rithner +1
    def get(self, request, format=None):
        myRunData = RunData.objects.all().order_by('time_final')
        serializer = RunDataSerializer(myRunData, many=True)
        return Response(serializer.data, status=status.HTTP_200_OK)

    Aurélien Rithner
    def post(self, request, format=None):
        serializer = RunDataSerializer(data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data, status=status.HTTP_201_CREATED)
        else:
            print(serializer.errors)
            return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
```

Figure 56: Extract from the API code

At the end of each run, the back-end part will push the run data to the database with a POST request.

4.3.7 BACK-END MODIFICATIONS

The back-end part of the prototype had to be modified to stream the video feed and the JSON file to the front end.



```
Aurélien Rithner
@app.route('/processed_stream')
def processed_stream():
    return Response(generate_processed_frames(),
                    content_type='multipart/x-mixed-replace; boundary=frame')

Aurélien Rithner
@app.route('/stream-json')
def stream_json():
    return Response(generate_json_stream(), content_type='text/event-stream')

if __name__ == '__main__':
    init()
    app.run(host='0.0.0.0', port=5000)
```

Figure 57: Flask implementation in the back end

For this task, Flask was used as it is fast to setup. It does not imply great modifications on the project structure and can effortlessly be added to the existing code.

4.3.8 TWO WAY COMMUNICATION

The different buttons or the drop list on the screen are not yet implemented and would require a two-way communication between the front and back-end parts.

Adding an API on the back-end part to trigger the start of a new run, or to inform it about the current participant will be the solution implemented in the final proof-of-concept.

4.4 RESULTS

Metrics	Goals	Results
Timing	Same time measurements as the previous system	To be tested at the actual race
F1 score	100%	No value
Mock-up	Robust and easily transportable	Works great to test the system Robust and easily transportable Camera mount and position easily adaptable
Adaptability	Tolerant to lightning changes Work with any circuit shape	Not dependant to a specific track shape Can adapt to environment easily with adjustable parameters in the GUI
Ease of use	Easily understandable results on a web page System parameters adjustable from web page	Simple web page with clear information No parameters control from web page implemented yet

Here is a table that sums up the prototype results based on the evaluation metrics previously defined in paragraph 2.6.

To evaluate the timing precision, the previous system using start and finish barriers is needed, hence the need to wait for the actual race.

The prototype proved it can successfully detect limits crossings, but not enough tests were made to calculate a relevant F_1 score.

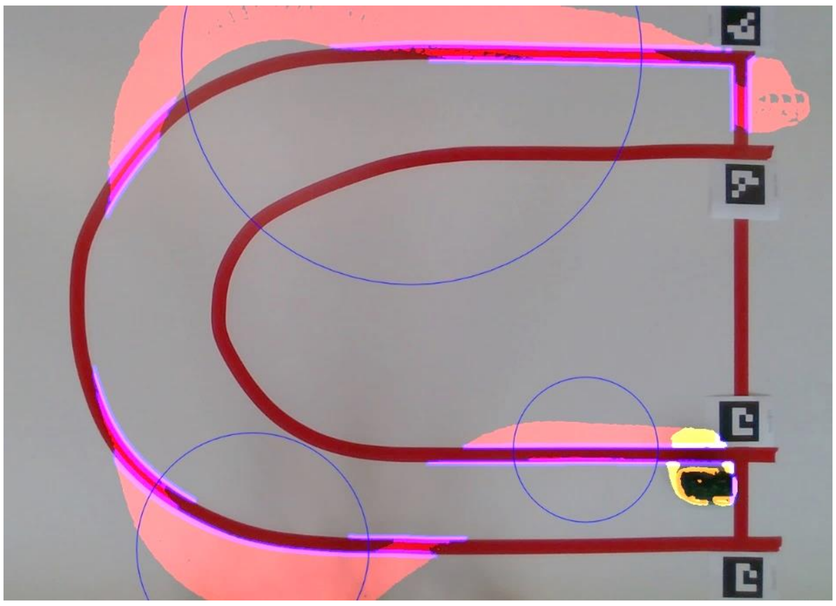


Figure 58: Result frame with the crossings detected

The overall features of the prototype work well but more testing time is needed to fully quantify the results.

Rather than focusing on perfect results outcome for a particular setup, energy was put into making the system flexible and easily adaptable. Therefore, the current results can be improved by tweaking the different parameters provided.

5 FINAL PROOF OF CONCEPT

5.1 ITS GOAL

The goal of this final proof-of-concept is to put to use the two parts developed during the prototyping phase during this year's summer school race.

5.2 CONCEPT & INSTALLATION

The prototype was developed so it can be used in the final proof-of-concept without much additional work except for the few modifications listed in the next section. Therefore, most of the work for this part resides in the physical installation and test phase planification.

The camera used will be the DLink DCS-8627LH. It will be mounted between the two concrete pillars using straps. The camera is configured on the school network and connects automatically to the appropriate Wi-Fi.

This allows for the PC used as processing unit and web server to be placed anywhere in the school where it has access to the network. Then any another device that can display an image on the big screens of the library and have access to the school local's network can be used to display the web page.

To facilitate the racetrack detection, the use of a green tape instead of the yellow wire is recommended.

In the previous races, black grip tapes were placed on the aluminium bridge. That creates a striped pattern that interferes with the racetrack markings detection. The solution would be to paint the bridge in black.

5.3 PRINCIPAL DIFFERENCES WITH THE PROTOTYPE

5.3.1 DIFFERENT WEB PAGES

The idea is to keep the same page already in place for the system operator and add another page that will only display the video feed and the scoreboard. This new page will be displayed on the big screen in the library for the spectators to watch.

5.3.2 WEB PAGE CONTROL

The current buttons of the web page are not functional yet. An API on the back-end side needs to be implemented so the buttons can work properly.

5.3.3 TRACKING OF THE VEHICLE

Interferences in the crossing detection caused by a person walking in the frame or some shadows, could be prevented by tracking the vehicle and applying the detection on a limited region of interest around it.

5.4 LIMITATIONS

One of the main problems that has been overlooked during this project, is the presence of a concrete pillar inside the racetrack perimeter. That means one portion of the circuit is not visible from the camera.

While the eventuality to use multiple cameras placed around the track was discussed and so the design of the system was kept modular, it was never implemented due to time constraints. For the moment, this part of the track will have to be manually monitored.

5.5 TESTING PHASE AND RESULTS

Some initial testing can be realized on the mock-up while waiting for the complete installation on the library, but the main tests will happen during the actual race.

Data needed to calculate the accuracy of the system will be gathered.
The results will be presented during the defence of this thesis.

6 MACHINE LEARNING

6.1 CONTEXT

If the drone solution had been chosen, another image processing technique should have been used. With an ever-moving image, background subtraction and pre-processing of track limits cannot be used. In this case training a model with machine learning becomes an interesting choice as the results should be more tolerant to environmental changes such as light, noise etc.

One of the most complex tasks to build a reliable model is to have a great dataset.
The dataset shall have coherent, useful, and representative data and be extensive. Especially for this application with high-resolution image, many images need to be generated.

To simplify this task the choice was made to create a virtual environment in Unity and use scripts to automatically render a lot of images.

6.2 DATASET CREATION

6.2.1 SETTING UP THE VIRTUAL ENVIRONMENT IN UNITY

After a few tutorial projects, a new scene was created for the virtual environment. The track surface was placed together with a small car. Different scripts were added to control the car manually with the arrow keys of the keyboard, move the camera etc.

6.2.2 RANDOM PLACING OF THE CAR

Once the scene was set, a script was created to place the car automatically and randomly on the track. For the tests purpose, only the rotation and position of the vehicle changes. But in future development to further improve the dataset other parameters like camera angle, camera offset, lights and shadows would need to be slightly changed randomly.

6.2.3 AUTOMATIC IMAGE CLASSIFICATION

The goal of the script is to generate hundreds of images with the car placed randomly on the track. But it would still take a lot of time to manually classify those images into the “inside” or “outside” categories.

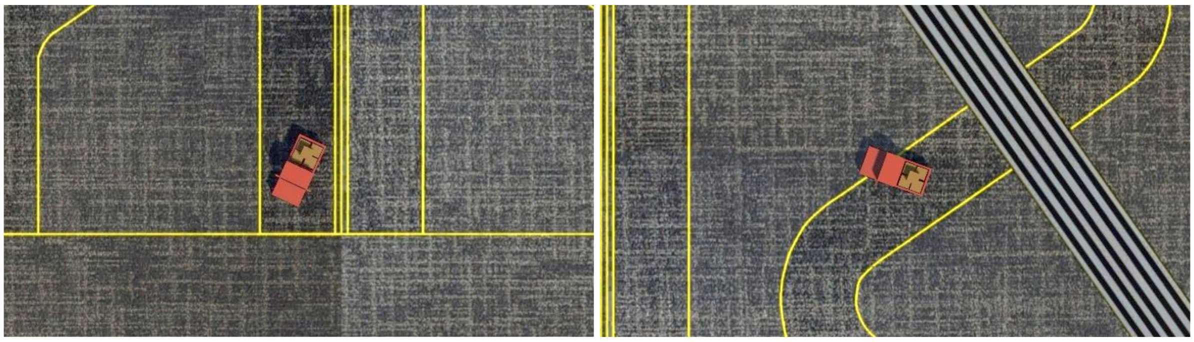


Figure 59: Inside and outside image generated

Since Unity is a game engine, it is relatively easy to implement objects collision detection. The idea is to have a transparent object that corresponds to the outside space of the track. When a collision is detected between the car and this object, Unity will know the car is outside of the track and can automatically classify the image generated.

This transparent object was created using Blender.

6.2.4 CONVEX MESH CHALLENGE

This collision idea detection seemed a good idea but revealed itself to be quite challenging for this application and with this much experience in Unity and Blender.

To detect a collision between objects, Unity needs to have a collider mesh for each one of them. It can automatically create one using simple shapes like boxes or spheres which are not precise enough in most cases, but it can also generate a polygon shaped mesh collider.

The problem is that Unity supports only convex mesh collider, so it won't work with our special shape that has concave spaces.

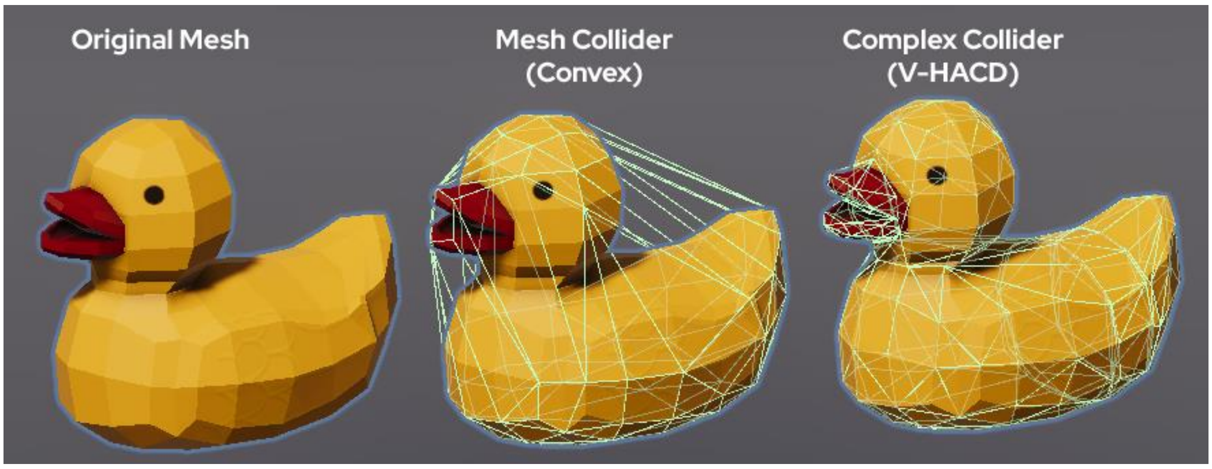


Figure 60: Convex VS Complex Mesh colliders [17]

The way to get around this issue is to decompose the original part into a lot of pieces that will have convex mesh colliders.

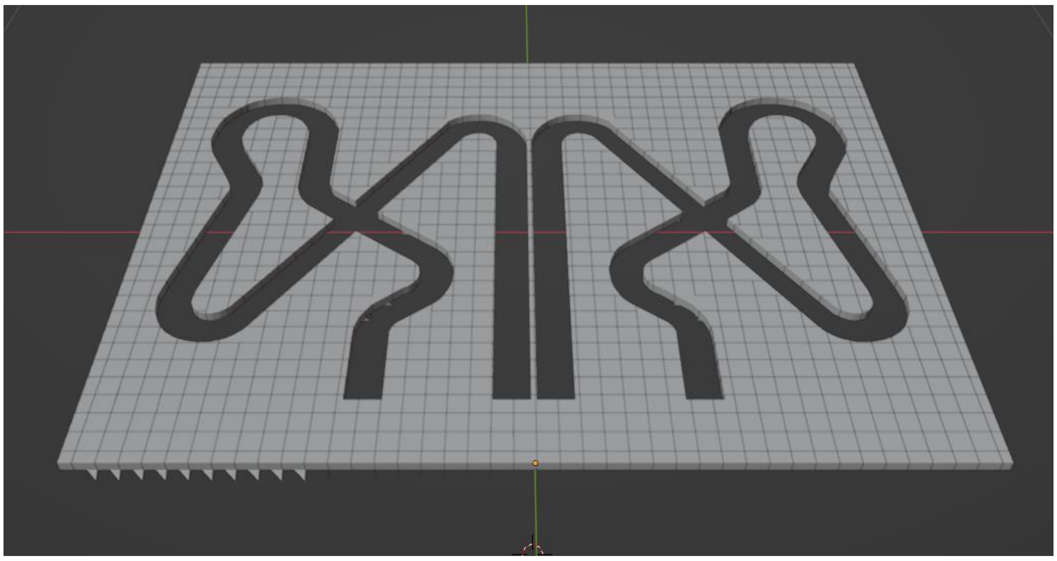


Figure 61: The "outside zone" object decomposed in Blender

This was done in Blender by creating a grid and then using the Boolean tool to cut the object.

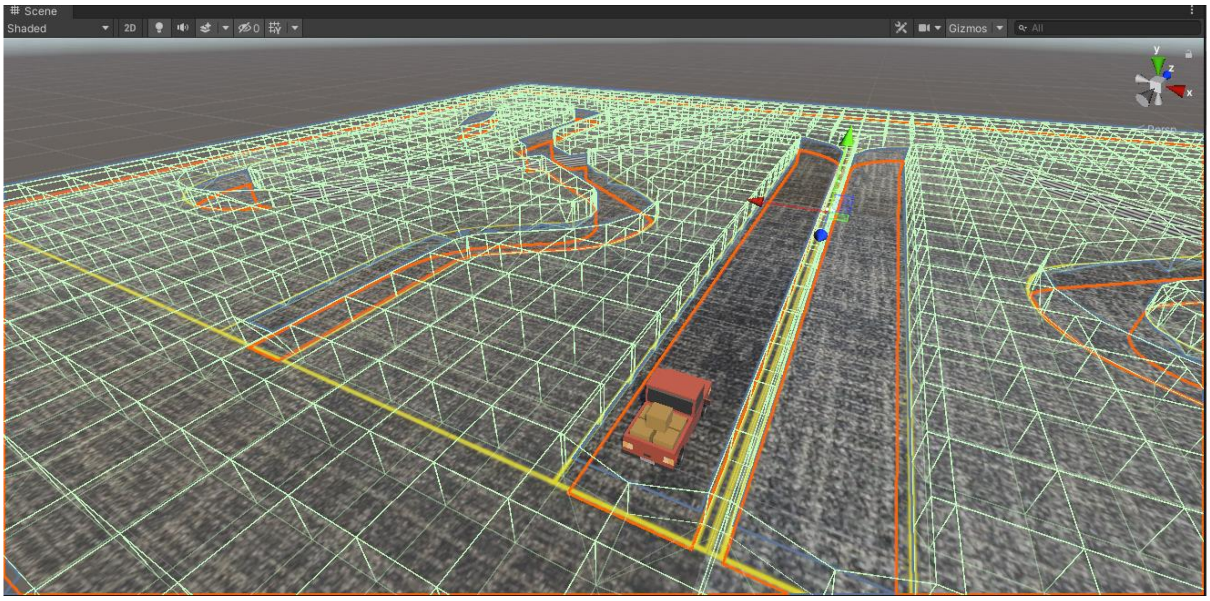


Figure 62: Colliders highlighted in Unity

Existing paying plugins are available on the Unity store and some open ones can be found on GitHub [17].

6.3 THE TRAINING

6.3.1 KAGGLE

After looking online for some examples and code samples, it was decided to use Python and Keras to write a script to train the model.

At first, the script was run in PyCharm on the lab computer, but the processing power was not enough to efficiently train models.

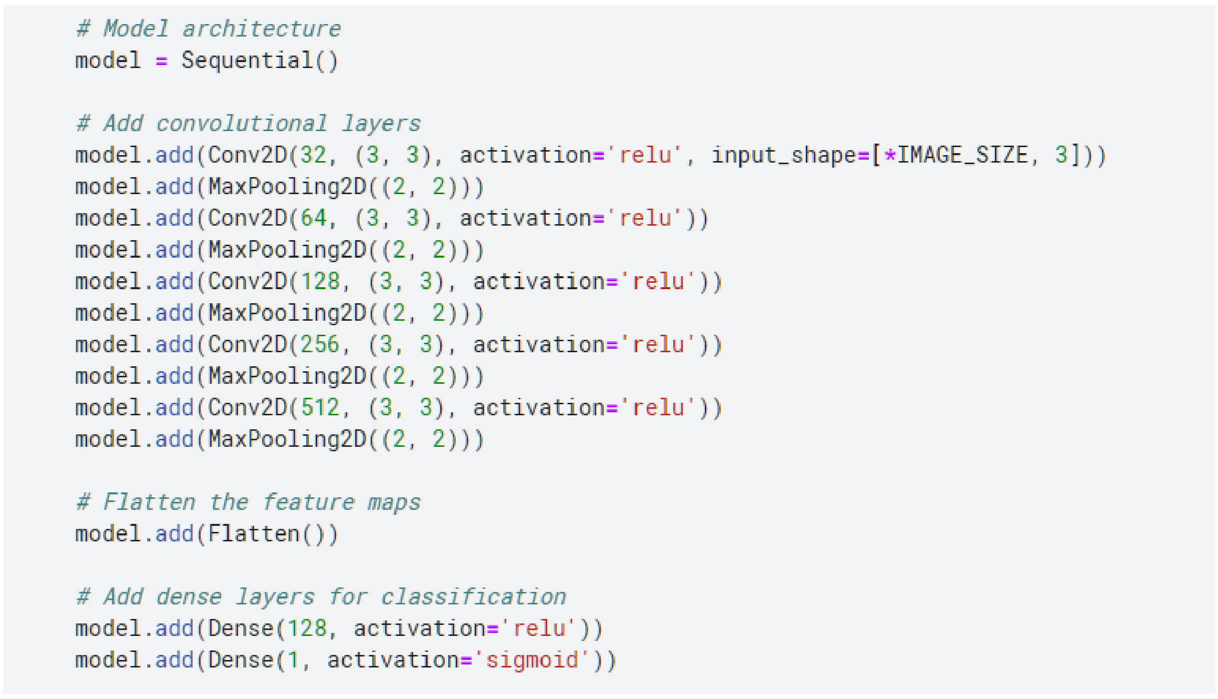
The alternative chosen was Kaggle. Kaggle is an online platform for collaborative machine learning projects. It is well known for the data science competition that are regularly held on the platform. Once a free account is created, users can create notebooks, test, and run their code. Once an account is verified, it gives access to accelerator such as GPU and TPU to help process data faster.

Here is a link to the Kaggle notebook created for this project:
https://www.kaggle.com/code/axalppro/rtlcd

6.3.2 MODEL ARCHITECTURE

As previously seen in the state-of-the-art, CNN models are the most suited to image recognition.

This architecture is based on standard examples found online and was tweaked with the help of a professor specialized in machine learning.



```
# Model architecture
model = Sequential()

# Add convolutional layers
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=[*IMAGE_SIZE, 3]))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(256, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(512, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))

# Flatten the feature maps
model.add(Flatten())

# Add dense layers for classification
model.add(Dense(128, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

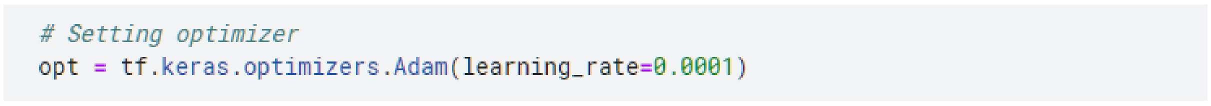
Figure 63: Model architecture definition using Keras

It is composed of five convolutional layers that will scan the image input to detect and highlight relevant features. The “ReLu” activation function will introduce non-linearity to improve the model’s ability to capture complex patterns.

The pooling layers help to reduce the spatial dimensions of the feature maps while retaining important information.

Then the feature maps are flattened into a one-dimensional vector to prepare data for processing in fully connected layers.

The dense layers will perform classification based on the extracted features. The final dense layer with a single unit and “sigmoid” activation will output a probability score that represent the probability of the input image to belong to a particular class.



```
# Setting optimizer
opt = tf.keras.optimizers.Adam(learning_rate=0.0001)
```

Figure 64: Optimizer settings

The chosen optimizer is “Adam” with a learning rate ten times smaller than default. This will help ensure a steady convergence during training.

6.3.3 ITERATIONS

After a few tests with mitigated results different modifications in the code were made.

f) BALANCED DATASET

Initially the dataset was not balanced with about 20% of the images in the “inside” category. This was due to the script correctly placing the car on the track randomly but since most of the area is considered “outside”, only a fraction of the images generated were “inside”. By balancing the dataset, it prevents the model to favour predictions to any category simply because this category is more present.

g) NUMBER OF EPOCHS

At first, limited by the lab computer processing power, the model was only trained on a very limited number or epochs, about fifteen. It appeared quite clear a lot more would be needed. Lately the model has been trained on about 80 epochs with learning curves looking a lot better. But some of them shows some signs of overfit, so it would be good to implement some early stopping to prevent that.

h) NUMBER OF CONVOLUTIONAL LAYERS

As the input image size is quite big, adding convolutional layers and max pooling layers enable reduce the quantity of information transmitted to the dense layers.

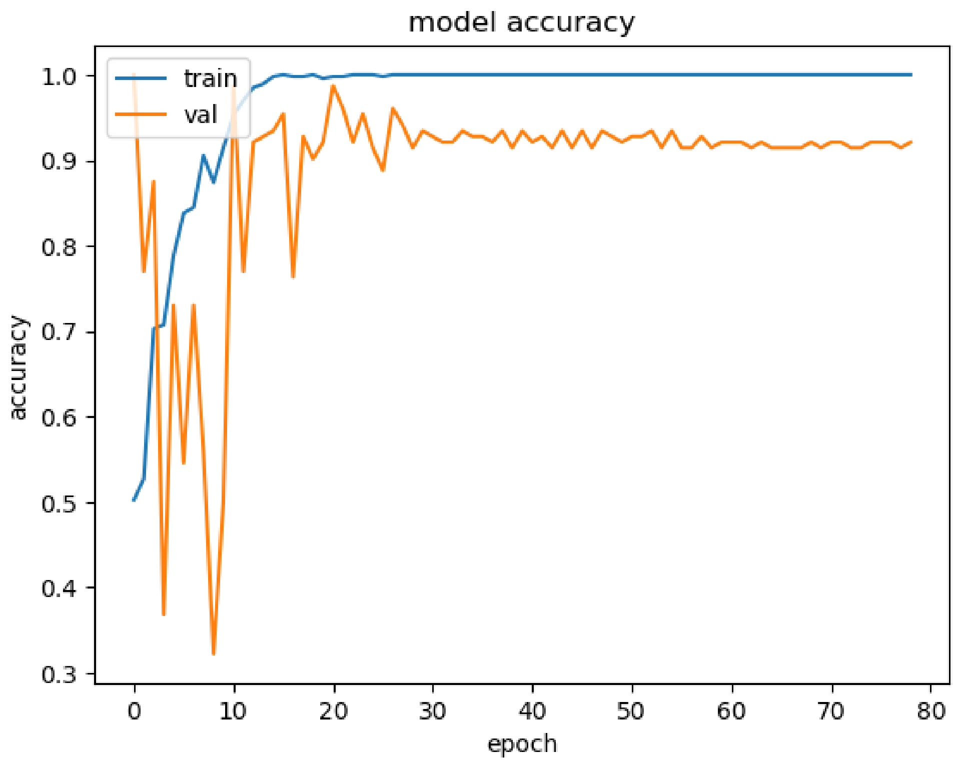
i) SIZE OF INPUT DATA

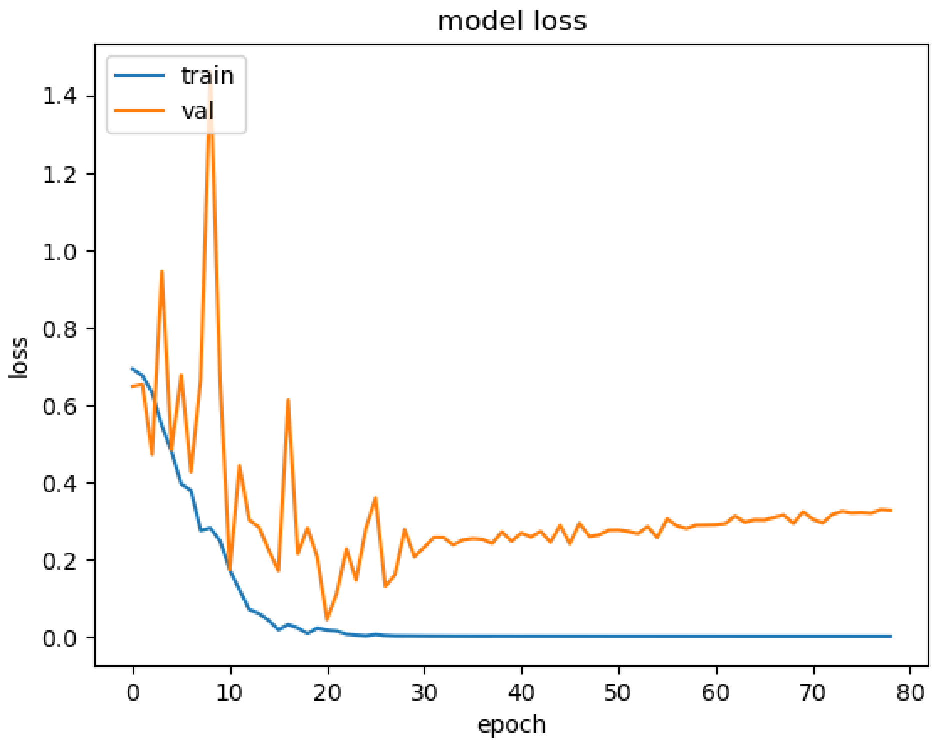
It’s better to show less data but more frequently than showing full resolution images not enough times. Therefore, it has been decided to reduce by five the resolution of input images.

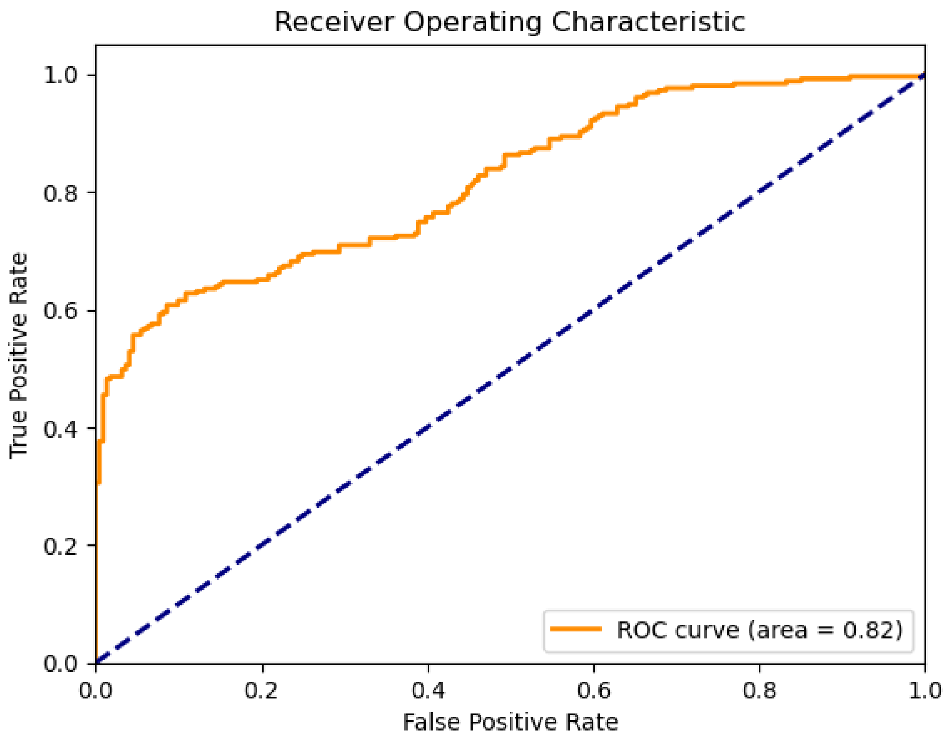
6.4 Results

6.4.1 Learning Curves

After a few iterations here are the results obtained.







As we can see on the loss learning curve the training curve is good with slow descent approaching 0.0 and then staying mostly flat but still slowly getting better. On the other hand, the validation curve seems more hectic at start and then stabilize around 0.3, what is problematic is it has a slight tendency to go up.

6.4.2 Evaluations

Here are the results of the model performance on the validation set.

	Precision	Reacall	F1-score	Support
inside	0.65	0.95	0.77	221
outside	0.92	0.56	0.7	255
Accuracy			0.74	476
Macro average	0.79	0.76	0.74	476
Weighted Average	0.8	0.74	0.73	476

As the dataset was balanced, the accuracy and f1-score metrics give a similar value. An accuracy of 74% would not be usable in this application but it is a good starting point.

6.4.3 CONCLUSION ON USING MACHINE LEARNING IN THIS PROJECT

Those results are encouraging but more time and knowledge would be needed to continue improving the model performance to make it usable for this application.

Further improvements could be adapting the code to truly use the accelerator power so it can train the model more extensively while not taking too much time. While being a simple notebook options to change, using the Tensor Processing Unit requires to follow a defined procedure. The Kaggle documentation is very explicit and detailed on this process [18].

Dataset augmentation could be a great way to give the model more material to work with. Varying the zoom, camera position, camera angle, lightning conditions, adding noise, as well as changing the car model are possibilities that would enhance the dataset size and quality.

7 EVALUATIONS

In this section, an assessment of the project's results compared to its initial goals is made as well as a brief analysis about the durability of the project.

7.1 GOALS & RESULTS

The first goal was to do an analysis of the state-of-the-art about line crossings detection. This resulted in the information gathered and discussed in the section 2.2 - State of the art. It allowed to have a good overview of the techniques used nowadays in the computer vision field.

The choice to use standard techniques compared to the machine learning based ones, has not prevented the training of a CNN model from being tested. This test confirmed the concerns about the need of an extensive and relevant dataset as well as the process power and time needed to tune a model.

The realisation of a proof-of-concept with a fixed camera was the second goal. While the concrete installation of the POC cannot happen before the actual race, the conception and development of the prototype and the mock-up to test the system are a good step in this direction. Furthermore, the concept for the POC installation and planned tests that will be done during the race were described in the section 5 - Final proof of concept.

The third goal was to evaluate the system's performances. For that, evaluation metrics were defined in the 2.6 Evaluations metrics paragraph. The results were continuously reported at the end of each section.

The final objective, which was optional, was to assess the possibility to use a drone instead of a fixed camera. The section 2.5 - Drone discusses this topic. As no out-of-the-box commercial drone suitable for this project was found and that building a custom drone would have been too time consuming, it was decided to not further explore the drone solution.

7.2 TASKS & PLANNING

In the project specifications written at the beginning of the project, a list of tasks was made. Those tasks were used to define an execution planning. Let's compare the initial planning with the actual one.

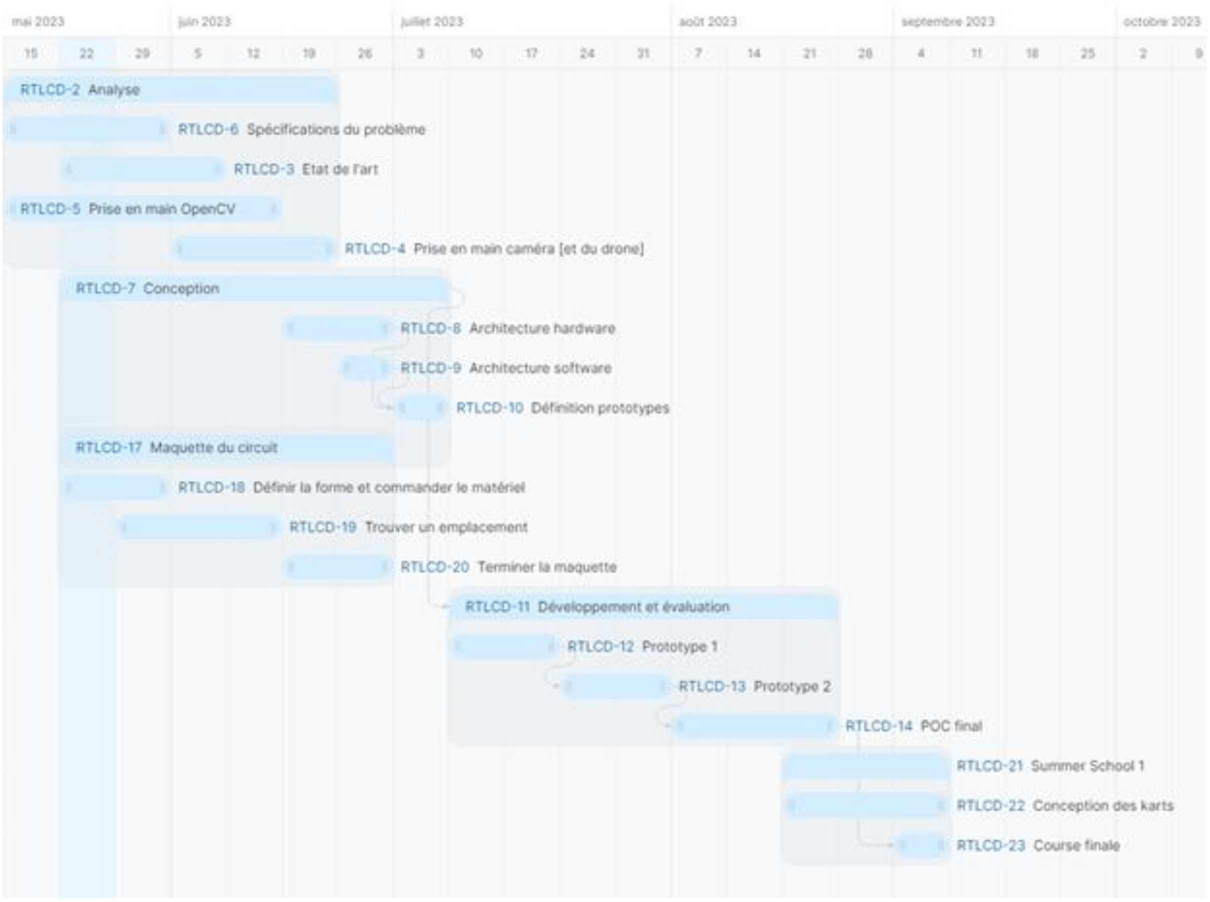


Figure 65: Initial planning

Globally, except for the few elements presented next, not much has changed from the initial planning and most tasks have been executed in times. Here are the few modifications made during the project.

Initially it was planned to realize two prototypes but after the design phase it was decided to work on only one prototype with two parts. So, the Prototype 1 and 2 elements in the planning have been grouped under the Prototype name.

A week worth of work, spanned on two weeks, has been added in early July. It was decided to test the training of a machine learning model, just before the prototype development.

And the final modification is the postponement of the proof-of-concept development to the last two weeks of summer school. As the proof-of-concept consists in combining the two parts of the prototype and applying minor modifications to test it during the race, it was decided to keep the last two weeks before the 25th of august to continue and finish this report.

One of the flaws of this initial planning is the lack of scheduled time to test the prototype in depth as well as the lack of time to complete the report at the end of the project.

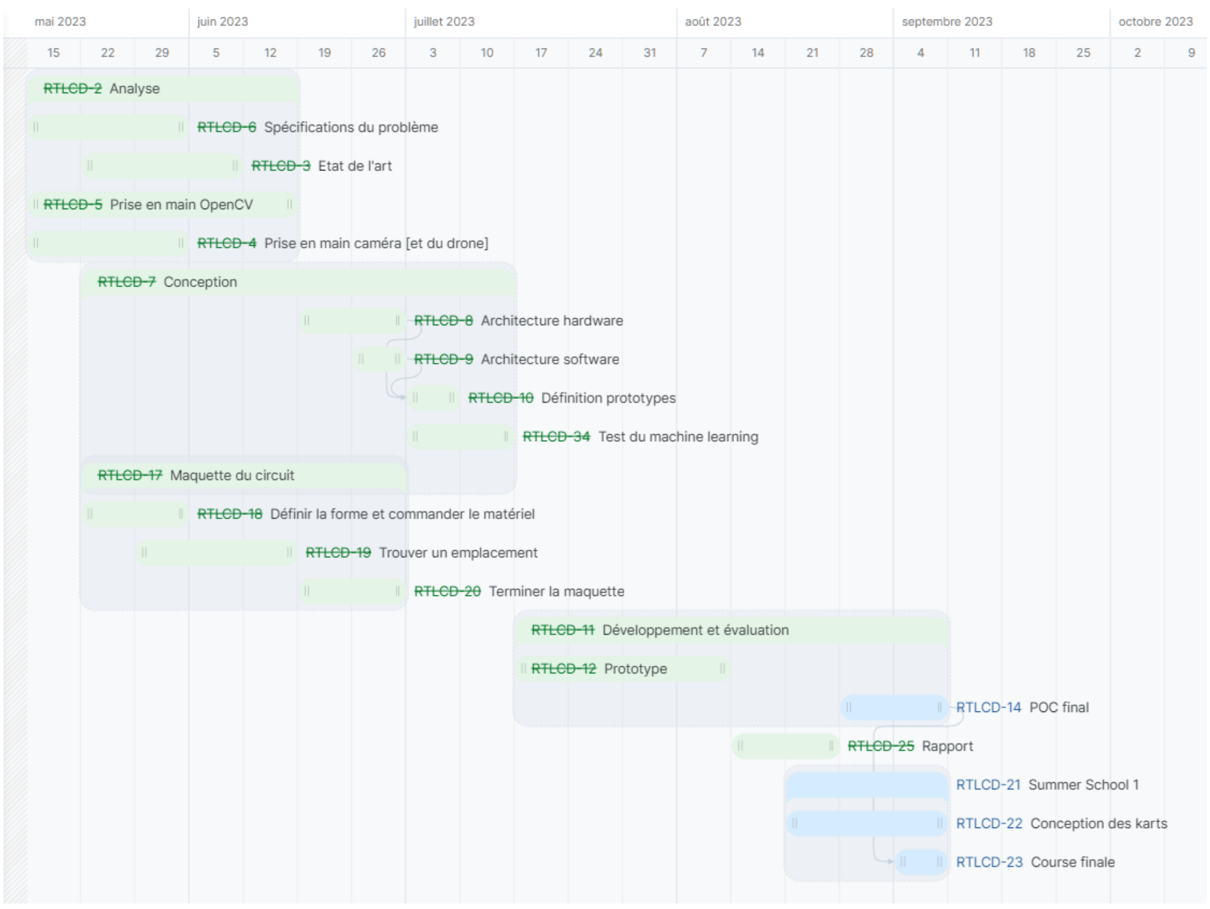


Figure 66: Actual planning

Here is a table that sums up the state of the tasks at the time of this report is submitted.

Every task has been completed except the ones in realisation as the development of the final proof-of-concept has been postponed and that it lacked time for proper prototype evaluation.

			Completed		
			Partially done		
			Not done		
	Tasks		**State**		
1	**Analysis**		▨	▨	▨
	a	Problem specification	🟩		
	b	State-of-the-art	🟩		
	c	Hands-on Camera and Drone	🟩		
	d	Hands-on OpenCV	🟩		
	e	Evaluation metrics definition	🟩		
2	**Conception**		▨	▨	▨
	a	Hardware	🟩		
	b	Software	🟩		
	c	Prototype	🟩		
	d	Maquette	🟩		
3	**Realisation**		▨	▨	▨
	a	Development		🟨	
	b	Evaluation		🟨	

8 DURABILITY

8.1 INTRODUCTION

The project to develop a complete solution for line crossings detection in the RC vehicle race held during the "Summer School I" resonates with various United Nations' Sustainable Development Goals [19], underscoring the potential for broader impacts beyond the event itself.

8.2 INNOVATION

By integrating computer vision techniques for line crossings detection, the project exemplifies **Goal 9: Industry, Innovation, Infrastructure** emphasis on innovation and technological advancements. While the primary focus is on the library's race, this initiative aligns with broader technological progress in traffic management and automated vehicle systems.

8.3 FAIRNESS & EQUITY

The adoption of automated line crossings detection addresses **Goal 10: Reduced Inequalities** by ensuring uniform and unbiased penalty administration. It supports **Goal 5: Gender Equality** as well as no discrimination based on the gender can be made by this system.

8.4 CONCLUSION

In conclusion, while durability may not be the principal goal, a correlation between the project and some of the Sustainable Development Goals is evident. Creating an automatic race monitoring system highlights the potential for technology and collaboration to contribute to broader goals of fairness, safety, and innovation in both localized events and the wider context of automated traffic management systems.

9 CONCLUSION

In conclusion, this thesis addressed the challenge of creating a race monitoring system. The initial phase involved analysing the problem's specifications, exploring hardware choices by looking at the different cameras and drones available on the market, and investigating computer vision techniques.

Following this, a prototype was first designed and then developed using Python and OpenCV. The prototype consisted of two main parts: a backend responsible for image processing, data extraction, and a front-end web server built with Django. This web server facilitated data storage and presentation through a user-friendly web page.

To evaluate the system, a test mock-up simulating race conditions was set up. Additionally, an exploration into machine learning techniques was conducted, involving the generation of synthetic race images to train a Convolutional Neural Network.

Ultimately, the prototype successfully achieved timing and crossing detection using traditional methods. The front-end web page provided users with the ability to monitor race information and view the best recorded times on a scoreboard.

Through a deeper understanding of Python and OpenCV, the setup of a web server with database, the configuration of a wireless security camera, the introduction to the machine learning world, some 3D printing and even wood working for the mock-up, this thesis started from problem analysis to a tangible prototype, making it a diverse and complete project.

Even if this system is dedicated to a local and fun use such as the summer school race, the computer vision methods used are an entry point towards more complex applications. Line-crossing detection find relevance in multiple domains like driverless cars, traffic management, and safety systems.

Sion, le 25 août 2023

Aurélien Rithner

10 REFERENCES

[1] HES-SO, [Online]. Available: https://www.hevs.ch/fr/photos/summer-school-2022--hei-204119.

[2] N. Joshi, “The Present And Future Of Computer Vision,” 26 Juin 2019. [Online]. Available: https://www.forbes.com/sites/cognitiveworld/2019/06/26/the-present-and-future-of-computer-vision/?sh=381733a517d8. [Accessed Août 2023].

[3] HES-SO Valais-Wallis, “Kart Gallery,” [Online]. Available: https://wiki.hevs.ch/fsi/index.php5/Kart/gallery. [Accessed 08 2023].

[4] LearnOpenCV, “Edge detection using OpenCV,” [Online]. Available: https://learnopencv.com/edge-detection-using-opencv/. [Accessed 06 2023].

[5] G. Boesch, “Deep Neural Network: The 3 Popular Types (MLP, CNN and RNN),” [Online]. Available: https://viso.ai/deep-learning/deep-neural-network-three-popular-types/.

[6] Wikipedia, “Precision and recall,” [Online]. Available: https://en.wikipedia.org/wiki/Precision_and_recall. [Accessed 2023].

[7] T. Wood, “F-Score,” [Online]. Available: https://deepai.org/machine-learning-glossary-and-terms/f-score. [Accessed 2023].

[8] G. Boesch, “What is OpenCV? The Complete Guide (2023),” 2023. [Online]. Available: https://viso.ai/computer-vision/opencv/.

[9] Wikipedia, “OpenCV,” [Online]. Available: https://en.wikipedia.org/wiki/OpenCV#. [Accessed 08 2023].

[10] JetBrains, “PyCharm,” [Online]. Available: https://www.jetbrains.com/pycharm/. [Accessed 08 2023].

[11] OpenCV, “Hough Line Transform,” [Online]. Available: https://docs.opencv.org/3.4/d9/db0/tutorial_hough_lines.html. [Accessed 2023].

[12] S. Mallick, “Object Tracking using OpenCV,” [Online]. Available: https://learnopencv.com/object-tracking-using-opencv-cpp-python/. [Accessed 2023].

[13] OpenCV, “Camera Calibration,” [Online]. Available: https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html. [Accessed 2023].

[14] DJI, “Tello,” [Online]. Available: https://store.dji.com/ch/product/tello?vid=38421. [Accessed 2023].

[15] studioSport, “Parrot Anafi Ai,” [Online]. Available: https://www.studiosport.fr/parrot-anafi-ai-a22224.html. [Accessed 2023].

[16] C. Bergquist, “Make an OpenCV drone,” [Online]. Available: https://dojofordrones.com/opencv-drone/. [Accessed 2023].

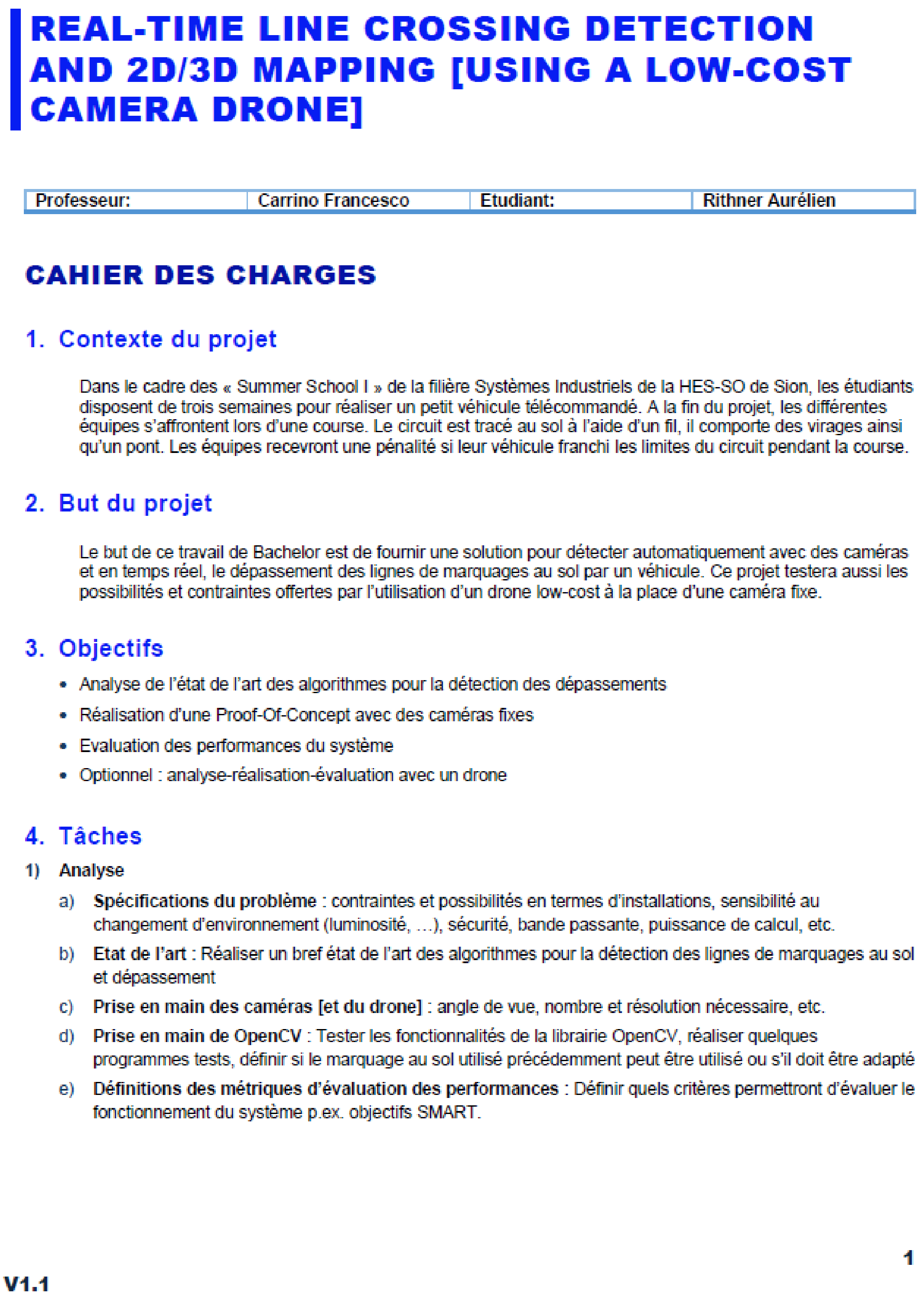
[17] rorygames, “V-HACD - Complex Colliders,” [Online]. Available: https://github.com/rorygames/VHACD. [Accessed 2023].

[18] Kaggle, “Tensor Processing Units,” [Online]. Available: https://www.kaggle.com/docs/tpu. [Accessed 2023].

[19] United Nations, “17 Goals to Transform Our World,” [Online]. Available: https://www.un.org/sustainabledevelopment/. [Accessed 2023].

11 Appendices

11.1 Project specifications



REAL-TIME LINE CROSSING DETECTION AND 2D/3D MAPPING [USING A LOW-COST CAMERA DRONE]

Professeur:	Carrino Francesco	Etudiant:	Rithner Aurélien

CAHIER DES CHARGES

1. Contexte du projet

Dans le cadre des « Summer School I » de la filière Systèmes Industriels de la HES-SO de Sion, les étudiants disposent de trois semaines pour réaliser un petit véhicule télécommandé. A la fin du projet, les différentes équipes s'affrontent lors d'une course. Le circuit est tracé au sol à l'aide d'un fil, il comporte des virages ainsi qu'un pont. Les équipes recevront une pénalité si leur véhicule franchi les limites du circuit pendant la course.

2. But du projet

Le but de ce travail de Bachelor est de fournir une solution pour détecter automatiquement avec des caméras et en temps réel, le dépassement des lignes de marquages au sol par un véhicule. Ce projet testera aussi les possibilités et contraintes offertes par l'utilisation d'un drone low-cost à la place d'une caméra fixe.

3. Objectifs

- Analyse de l'état de l'art des algorithmes pour la détection des dépassements
- Réalisation d'une Proof-Of-Concept avec des caméras fixes
- Evaluation des performances du système
- Optionnel : analyse-réalisation-évaluation avec un drone

4. Tâches

1) **Analyse**

   a) **Spécifications du problème** : contraintes et possibilités en termes d'installations, sensibilité au changement d'environnement (luminosité, …), sécurité, bande passante, puissance de calcul, etc.

   b) **Etat de l'art** : Réaliser un bref état de l'art des algorithmes pour la détection des lignes de marquages au sol et dépassement

   c) **Prise en main des caméras [et du drone]** : angle de vue, nombre et résolution nécessaire, etc.

   d) **Prise en main de OpenCV** : Tester les fonctionnalités de la librairie OpenCV, réaliser quelques programmes tests, définir si le marquage au sol utilisé précédemment peut être utilisé ou s'il doit être adapté

   e) **Définitions des métriques d'évaluation des performances** : Définir quels critères permettront d'évaluer le fonctionnement du système p.ex. objectifs SMART.

1

V1.1

2) **Conception**
   a) **Hardware** : Définition de l'architecture du système (hardware), positionnement de la caméra, machine utilisée pour le traitement des images, canal de transmission des données, interface avec les utilisateurs
   b) **Software** : Définition de l'architecture du système (software), quels outils pour la détection de ligne/ reconnaissance d'images, protocole de transmission des données, interface avec les utilisateurs
   c) **Prototypes** : Définition des objectifs/limites pour 1 à 2 prototypes vers le Proof-Of-Concept final, spécifier les points que les différents prototypes permettront de tester
   d) **Maquette** : Définition de la forme de la maquette, commande du matériel, recherche d'un emplacement
3) **Développement et évaluation**
   a) **Développement** : Réaliser concrètement les différents prototypes jusqu'à la réalisation du POC final et de manière itérative
   b) **Evaluation** : Tester le système suivant les objectifs et métriques définis pendant la phase d'analyse, aussi de manière itérative durant le développement

5. Planning

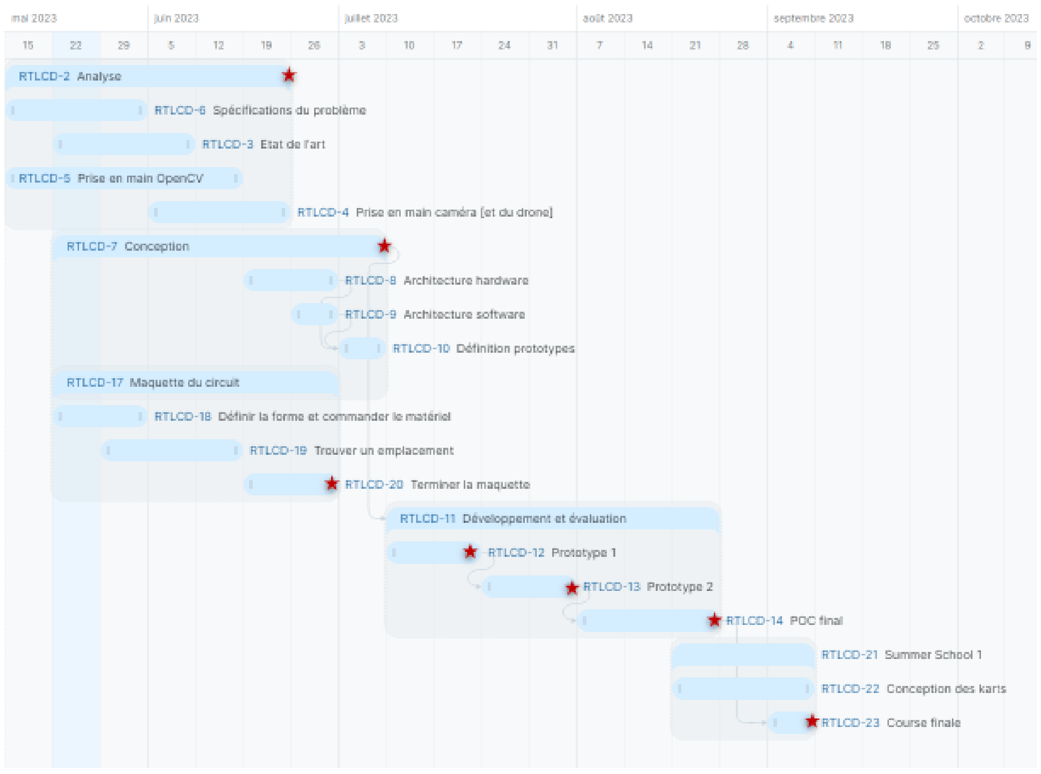


Figure 1: Planning global Gantt avec les parties principales et leurs tâches. Les « milestones » du projet sont représentés par les étoiles rouges.

6. Milestones

Date	Description	Délivrables
20.06.2023	Présentation intermédiaire	○ Début du rapport (Introduction, partie analyse)
30.06.2023	Fin de Maquette	○ Maquette du circuit pour effectuer les tests
07.07.2023	Fin de Conception	○ Architecture du systèmes et définitions des prototypes ○ Partie conception du rapport
21.07.2023	Fin de Prototype #1	○ Prototype #1 fonctionnel
04.08.2023	Fin de Prototype #2	○ Prototype #2 fonctionnel
25.08.2023	Fin du TB	○ Proof of Concept fonctionnel ○ Rapport complet
08.09.2023	Fin de SS1	Test du POC pendant les course réelles

11.2 USER INTERFACE

As mentioned earlier in the problem specifications, there has to be a way the user/public can manage the system and see the results.

Here's a few methods that could work:

11.2.1 CONSOLE OUTPUT

The user will run the script processing the images and the results will be displayed in the console.

Inputs would be terminal inputs with a simple text menu.

11.2.2 CONSOLE & OPENCV WINDOW

The data will be displayed in the console but the user will have a graphical view of the result in an OpenCV window.

Inputs would be terminal inputs with a simple text menu.
There can be multiple windows opening and closing to show for example the binarized image with the contours of the track and then the live camera feed.

11.2.3 OPENCV WINDOW ONLY

Displaying the data by adding text to the displayed images in an OpenCV window.

Inputs would be via the waitKey method.

11.2.4 GRAPHICAL UI

Using a library like PySimpleGUI a full GUI with buttons, sliders and result image can be displayed to the user.

This method would be easy to use and not too complicated to implement but with the tests done at the moment the output seems to be a bit slow to process resulting in a pretty large delay. This is linked with the way the code is built to run the GUI.

11.2.5 FRONT END FOR DISPLAY & BACK END FOR INPUT

A web page could display the result while user inputs are managed by the python script using any methods previously proposed.

The data could be passed to the web page in a json file and displayed aesthetically on a page together with the image feed.

This method enables a one central machine for processing and multiple terminals for display architecture while being relatively simple to implement.

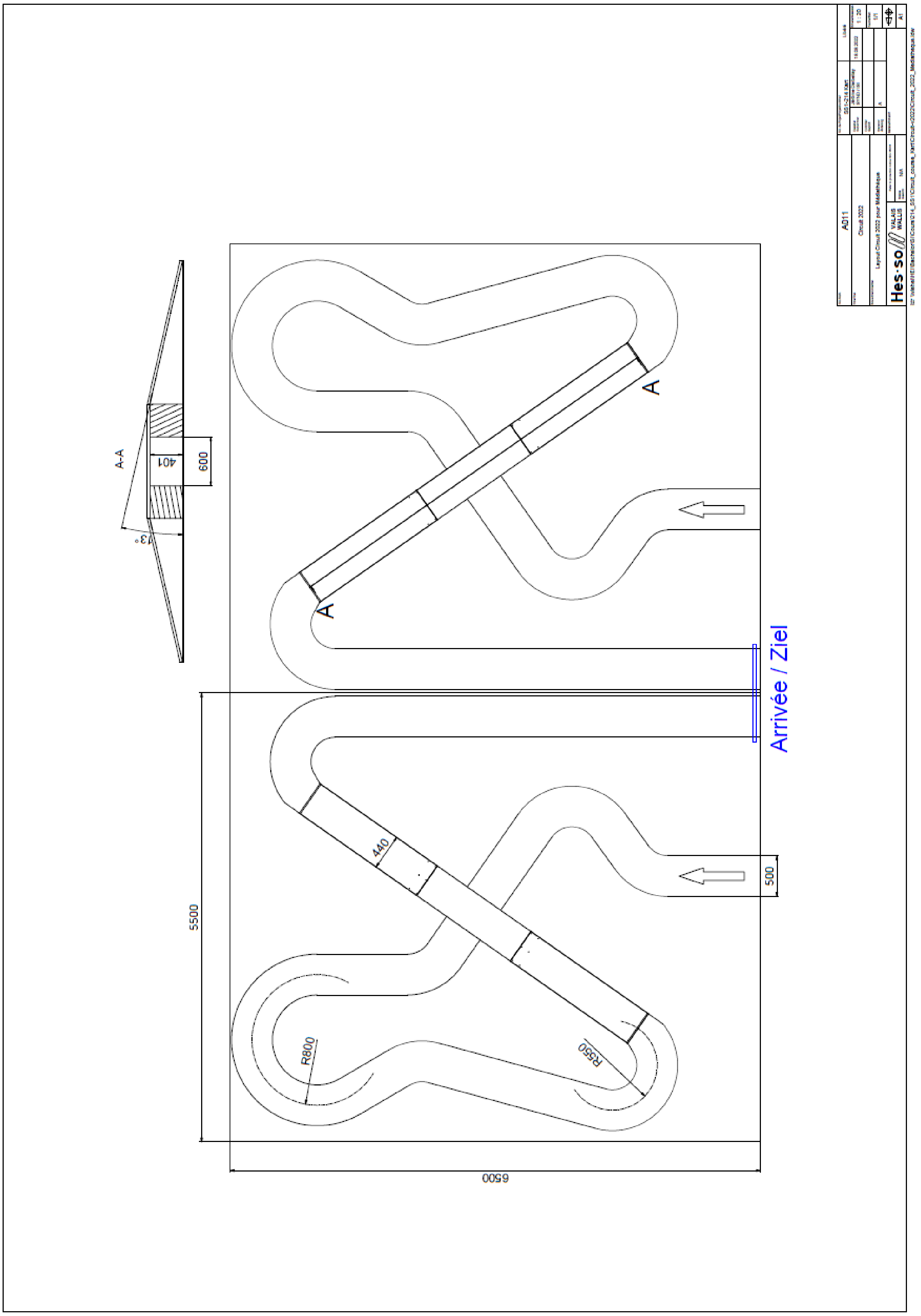
11.2.6 FRONT END FOR DISPLAY + INPUT & BACK END FOR PROCESSING

User will be provided with a full GUI on the web page and only the image processing will be managed in the back end.

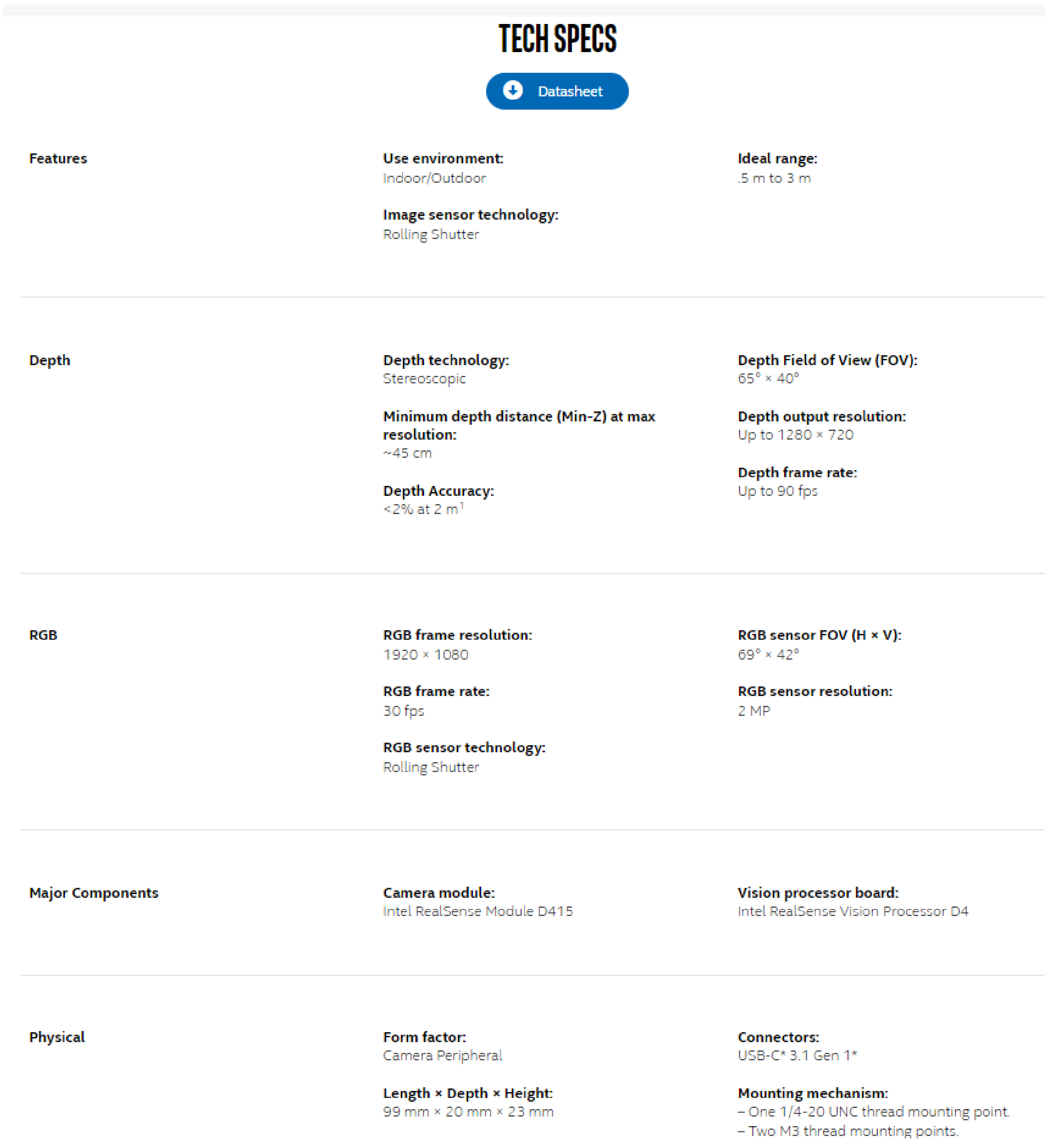
This method requires running a web server, and two-way communication.

There would be one page with only the result information displayed to the public/competitors. And another page with sliders and buttons to control the system.

11.3 RACE TRACK DRAWINGS



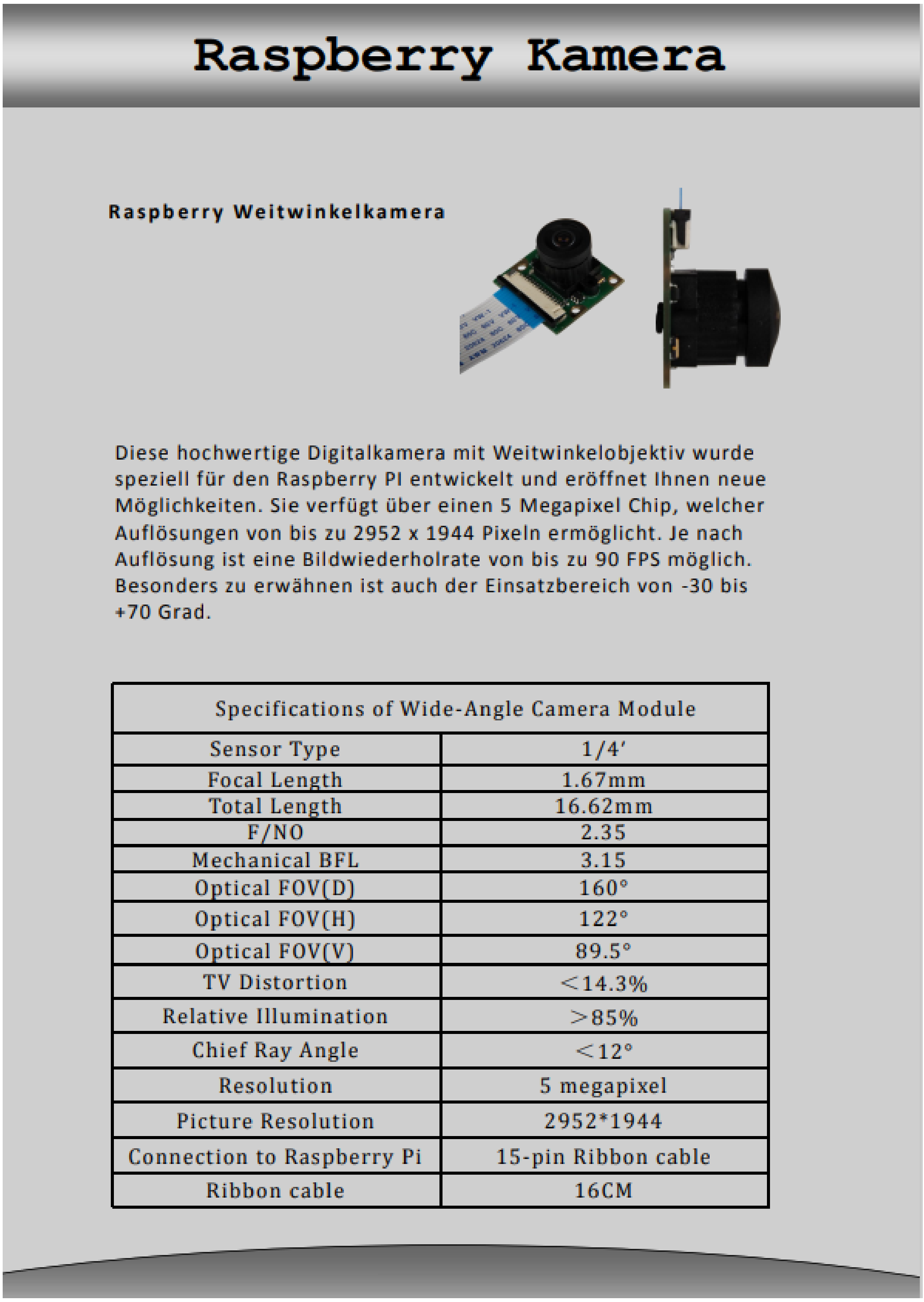
11.4 Cameras Specifications



TECH SPECS

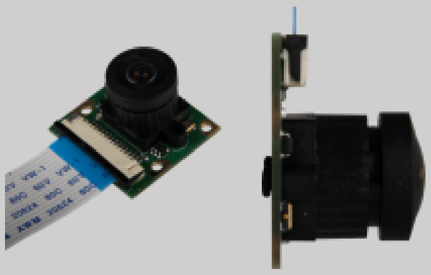
Features	**Use environment:** Indoor/Outdoor	**Ideal range:** .5 m to 3 m
	Image sensor technology: Rolling Shutter	
Depth	**Depth technology:** Stereoscopic	**Depth Field of View (FOV):** 65° × 40°
	Minimum depth distance (Min-Z) at max resolution: ~45 cm	**Depth output resolution:** Up to 1280 × 720
	Depth Accuracy: <2% at 2 m[1]	**Depth frame rate:** Up to 90 fps
RGB	**RGB frame resolution:** 1920 × 1080	**RGB sensor FOV (H × V):** 69° × 42°
	RGB frame rate: 30 fps	**RGB sensor resolution:** 2 MP
	RGB sensor technology: Rolling Shutter	
Major Components	**Camera module:** Intel RealSense Module D415	**Vision processor board:** Intel RealSense Vision Processor D4
Physical	**Form factor:** Camera Peripheral	**Connectors:** USB-C* 3.1 Gen 1*
	Length × Depth × Height: 99 mm × 20 mm × 23 mm	**Mounting mechanism:** – One 1/4-20 UNC thread mounting point. – Two M3 thread mounting points.

Figure 67: Technical specifications for the Intel D145



Raspberry Kamera

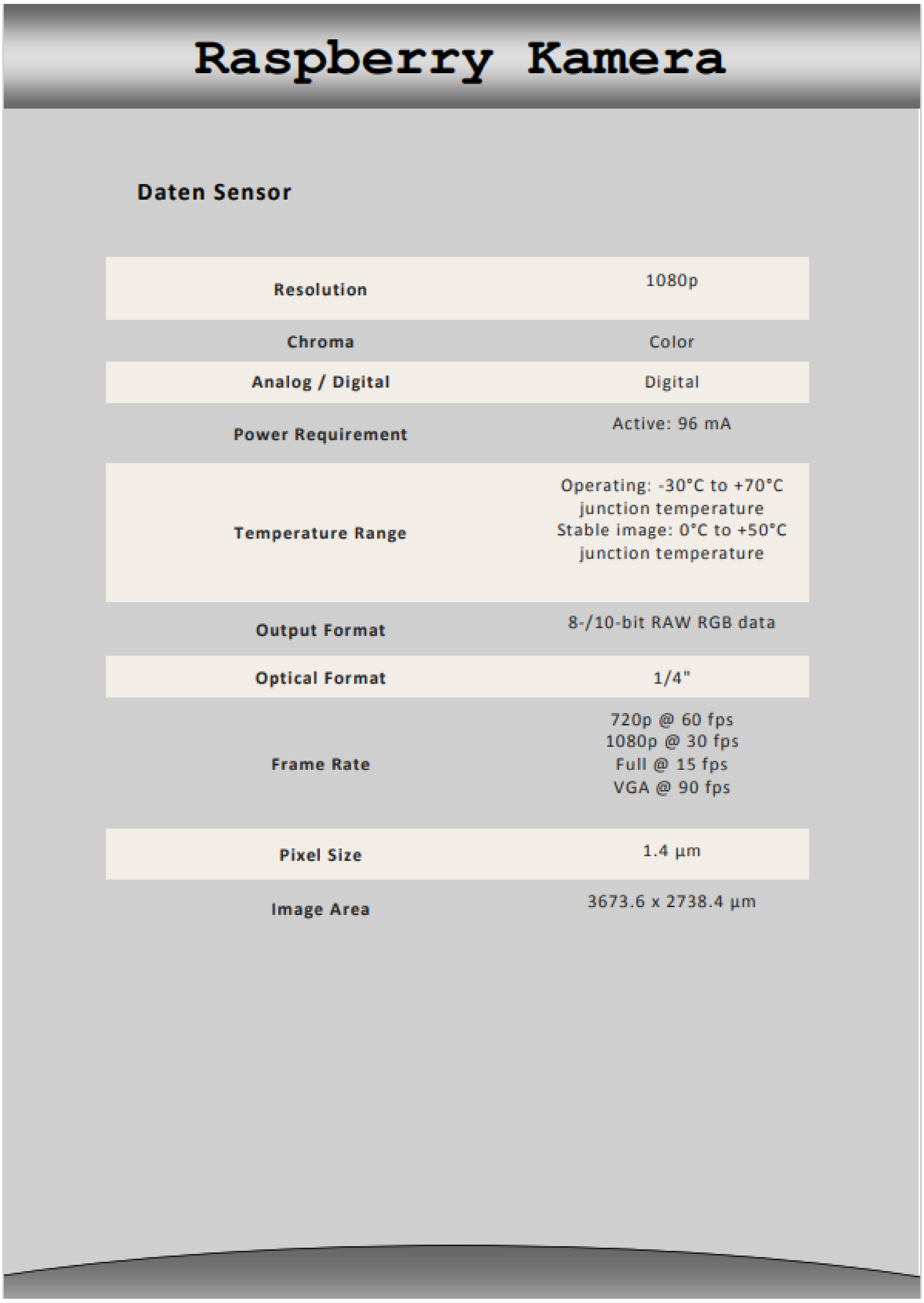
Raspberry Weitwinkelkamera



Diese hochwertige Digitalkamera mit Weitwinkelobjektiv wurde speziell für den Raspberry PI entwickelt und eröffnet Ihnen neue Möglichkeiten. Sie verfügt über einen 5 Megapixel Chip, welcher Auflösungen von bis zu 2952 x 1944 Pixeln ermöglicht. Je nach Auflösung ist eine Bildwiederholrate von bis zu 90 FPS möglich. Besonders zu erwähnen ist auch der Einsatzbereich von -30 bis +70 Grad.

Specifications of Wide-Angle Camera Module	
Sensor Type	1/4′
Focal Length	1.67mm
Total Length	16.62mm
F/NO	2.35
Mechanical BFL	3.15
Optical FOV(D)	160°
Optical FOV(H)	122°
Optical FOV(V)	89.5°
TV Distortion	<14.3%
Relative Illumination	>85%
Chief Ray Angle	<12°
Resolution	5 megapixel
Picture Resolution	2952*1944
Connection to Raspberry Pi	15-pin Ribbon cable
Ribbon cable	16CM

Figure 68: Page 1 of the technical specifications for the RPI WWCAM

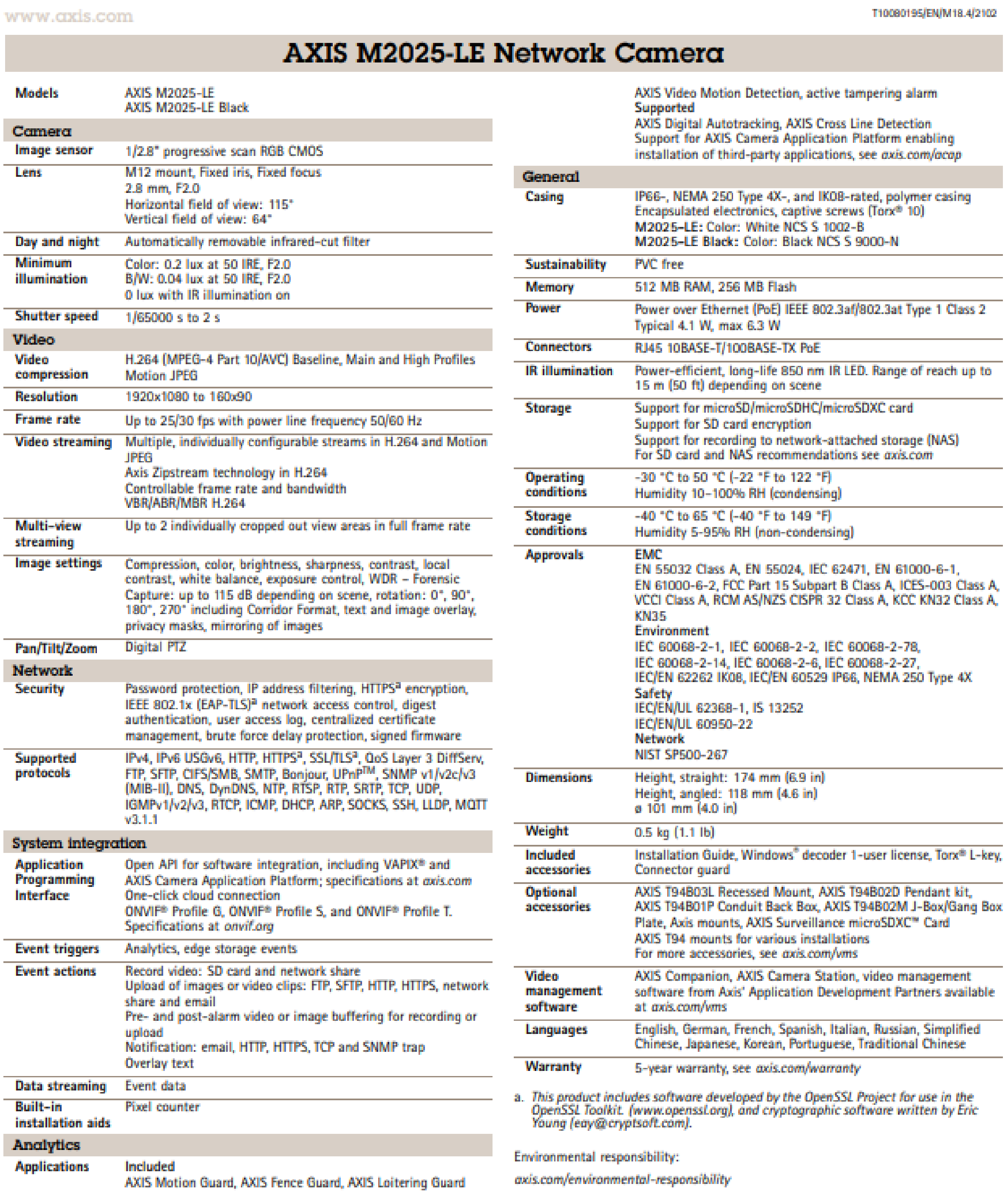


Raspberry Kamera

Daten Sensor

Resolution	1080p
Chroma	Color
Analog / Digital	Digital
Power Requirement	Active: 96 mA
Temperature Range	Operating: -30°C to +70°C junction temperature Stable image: 0°C to +50°C junction temperature
Output Format	8-/10-bit RAW RGB data
Optical Format	1/4"
Frame Rate	720p @ 60 fps 1080p @ 30 fps Full @ 15 fps VGA @ 90 fps
Pixel Size	1.4 µm
Image Area	3673.6 x 2738.4 µm

Figure 69: Page 2 of the technical specifications for the RPI WWCAM



www.axis.com

T10080195/EN/M18.4/2102

AXIS M2025-LE Network Camera

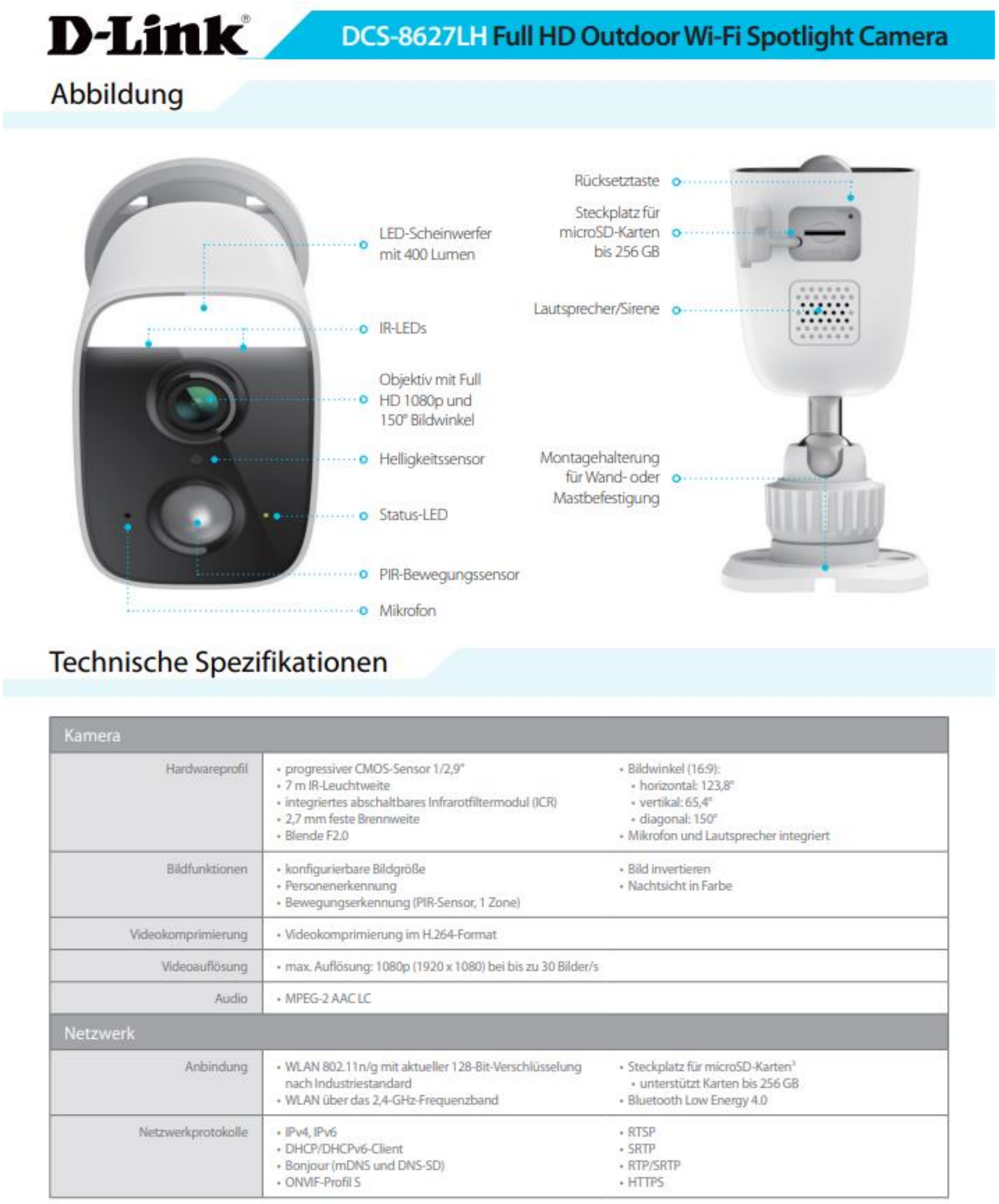
Models	AXIS M2025-LE AXIS M2025-LE Black
Camera	
Image sensor	1/2.8" progressive scan RGB CMOS
Lens	M12 mount, Fixed iris, Fixed focus 2.8 mm, F2.0 Horizontal field of view: 115° Vertical field of view: 64°
Day and night	Automatically removable infrared-cut filter
Minimum illumination	Color: 0.2 lux at 50 IRE, F2.0 B/W: 0.04 lux at 50 IRE, F2.0 0 lux with IR illumination on
Shutter speed	1/65000 s to 2 s
Video	
Video compression	H.264 (MPEG-4 Part 10/AVC) Baseline, Main and High Profiles Motion JPEG
Resolution	1920x1080 to 160x90
Frame rate	Up to 25/30 fps with power line frequency 50/60 Hz
Video streaming	Multiple, individually configurable streams in H.264 and Motion JPEG Axis Zipstream technology in H.264 Controllable frame rate and bandwidth VBR/ABR/MBR H.264
Multi-view streaming	Up to 2 individually cropped out view areas in full frame rate
Image settings	Compression, color, brightness, sharpness, contrast, local contrast, white balance, exposure control, WDR – Forensic Capture: up to 115 dB depending on scene, rotation: 0°, 90°, 180°, 270° including Corridor Format, text and image overlay, privacy masks, mirroring of images
Pan/Tilt/Zoom	Digital PTZ
Network	
Security	Password protection, IP address filtering, HTTPS[a] encryption, IEEE 802.1x (EAP-TLS)[a] network access control, digest authentication, user access log, centralized certificate management, brute force delay protection, signed firmware
Supported protocols	IPv4, IPv6 USGv6, HTTP, HTTPS[a], SSL/TLS[a], QoS Layer 3 DiffServ, FTP, SFTP, CIFS/SMB, SMTP, Bonjour, UPnP™, SNMP v1/v2c/v3 (MIB-II), DNS, DynDNS, NTP, RTSP, RTP, SRTP, TCP, UDP, IGMPv1/v2/v3, RTCP, ICMP, DHCP, ARP, SOCKS, SSH, LLDP, MQTT v3.1.1
System integration	
Application Programming Interface	Open API for software integration, including VAPIX® and AXIS Camera Application Platform; specifications at *axis.com* One-click cloud connection ONVIF® Profile G, ONVIF® Profile S, and ONVIF® Profile T. Specifications at *onvif.org*
Event triggers	Analytics, edge storage events
Event actions	Record video: SD card and network share Upload of images or video clips: FTP, SFTP, HTTP, HTTPS, network share and email Pre- and post-alarm video or image buffering for recording or upload Notification: email, HTTP, HTTPS, TCP and SNMP trap Overlay text
Data streaming	Event data
Built-in installation aids	Pixel counter
Analytics	
Applications	**Included** AXIS Motion Guard, AXIS Fence Guard, AXIS Loitering Guard AXIS Video Motion Detection, active tampering alarm **Supported** AXIS Digital Autotracking, AXIS Cross Line Detection Support for AXIS Camera Application Platform enabling installation of third-party applications, see *axis.com/acap*
General	
Casing	IP66-, NEMA 250 Type 4X-, and IK08-rated, polymer casing Encapsulated electronics, captive screws (Torx® 10) **M2025-LE:** Color: White NCS S 1002-B **M2025-LE Black:** Color: Black NCS S 9000-N
Sustainability	PVC free
Memory	512 MB RAM, 256 MB Flash
Power	Power over Ethernet (PoE) IEEE 802.3af/802.3at Type 1 Class 2 Typical 4.1 W, max 6.3 W
Connectors	RJ45 10BASE-T/100BASE-TX PoE
IR illumination	Power-efficient, long-life 850 nm IR LED. Range of reach up to 15 m (50 ft) depending on scene
Storage	Support for microSD/microSDHC/microSDXC card Support for SD card encryption Support for recording to network-attached storage (NAS) For SD card and NAS recommendations see *axis.com*
Operating conditions	-30 °C to 50 °C (-22 °F to 122 °F) Humidity 10–100% RH (condensing)
Storage conditions	-40 °C to 65 °C (-40 °F to 149 °F) Humidity 5–95% RH (non-condensing)
Approvals	**EMC** EN 55032 Class A, EN 55024, IEC 62471, EN 61000-6-1, EN 61000-6-2, FCC Part 15 Subpart B Class A, ICES-003 Class A, VCCI Class A, RCM AS/NZS CISPR 32 Class A, KCC KN32 Class A, KN35 **Environment** IEC 60068-2-1, IEC 60068-2-2, IEC 60068-2-78, IEC 60068-2-14, IEC 60068-2-6, IEC 60068-2-27, IEC/EN 62262 IK08, IEC/EN 60529 IP66, NEMA 250 Type 4X **Safety** IEC/EN/UL 62368-1, IS 13252 IEC/EN/UL 60950-22 **Network** NIST SP500-267
Dimensions	Height, straight: 174 mm (6.9 in) Height, angled: 118 mm (4.6 in) ø 101 mm (4.0 in)
Weight	0.5 kg (1.1 lb)
Included accessories	Installation Guide, Windows® decoder 1-user license, Torx® L-key, Connector guard
Optional accessories	AXIS T94B03L Recessed Mount, AXIS T94B02D Pendant kit, AXIS T94B01P Conduit Back Box, AXIS T94B02M J-Box/Gang Box Plate, Axis mounts, AXIS Surveillance microSDXC™ Card AXIS T94 mounts for various installations For more accessories, see *axis.com/vms*
Video management software	AXIS Companion, AXIS Camera Station, video management software from Axis' Application Development Partners available at *axis.com/vms*
Languages	English, German, French, Spanish, Italian, Russian, Simplified Chinese, Japanese, Korean, Portuguese, Traditional Chinese
Warranty	5-year warranty, see *axis.com/warranty*

a. *This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit. (www.openssl.org), and cryptographic software written by Eric Young (eay@cryptsoft.com).*

Environmental responsibility:

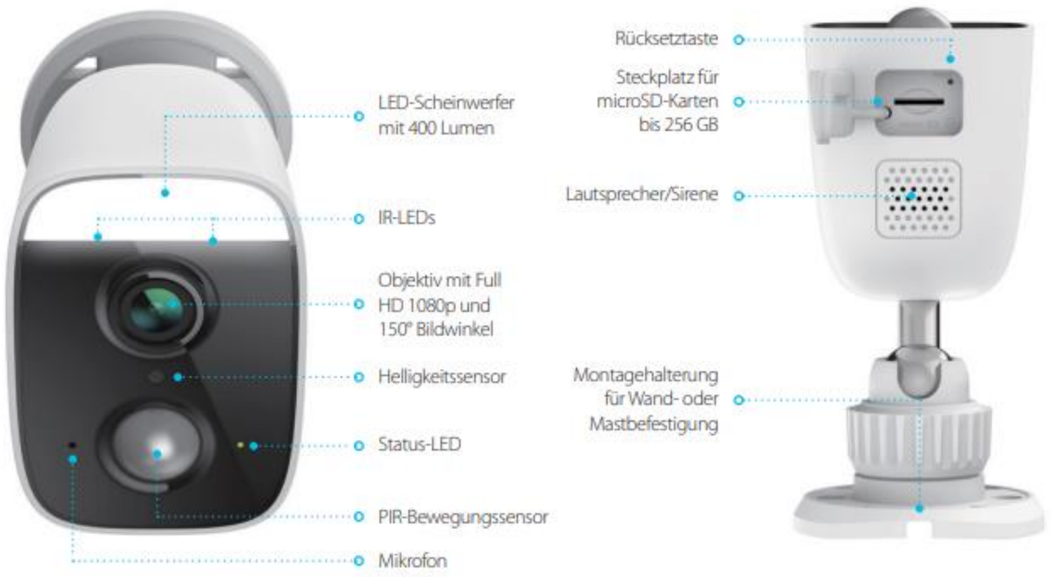
axis.com/environmental-responsibility

Figure 70: Technical specifications for the Axis M2025-LE



D-Link DCS-8627LH Full HD Outdoor Wi-Fi Spotlight Camera

Abbildung


LED-Scheinwerfer mit 400 Lumen
IR-LEDs
Objektiv mit Full HD 1080p und 150° Bildwinkel
Helligkeitssensor
Status-LED
PIR-Bewegungssensor
Mikrofon
Rücksetztaste
Steckplatz für microSD-Karten bis 256 GB
Lautsprecher/Sirene
Montagehalterung für Wand- oder Mastbefestigung

Technische Spezifikationen

Kamera		
Hardwareprofil	• progressiver CMOS-Sensor 1/2,9" • 7 m IR-Leuchtweite • integriertes abschaltbares Infrarotfiltermodul (ICR) • 2,7 mm feste Brennweite • Blende F2.0	• Bildwinkel (16:9): • horizontal: 123,8° • vertikal: 65,4° • diagonal: 150° • Mikrofon und Lautsprecher integriert
Bildfunktionen	• konfigurierbare Bildgröße • Personenerkennung • Bewegungserkennung (PIR-Sensor, 1 Zone)	• Bild invertieren • Nachtsicht in Farbe
Videokomprimierung	• Videokomprimierung im H.264-Format	
Videoauflösung	• max. Auflösung: 1080p (1920 x 1080) bei bis zu 30 Bilder/s	
Audio	• MPEG-2 AAC LC	
Netzwerk		
Anbindung	• WLAN 802.11n/g mit aktueller 128-Bit-Verschlüsselung nach Industriestandard • WLAN über das 2,4-GHz-Frequenzband	• Steckplatz für microSD-Karten[3] • unterstützt Karten bis 256 GB • Bluetooth Low Energy 4.0
Netzwerkprotokolle	• IPv4, IPv6 • DHCP/DHCPv6-Client • Bonjour (mDNS und DNS-SD) • ONVIF-Profil S	• RTSP • SRTP • RTP/SRTP • HTTPS

Figure 71: Technical specifications for the DLink DCS-8627LH

11.5 Cameras placement height estimations

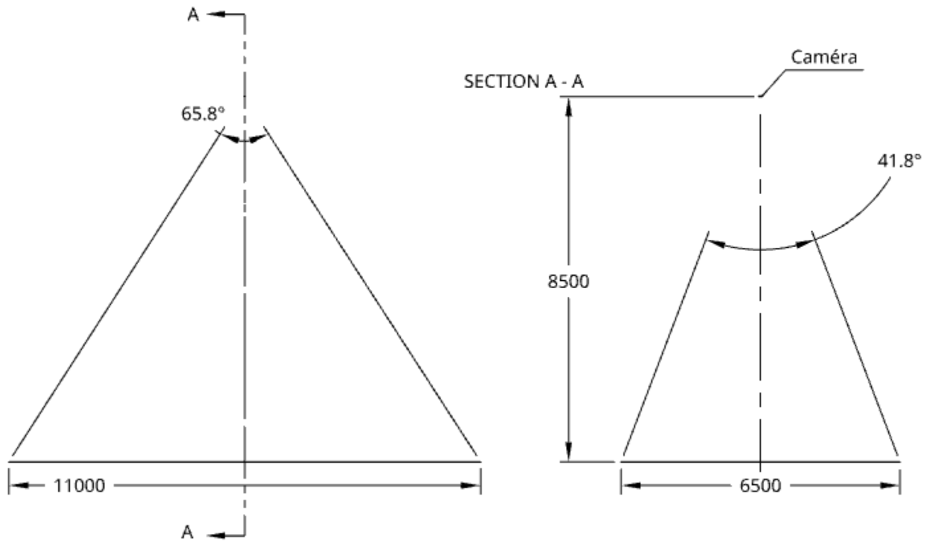


Figure 72 : Height estimation for the Intel D415

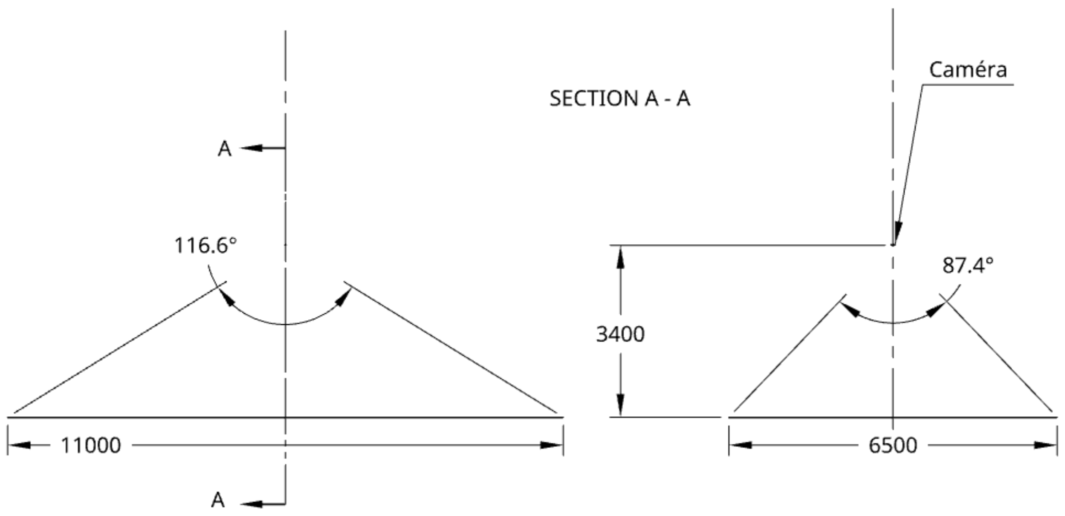


Figure 73 : Height estimation for the RPI WWCAM

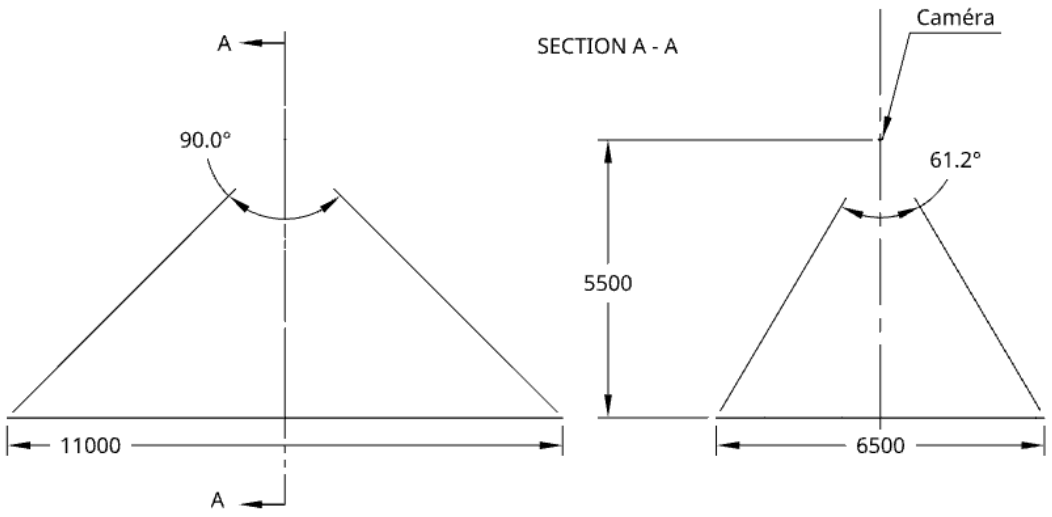


Figure 74 : Height estimation for the DLINK DCS-8627LH

11.6 Particles filter Tracking tests

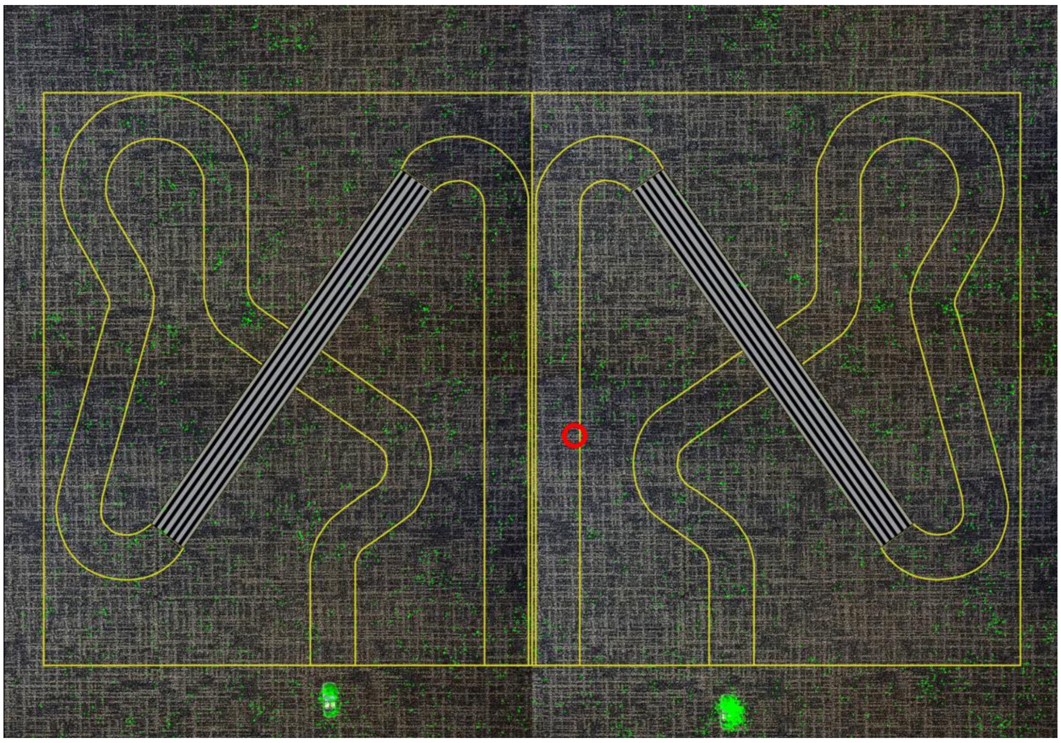


Figure 75 : At first particles are distributed everywhere randomly

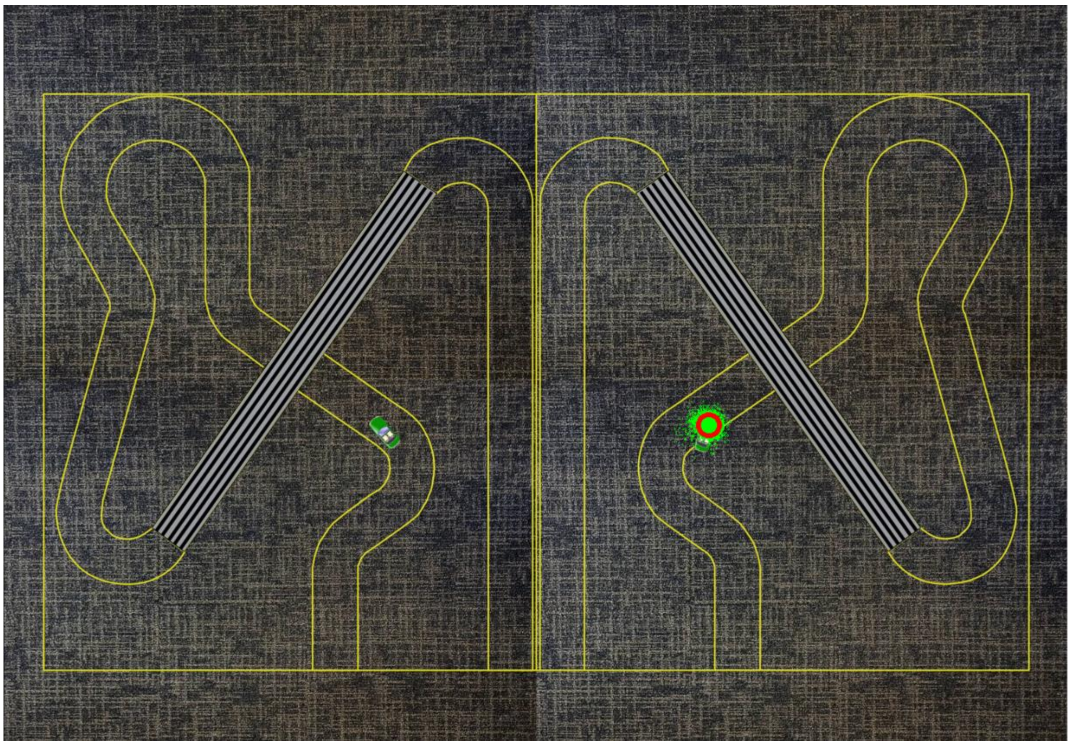


Figure 76 : Quickly they focus on the green object. In this example there are two identic cars and with the parameters used they followed only one

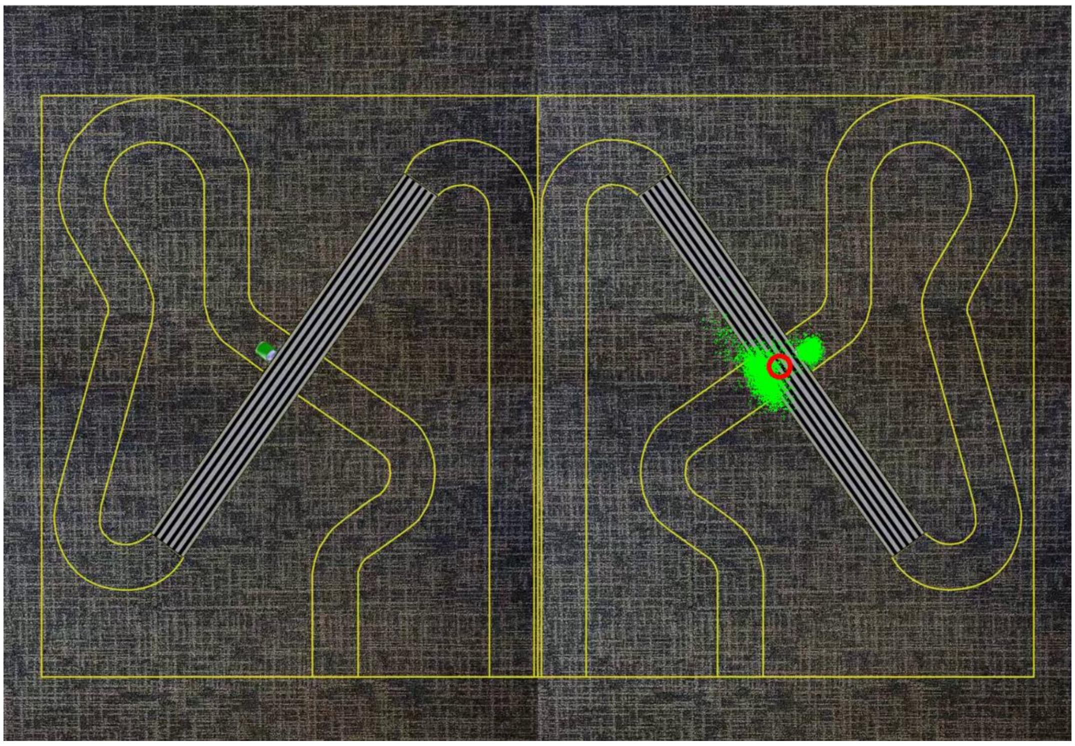


Figure 77 : With this algorithm the tracking continues after the bridge

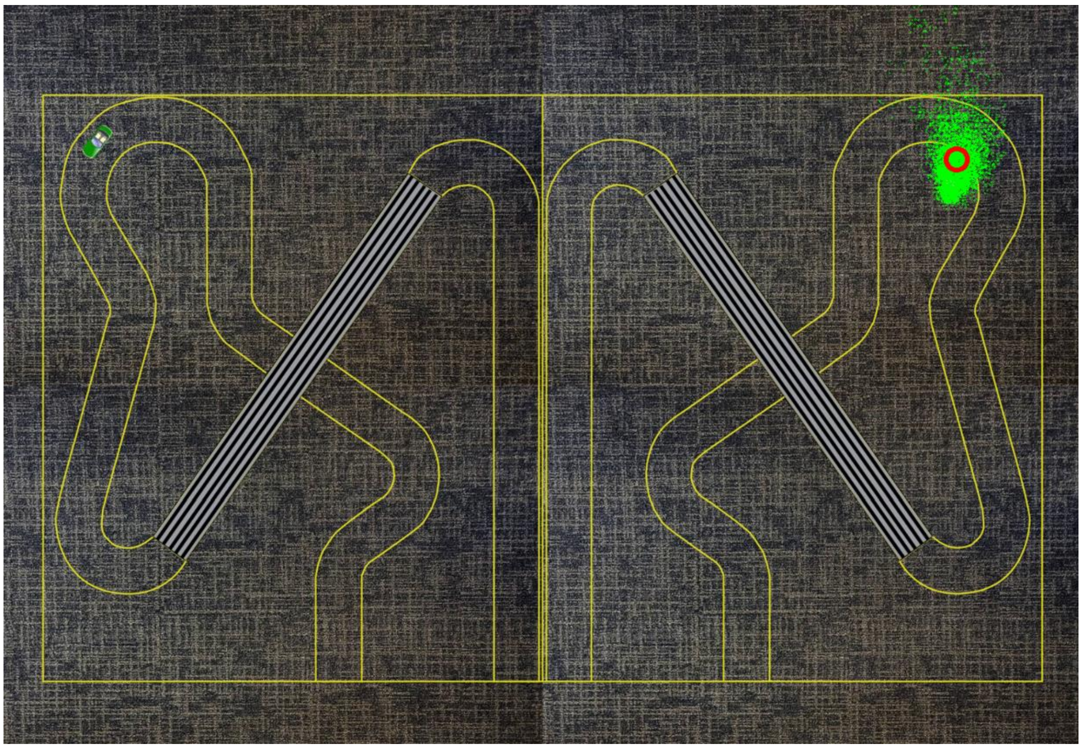


Figure 78 : During the 180 degree turn some of the particles drift away but the tracking continues

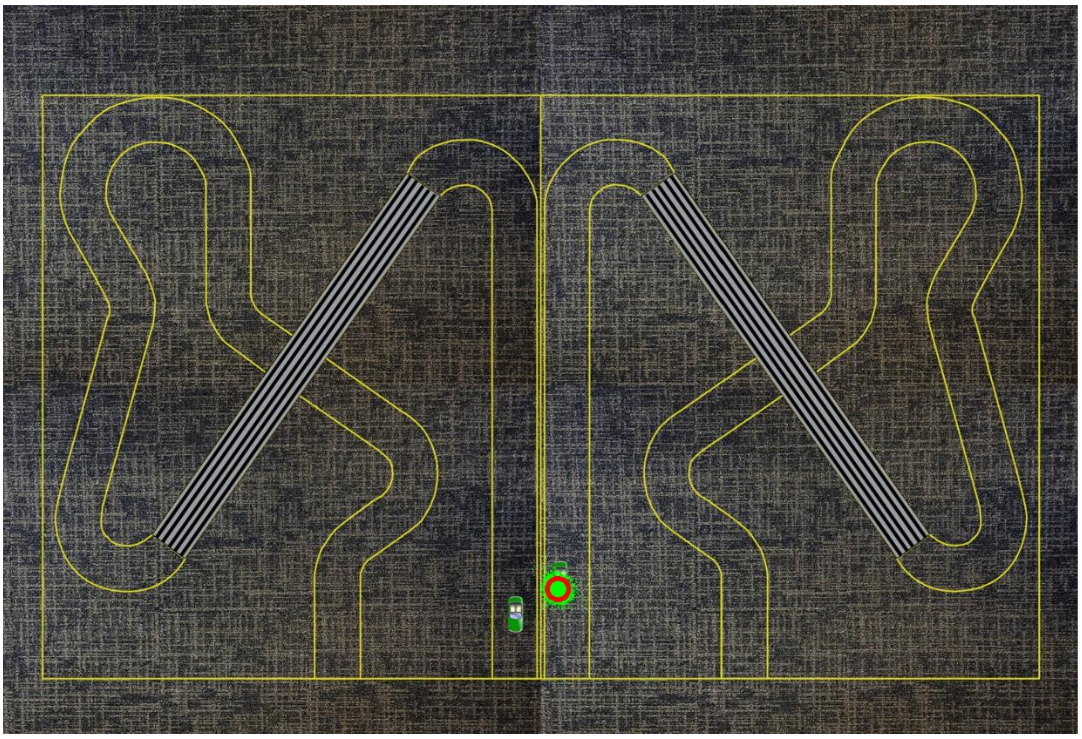


Figure 79 : Tracking all the way until the end

11.7 MOCK-UP PARTS DETAILS

11.7.1 WOODEN FRAME

The frame is made of 18mm x 47mm lumbers. Its dimensions match those of the printed board.

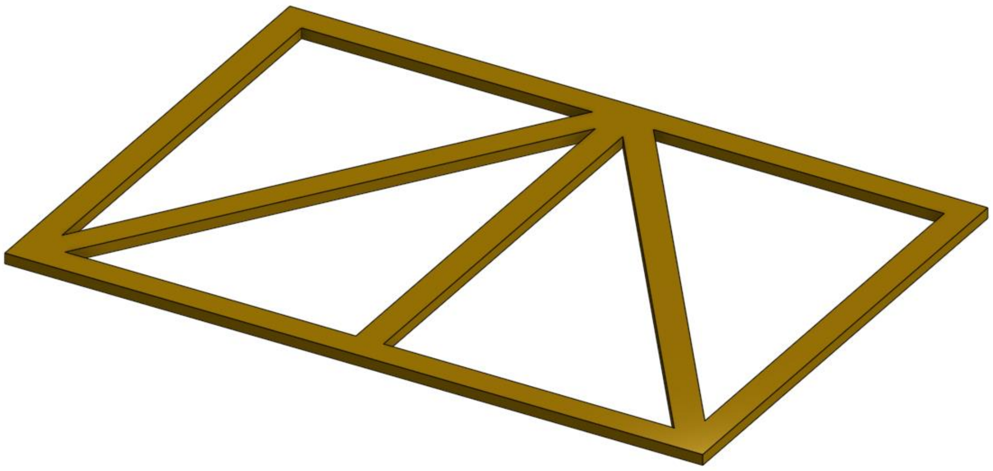


Figure 80 : 3D view of the frame

11.7.2 HANDLES

The handles will be 3D printed and have two holes so they can be screwed directly on the side of the wooden frame.

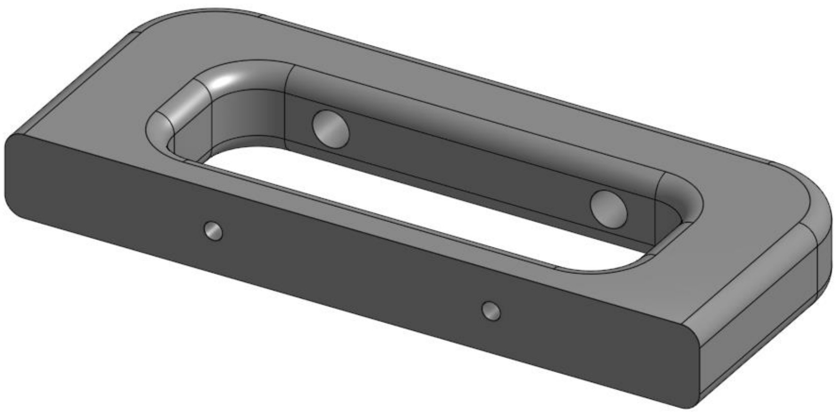


Figure 81 : 3D view of a handle

11.7.3 BASE

The main base is 3D printed and will connect the pole to the frame. As the first design was too big to fit on the bed of the printer, the second is made of three parts. The two side arms are fixed to the middle part using screws and threaded inserts. The pole and frame are secured using wood screws.

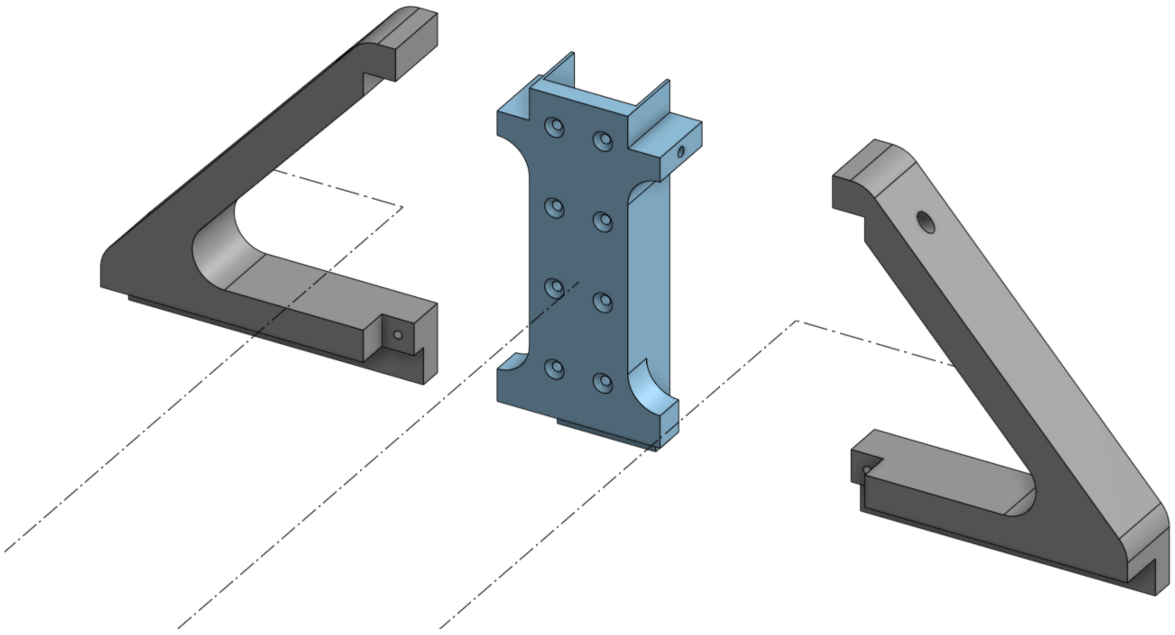


Figure 82 : 3D exploded view of the base parts

11.7.4 Pole

The pole is a 33mm x 57mm lumber. It will be cut to the right size depending on what height the camera must be positioned.

Figure 83 : 3D view of the pole

11.7.5 TUBE-POLE CONNECTOR

This 3D printed part joins the steel tube and the pole together. It also enables the tube to slide back and forth so the camera position can be adjusted. Once the camera is set the tube can be locked in place using two bolts and butterfly nuts.

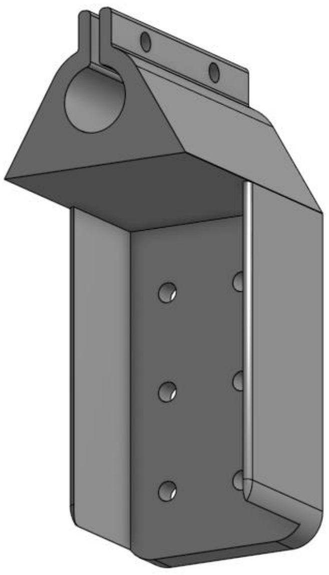


Figure 84 : 3D view of the pole-tube connector

11.7.6 END CAP

This 3D printed part will be placed at the end of the steel tube. The camera will be screwed on it.

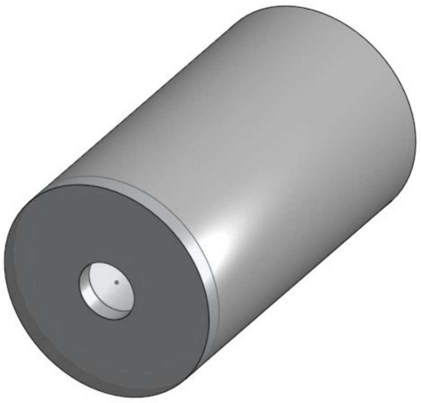


Figure 85 : 3D view of the end cap