Development of a Tool for Rider Posture Tracking on a Motorcycle

Report Nr. 686/18
Editor: Pau Gil Barbeta | 2462837
Advisor: Marius Hofmann, M. Sc.

Pau Gil Barbeta
matriculation number: 2462837
course: Mechanical and Process Engineering

Thesis Nr. 686/18
topic: Development of a Tool for Rider Posture Tracking on a Motorcycle

submitted: 14. September 2018

Technische Universität Darmstadt
Fachgebiet Fahrzeugtechnik
Prof. Dr. rer. nat. Hermann Winner
Otto-Berndt-Straße 2
64287 Darmstadt

TECHNISCHE UNIVERSITÄT DARMSTADT

Masterthesis Nr. 686/18 im Studiengang Mechanical and Process Engineering (30 CP)

von Pau Gil

Beginn: 05.03.2018
Zwischenkolloquium: 16.05.2018
Ende: ~~05~~ 14.09.2018

Thema: **Entwicklung eines Tools zur Fahrerhaltungsbestimmung auf dem Motorrad**

Topic: ***Development of a Tool for Rider Posture Tracking on a Motorcycle***

Fachgebiet Fahrzeugtechnik

FZD

Prof. Dr. rer. nat. Hermann Winner

Otto-Berndt-Straße 2
64287 Darmstadt

Bearbeiter:
Marius Hofmann, M.Sc.
Tel. +49 6151 16 - 24236
Fax +49 6151 16 - 24205
hofmann@fzd.tu-darmstadt.de
www.fahrzeugtechnik-darmstadt.de

Datum
01.03.2018

The Institute of Automotive Engineering at the TU Darmstadt (FZD) is doing research on the topic of driver state evaluation in a cooperation with the Honda Research Institute Europe. Therefor a measurement vehicle is build up, which should be able to determine the current position and posture of the rider, measure his gaze direction and capture data about the vehicle dynamics and the surrounding. In the future, the driver state and behavior in different situations will be researched. Target of this thesis is to build up a system that could robustly track the rider's movements and his position with help of inertial measurement units, which are integrated in a motion tracking-suite, the processed output of the suite and further equipment on the motorcycle.

The following outcomes are expected:

- selection of fitting algorithms based on the state of the art
- an operational tool to track the driver's posture in post-processing
  - implementation in python and/or MATLAB
  - the data is synched with the other sensors on the motorcycle
  - the tool focuses on the upper body segments
- analysis of the accuracy and reliability of the developed solution regarding angular and absolute positions
- documentation of the results

CP	Dauer Zwischenkolloq (Vortrag / gesamt)	Dauer Abschlusskolloq (Vortrag / gesamt)
< 18	5 - 7 / 15 min	15 / 45 min
> 18	10 / 20 min	20 / 60 min

TECHNISCHE UNIVERSITÄT DARMSTADT

The thesis stays property of the institute. The student is informed about the institute's handout.

Prof. Dr. rer. nat. Hermann Winner

Betreuer: Marius Hofmann

Declaration of Authorship

Thesis Statement pursuant to § 23 paragraph 7 of APB TU Darmstadt

I herewith formally declare that I, Pau Gil Barbeta, have written the submitted thesis independently. I did not use any outside support except for the quoted literature and other sources mentioned in the paper. I clearly marked and separately listed all of the literature and all of the other sources which I employed when producing this academic work, either literally or in content. This thesis has not been handed in or published before in the same or similar form.

I am aware, that in case of an attempt at deception based on plagiarism (§38 Abs. 2 APB), the thesis would be graded with 5,0 and counted as one failed examination attempt. The thesis may only be repeated once.

In the submitted thesis the written copies and the electronic version for archiving are identical in content.

For a thesis of the Department of Architecture, the submitted electronic version corresponds to the presented model and the submitted architectural plans.

Datum / Date: 14. September 2018

Unterschrift/Signature:

Abstract

Safety has become one of the most important fields to develop for the vehicle industry. Both in the automotive industry and others, new features for traditional vehicles are being developed to increase the safety and comfort of passengers, such as autonomous driving. In the case of the motorcycle industry, autonomous driving is further away from implementation, so this sector is developing other ways to increase the safety of its vehicles.

At the Technische Universität Darmstadt, a test motorcycle is setup to better understand how motorcycle riders behave in different situations. The system is based on different sensors and cameras to capture the status of both the motorcycle and the rider in each of these situations.

One of the objectives of this project is to develop a tool that fits the TU Darmstadt project to estimate the position of the rider through fish-eye cameras located on the back and front of the motorcycle. For this purpose, different methods of object detection through image processing have been evaluated, taking into account the potential of each and the final objective of this tool. The results of these different tests have resulted in a system composed of small markers that, incorporated on the jacket of the rider, allow to extract the position of each one of the locations.

For this, a tool based on three stages has been developed. The first is to extract the parameters of the camera to be able to analyse the images afterwards. The second consists of compensating the distortion caused by the fisheye cameras, using the parameters extracted in the first stage. In the last stage, the tool allows to analyse the position and orientation of the markers in the image and converting it into the position it occupies in the motorcycle coordinates.

This tool has been tested with different tests to analyse its accuracy. Analysing the results, it is determined that this tool is a valid approach to satisfy the purposes of the project of the Technische Univertität Darmstadt. The precision, although variant in each of the tests, is sufficiently high to satisfy its objective in the ranges in which the tool has been configured.

Finally the tool is tested under the conditions for which it was designed, and the possible aspects to be improved are detailed for its subsequent implementation.

Table of content

Symbol and index directory

Greek letters:

symbol	unit	description
γ	-	skew
α	º, rad	angle

Indexes:

symbol	description
n	number
x,y,z	coordinates
X,Y,Z	axes
R	rotation matrix
$\vec{t}$	translation vector
f_x, f_y	focus length
c_x, c_y	optical centres
k_i	distortion parameters
$x_{meas}, y_{meas}, z_{meas}$	measured positions in the X, Y and Z axis
x_c, y_c, z_c	estimated coordinates in the X, Y and Z
$Roll_{meas}$	measured rotation around the X axis
$Pitch_{meas}$	measured rotation around the Y axis
Yaw_{meas}	measured rotation around the Z axis
$Roll_c$	estimated rotation around the X axis
$Pitch_c$	estimated rotation around the Y axis
Yaw_{mc}	estimated rotation around the Z axis
$diff$	absolute value of the difference between the measured value and the estimated

List of abbreviation

ARAS	Advanced Rider Assistant Systems
GPS	Global Positioning System
CPU	Central processing unit
GPU	Graphics Processing Unit
OS	Open Source
OpenCV	Open Source Computer Vision Library
OCamCalib	Omnidirectional camera calibration implementation
FPS	Frames per second

List of figures

List of tables

1. Introduction

1.1 Motivation

Safety has become a main target to develop in the automotive companies. Some technologies, for example autonomous driving, need to have a strong security system, otherwise it could be dangerous for the passengers. There are several technologies in car safety, but the most dangerous road vehicle are with no doubt motorbikes.

ARAS are more difficult to develop on motorbikes, mainly due to the changing position of the center of weight of the motorbike and the rider as well. However, the mobility of this kind of vehicles and the lack of space in the cities will make this technology a field to investigate. In another hand the safety of the rider is always a field to improve, consequently it should exist a tool to improve the safety of the rider. One possibility could be generating a database of the movements of motorbike riders against the different situations on the road.

Extracting the pose data of the rider gives information about how riders react to different situations, and training an algorithm with these cases, a predictive pose estimator can be developed. This technology could help improving the safety of motorbikes. Overall, an easy to use tool for pose estimation is crucial to understand the behaviors of motorcyclists in different situations that may appear on the road.

Cameras offer a rich source of visual information, which can be processed in real-time thanks to the recent advances in computing hardware. In modern autos, already multiple cameras are used for advanced rider assistant system such as lane detection, traffic light detection, and recognition of the movements of the other vehicles on the roads. Applying tools based on image recognition may help the motorcycle industry to develop useful tools for the riders.

Tracking the position of the rider is a part of a more ambitious project at Technische Universität Darmstadt. There a research motorcycle is setup that collects several data about the state of the motorbike and its rider

Figure 1: Measurement system *Hofmann, Marius (2018)*

The measurement system shown in the figure 1, is composed of different sensors, a tracking suit, a GPS and a front camera mounted on the motorbike Honda CBR 650 F. The goal of the project is to understand the parameters of the motorbike in front of the different scenarios on the road. Some of these parameters relate the position of the motorcycle parts, such as the lean angle, the steering angle, the pitch of the spring and some inertial measurements. Others, describe the speed of the motorcycle, for example the wheel speed or the engine speed. The camera records in front of the motorcycle, to recognize what the rider is facing. A tracking suit was used to track the position of the rider, but due to the high dynamics in a motorcycle, the results were not accurate enough. Developing a tool for this project will make the measurement system complete, allowing to relate all the different sorts of data.

1.2 Goal

The goal of this master's thesis is to create a tool to track the movement of the rider on a motorcycle integrated into the Honda CBR 650 F of the Technische Universität Darmstadt for further usage.

The aim of this project is to provide an easy to set up tool written in Python to analyze those motorcyclist views and to extract the movements of the rider in motorcycle coordinates.

Another of the objectives is due to the first goal. To develop a tool of these characteristics, first a satisfactory research of the different technologies, which can be used in this project, must be done.

Once developed the tool, testing the accuracy of the tool will be of great importance to demonstrate its usefulness.

Another goal of this project is also to provide the basis for a further development of these types of technologies.

1.3 Thesis structure

This thesis is structured as follows: After the current introduction, the essentials of image processing technologies, camera models, coordinate systems, camera calibration, and software developing methods will be explained. Afterwards, chapter three will illustrate which of the technologies are finally used and will cover the implementation process made for this thesis in order to calibrate the camera, undistorting the video recordings and tracking the pose of the rider. Afterwards, the results will be evaluated, and eventually, in the last chapter the achieved results and possible optimizations will be discussed.

2. State of art

Several types of technologies are nowadays available to satisfy the purpose of this project. Hence, in this chapter some of the found technologies to perform this project are explained. Due to the reduced space on a motorbike, the aim to perform an algorithm which is easy to implement and due to the available tools for this project, the technologies will be based on image processing.

In this chapter are reviewed the main and most used technologies applicable for the task. The basic knowledge for the further progressions of this thesis is provided in this chapter, starting with a review about pose tracking technologies, and the basics of camera calibrations. An overview of the most used software development methods completes this chapter.

2.1 Haar Cascade Classifier

Haar Cascade classifier consist of a machine learning based algorithm to detect objects, persons, parts of the body, and anything that can be distinguished in an image. The algorithm can be trained with any figure that can be recognised. Several types of object detections are developed based on this algorithm, such as those developed by Wilson[1], Soo[2], Lienhart[3] and Viola[4].

The cascade classifier consist of a list of stages, where each stage consist of a list of weak learners. The system detects objects in question by moving features over the image. In each stage of the classifier labels, the specific region is defined by the current location of the window as either positive or negative. Positive means that the specified object was detected in the current region, and negative means the opposite. If the labelling yields a negative result, then the classification of the specific region is complete and the location of the classifier is moved to the next location. On the other hand, if the classifier yields a positive result, the specified region moves to the next step of the classification.

The classifier gives a final verdict of positive or negative when all the stages are completed, resulting as the specified object either was or was not found in the image. Until the final stage of the classification, it cannot determine if the object is or is not in the image, due to the possibility of a false positive or a false negative in the detection. False positive means that the labelling process falsely determines, that the object is located in the image. A false positive occurs when the process is unable to detect the object in the image and the region was not correctly classified.

In order to prevent both false positives and false negatives, each stage of the classifier must have a low false negative rate. If the image is classified as negative, the branch stops. This process is shown in the figure 2. However, each stage can allow a relatively high false positive rate. If the algorithm detects the region as a false positive, it still have the opportunity to correct this mistake in the next stages of the classifier.

[1] Wilson, P. I.; Dr. John Fernandez: Facial feature detection using Haar classifiers (2006).

[2] Soo, S.: Object detection using Haar-cascade Classifier (2015).

[3] Lienhart, R.; Maydt, J.: An extended set of Haar-like features for rapid object detection (2002).

[4] Viola, P.; Jones, M.: Rapid object detection using a boosted cascade of simple features (2001).

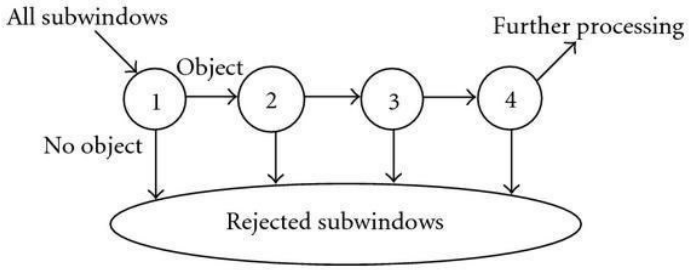

Figure 2: Stages of cascade classifier

The core of the Haar classifier object detection is the Haar-like features. These features, rather than using the intensity values of each pixel, use the change in the contrast values between consequent groups of pixels. These values are used to determine light or dark areas between adjacent groups. These features shown in the figure 3, can be scaled to detect more accurately the region where the object remains or to detect different sizes of the object.

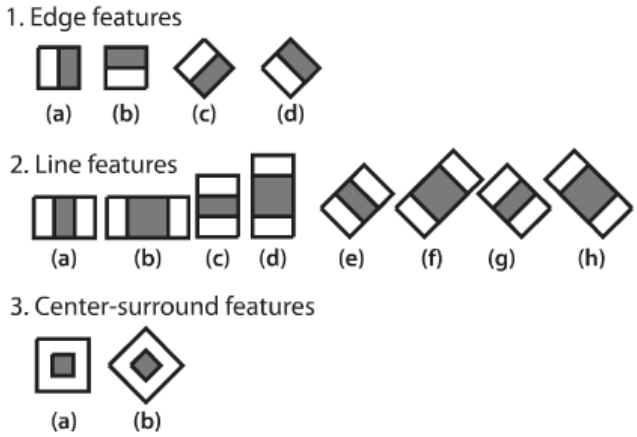

Figure 3: Common Haar-like features

These rectangular features of the image are calculated using an intermediate representation of the image, called integral image. Integral image consist in an array, which contains the sums of the pixels intensity values located to the left of the pixel and directly above the pixel location. The rotated features require another intermediate representation called rotated integral image or rotated sum auxiliary image.

Calculating a feature is extremely efficient and fast. However, calculating all the 180000 features contained within a 24x24 sub-image is not practical. Fortunately, only a tiny fraction of those are necessary to determine if a sub-image potentially contains or does not contain the desired object. The goal is to eliminate a substantial amount of sub-images that do not contain the defined object. Furthermore, boosting methods can be used to make the process more efficient and fast, such as: Discrete Adaboost, Real Adaboost, Gentle Adaboost or Logiboost. Adaboost algorithms basically gives a value to each part of the image. If the sub-region is correctly predicted it increases its value, otherwise it decreases it. It repeat these steps until all the regions are properly predicted.

The algorithm is trained with libraries of positives and negatives images. Positives means that contain the defined object, and negatives means the opposite. There are several pre-trained open-source models that make the detection process much easier to apply or develop for a specific purpose. These models can also contain thousands of positives and negatives with the specific object or figure to be detected.

Pre trained classifiers are available free of charge and each of them are performed to detect different objects. The fact of having available libraries make Haar classifier easy to use, but using already trained models means that the efficiency of the algorithm must be tested. If the accuracy of the detection is not high enough, there is always the option of training it again.

The aim of this project is detecting several parts of the rider body. Using this method, the position of the different parts in the body must be changed into real word coordinates. This could be possible extracting the parameters of the camera and extrapolating the coordinates in the image into 3D coordinates.

However, there are different methods to detect 3D coordinates which have more fiducial points to make the detection more accurate. Haar Cascade algorithm performs very good detecting objects, it is fast, easy to use and precise. Nevertheless, to detect 3D coordinates it is not a precise method, there can be a lots of imprecisions due to the different positions and rotations of the body parts of the rider that could make the algorithm estimate wrongly the coordinates of the targets.

With all these reasons, it was decided not to use this algorithm to detect the selected body parts of the rider. However, some of the methodology used in this algorithm was useful to develop the software of the final detection tool.

2.2 Tensorflow pose estimation

Tensorflow is an open source software library for numerical computation using data flow graph programming across a wide range of tasks. It is used as a math library and for different machine learning applications, for example neural networks and pose estimation. The graph nodes represent mathematical operations, while the graph edges represent the multidimensional data arrays, also called tensors, which flow between them. This type of flexible architecture allows Tensorflow to deploy computation in one or more CPUs or GPUs, on a server, a mobile phone, and other devices without rewriting the code. Tensorflow provides functionalities for C++, Python, and other programming languages. This software was developed by the Google Brain team for their own purposes but released as an open-source in 2015 under the Apache 2.0 open source license.

Due to the new open source philosophy of Tensorflow, several studies about how to understand the software and how to develop it can be easily found. Some developed software based on Tensorflow

are also published as an upgrade of the software itself, such as those developed by Martinez[5], Abadi[6], Andriluka[7]and Insafutdinov[8].

The Tensorflow pose estimation works as a convolutional neural network[9], a class of deep, feed-forward artificial networks. The pose estimation consist of a sequence of convolutional networks that repeatedly produce 2D belief maps for the location of each part. At each stage, image features and beliefs maps produced by the previous stages are used as an input. The belief map provides the next stages about spatial uncertainty for each part, allowing the software to learn spatial models of the relationships between parts and create a library including all the feedback obtained in the previous stages.

Pre-trained networks allow the user to use this software for a specific application without the need of creating a library, or simply training it again to change the purpose of the software. For example, as a pose estimation library, MPII data set consist of more than 25000 images, which contain 40000 annotated persons. This library was designed for 2D pose estimation and contain more than 410 activities. The architecture of convolutional pose machines, basically, consist of a sequence of multi-class predictors, which are trained to detect the position of each part in each level of the hierarchy. In each stage, the classifiers (g_t) predict beliefs for assigning a location to each part , based on features extracted from the image at the location and contextual information from the preceding classifier in the neighbourhood in each stage.

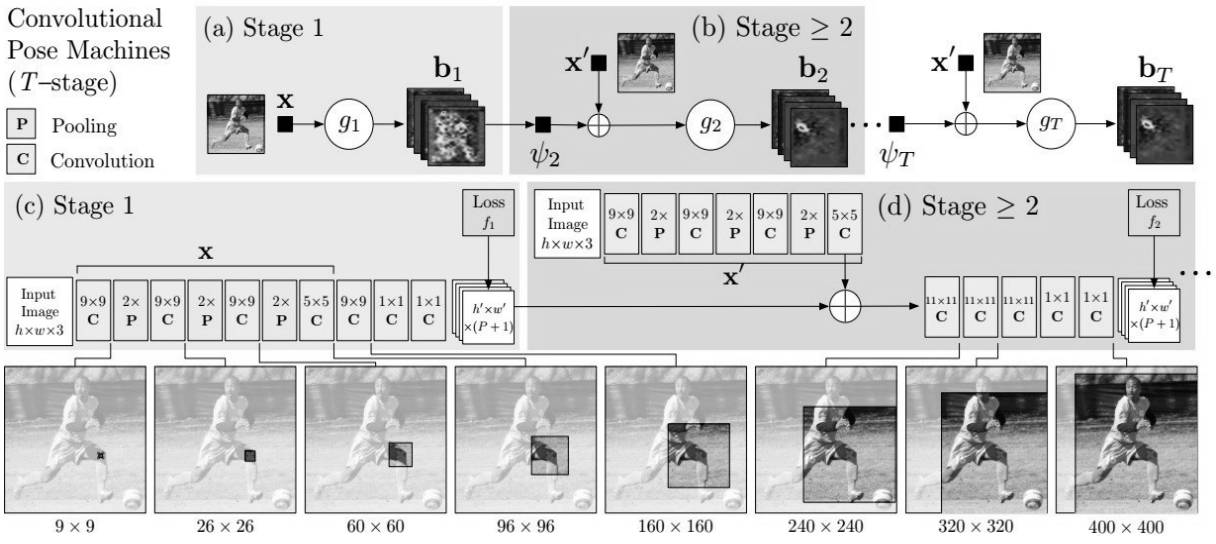

Figure 4: Architecture of convolutional pose machines and receptive fields.

[5] Martinez, J. et al.: A simple yet effective baseline for 3d human pose estimation (2017).

[6] Abadi, M. et al.: TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems (3/14/2016).

[7] Insafutdinov, E. et al.: ArtTrack: Articulated Multi-person Tracking in the Wild (2016).

[8] Insafutdinov, E. et al.: DeeperCut: A Deeper, Stronger, and Faster Multi-Person Pose Estimation Model (2016).

[9] Wei, S.-E. et al.: Convolutional Pose Machines (1/30/2016).

In the figure 4, the pose machine is shown in insets (a) and (b) and the corresponding convolutional network is shown in the insets (c) and (d), which operate with the preceding. The insets (a) and (b) only operate on image evidence in the first stage. The insets (b) and (c) show the architecture from the subsequent stages, from stage 2 to T, which operate with both image evidences and belief maps from preceding stages. The network is also supervised after each stage with loss layers, which prevent vanishing gradients during the algorithm training. Below the architecture model, it is shown the further is the algorithm trained, wider is its range to detect, in this case the knee.

Because of Tensorflow pose estimation is an open source code, there are several types of architectures working on, depending only on who built and for which purpose. However, in addition to the input and output layers, usually there in a convolutional there are also the named hidden layers. Typically, the hidden layers consist of: convolutional layers, pooling layers, fully connected layers and normalization layers. Hidden layers are combined to extract the information from the image and generalize for future treatments or recognitions.

2.2.1 Convolutional layers

The convolutional layers are those which apply filters known as convolutional kernels to the images, extracting features. The features are basically representations of the original input and are used by the following layers as an input. Convolutional layers are shown in the figure 4, and are the core of any convolutional neural network. Kernel is based on the idea how the human brain works. The human brain processes the images into fragments called receptive fields. Kernel does the equal, defines a region and only considers this area for the calculation, it is a local or neighbourhood method. The kernel slides through the image, with a specified stride. If the kernel is an edge detector, the output image would be an image of edges, where the edges were the detected feature. Image kernel considers a weight for every single part depending on its learnings.

Different kernels are applied to the image at the same time, generating new images called feature maps. The weights in the kernel are trained in the training phase, and define the future feature transformation for each of the future maps.

2.2.2 Pooling layers

Polling is a form of non-linear down sampling. Pooling reduce computational costs and increase the locally independence of features found. There are several non-linear functions to apply the pooling, among "*max pooling*" is the most common. Max pooling divides the input image into a set of non-overlapping rectangles and, for each sub-region, outputs the maximum of the region. The pooling layers are used to progressively reduce the spatial size of the image, to reduce the number of parameters and consequently the amount of computation in the network. As shown in the figure 4, it is common to periodically insert pooling layers between the convolutional layers.

Due to the aggressive size reduction of the representation, the trend is towards using smaller filters or discarding the pooling layers altogether. Pooling layers are a main point in convolutional neural networks, reducing the computation amount

2.2.3 Fully connected layers

After several convolutional and pooling layers, the reasoning in the convolutional neural networks is made by fully connected layers. It works as a convolutional layer with a 1x1 kernel. The fully connected layer gathers the features from the previous layers and regresses and classifies based on the most abstract features. Usually there are several of these layers connected in succession to make the classifier more complex and complete.

The fully connected layer performs connecting each neuron from one layer to every other neuron in the next layer. All the connection between neurons can be trained, hence this layer is usually the responsible for the most of free parameters and afterwards the training duration.

2.2.4 Loss layers

The loss layer, as shown in the figure 4, determines how the training penalizes the deviation between the predicted results and the true labels. Normally this layer is located at the end, as the final layer. Different loss functions are usually applied to perform different tasks.

2.3 ArUco markers

There are several types of OS libraries for detecting fiducial markers in images, such as ArUco markers, ArUco boards, CharUco markers, and diamond marker. However, due to the space requirements of this project, the marker that fit it best is the ArUco. Notwithstanding its size, it gives enough accuracy for the purpose itself with a compact format that will help placing it in the different spots it will be used.

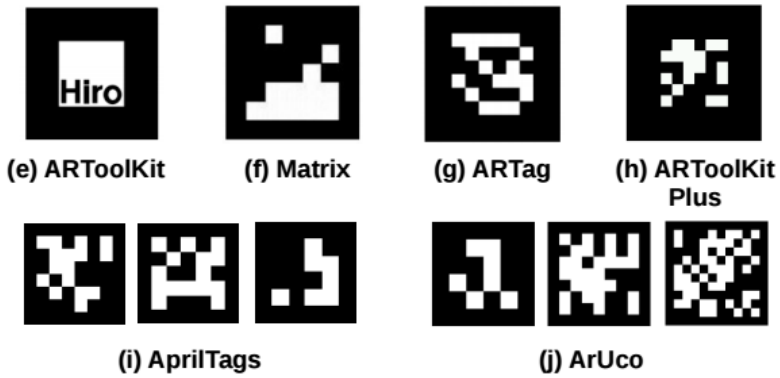

Figure 5: Artificial landmarks

Owing to it is an open source, there is a lot of information about these technologies. Latest researches are focused on how to improve the quality of the detection in different light, distance or filtering conditions, such as research projects from Garrido[10] , Garrido-Jurado[11], Muñoz[12] and Agnus[13].

There are different types of markers, each of every type belongs to a dictionary. These squared markers are composed by a wide black border and an inner region that encodes a binary pattern. This pattern is unique and identifies marker with an id. The black border facilitates its fast detection in the image and the binary codification allows its identification and the application of error detection and correction technique.

The different dictionaries basically are differentiated for the number of bits that compose the inner matrix in the marker. The more bits, the more words in the dictionary, and the smaller the chance of confusion. However, more beats need more resolution for the correct detection of the marker. In this project it is used the 6x6_250 dictionary. This means that the size of the markers is 6x6 bits, and the total number of markers in this one is 250 markers.

The markers can be used as 3D landmarks for camera pose estimation, however, in this project will be used as a points of reference to detect the position of the markers. The following image shows the coordinate system employed in the library used.

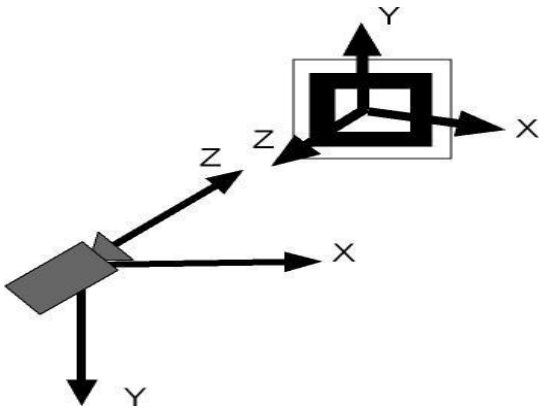

Figure 6: Marker coordinate system

2.3.1 Marker detection

Each marker is a vector of 4 points that represent the corners of the marker, a unique id, its size in meters, and the translation and rotation that relates the centre of the marker and the camera location.

The detection process of ArUco markers can be defined for the following steps:

1. Apply an adaptive Thresholding so as to obtain the marker borders. The adaptive Threshold method consist in classifying and changing each pixel into either black or white. After using

[10] Garrido-Jurado, S. et al.: Automatic generation and detection of highly reliable fiducial markers under occlusion (2014).

[11] Garrido-Jurado, S. et al.: Generation of fiducial marker dictionaries using Mixed Integer Linear Programming (2016).

[12] Muñoz-Salinas, R. et al.: Mapping and localization from planar markers (2018).

[13] Agnus, V. et al.: Illumination Independent and Accurate Marker Tracking Using Cross-Ratio Invariance (2015).

this filter, we obtain a processed image with a high contrast between the light and dark areas. This method makes the detection more efficient. In the basic Thresholding method, the user sets the ranges out to consider one pixel black or white. In the adaptive, the function analyse small parts of the image and sets the ranges out for those regions according on the light conditions of each ones. Thereby, the limits are decided according to the needs of each region to have more chances to detect the marker properly.

2. Find contours. With specific functions the contours can be detected. A problem that may occur is that the function detects not only the real markers. Hence, the rest of the process aims to filter those unwanted borders.
3. Remove borders with small amounts of points that probably could be not markers.
4. Polygonal approximation of contour and keep the concave contours with exactly 4 corners.
5. Sort corners in anti-clockwise direction.
6. Remove too close rectangles. This step is required because the adaptive threshold normally detects the internal and external part of the borders of the marker. Hence the most external border is kept.
7. Marker identification:
    a. Remove the projection perspective so as to obtain a frontal view of the rectangle area using a homography, a simple method to extract a frontal view of the region.
    b. Threshold the area using Otsu. Otsu's algorithms assumes a bimodal distribution and finds the threshold that maximizes the extra-class variance while keeping a low intra-class variance.
    c. Identification of the internal code. If it is a marker, then it has an internal code. In this case, the marker is divided in a 6x6 grid, of which the internal 5x5 cells contains id information. The rest correspond to the external black border. Here, first the external black border presence is checked. Afterwards, the 5x5 cells are read in all the possible orientations to check if it is a valid marker id.
    d. For the valid markers, using subpixel interpolation, the corners are refined.

2.3.2 Pose estimation

There are different paths to estimate the pose of a marker. Some of are based only on finding the borders, others come from the ArUco module. Hence, they need the pattern of the marker to detect it.

Once the markers are detected, their corners and their identifiers, it is possible to obtain the camera pose from the marker. In this project this process will be used to detect the position of the marker from the camera, not vice versa.

To perform the marker pose estimation it is first needed to know the calibration parameters of the camera. Those include the camera matrix and the distortion coefficients, both together determine the relation between the natural units of the camera. The camera matrix represents the intrinsic parameters, this means a matrix of 3x3 elements with the focal distances and the camera centre coordinates.

The distortion coordinates define itself with at least five element vector which models the distortion produced by the camera.

Once the camera parameters are known, the markers position can be detected. The camera pose respect to a marker is the 3D transformation from the marker coordinate system to the camera coordinate system. It is specified by a rotation and a translation vector. However, this project is interested in finding the position of the marker from the camera. Hence, it will require an axis transformation.

Initially the markers axis are defined as in the following picture. The marker coordinate system is assumed and placed in the center of the marker with the Z axis pointing out of it. There are some functions in the ArUco module that represent the axis in the pictures. With the shown axis, the pose estimation can be checked out in the real images.

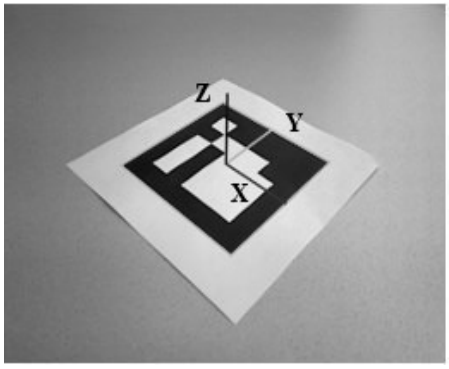

Figure 7: Representation of the axis on a detected ArUco marker

2.4 Camera calibration

All these methods explained before, need indeed a camera calibration. Otherwise the pose estimation cannot be properly done. There are different methods to find the camera calibration parameters, such as using a chessboard pattern, with some marker-boards, and some others. Some of these methods are explained and developed by Zhang[14], Hartley[15].

2.4.1 Coordinate system

Working with multiple cameras involves multiple coordinate systems. The axis need to be transformed to each other, or into a new reference coordinate system. Some of the used coordinate systems are used by the pinhole camera model [2.4.2] . The three-dimensional camera coordinate frame and the image frame referencing to pixel coordinates. As shown in the next images, the camera coordinate system viewpoint with the z-axis pointing away from the camera in the direction of the view is the origin of the camera system. Captured images from the sensors of the camera use the image coordinate frame. Its center is at the top left of the position of the image and references to pixel coordinates.

[14] Zhang, Z.: A flexible new technique for camera calibration (2000).

[15] Hartley, R. I.: Theory and Practice of Projective Rectification (1999).

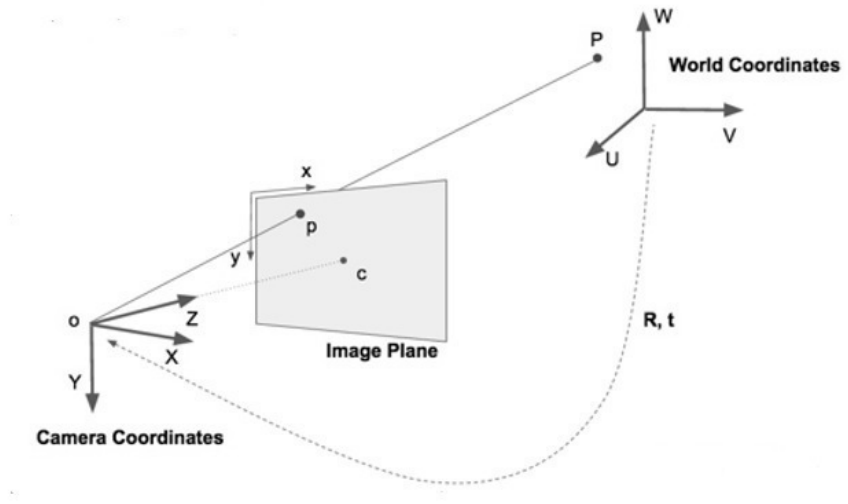

Figure 8: Coordinate systems [16]

As shown in the image, the world coordinates are represented by (u, v, w), the camera coordinates by (X, Y, Z), and the image coordinates by (x, y). Hence, the axes have been defined in this project to make it as easy as possible to evaluate the results. The coordinate system then is configured according to the next figure 9. As shown, the y axis is inverted to make it more understandable.

Figure 9: Coordinate system in the motorbike

Coordinate representation of n-dimensional space, representing lines in $(n + 1)$-dimensional space by adding one as a constant to the vector can be defined as homogeneous coordinates.

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \rightarrow \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \tag{2-1}$$

[16] Mallick, S.: Head Pose Estimation using OpenCV (2016).

The combination of rotations and translations are defined as $T_{A\to B}$. Usually, for transformation from n-dimensional coordinate system A to d-dimensional coordinate system B, these annotations can be used, and they define a transformation matrix, which consist of a rotation matrix R and a translation vector $\vec{t}$ used to transform homogeneous coordinates.

The translation vector represents the position of the camera, and the rotation matrix describes the yaw, pitch and roll angles movements[17].

$$T_{A\to B} = \begin{pmatrix} & R & & \vec{t} \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2-2}$$

2.4.2 Pinhole camera model

A simple model to describe the image properties of cameras is the pinhole camera model. This model is explained in detail in the works of Kühling[18], Muñoz[19],Mallon[20] and Zhang[21]. In this model, light is envisioned as entering from the scene or distant object, but only a single ray enters from a particular point, which is projected onto an imaging surface.

A single parameter of the camera, the focus length, gives the size of the image relative to the distant object. The distance from the pinhole aperture to the screen in this idealized model is precisely the focus length(f_x, f_y), .

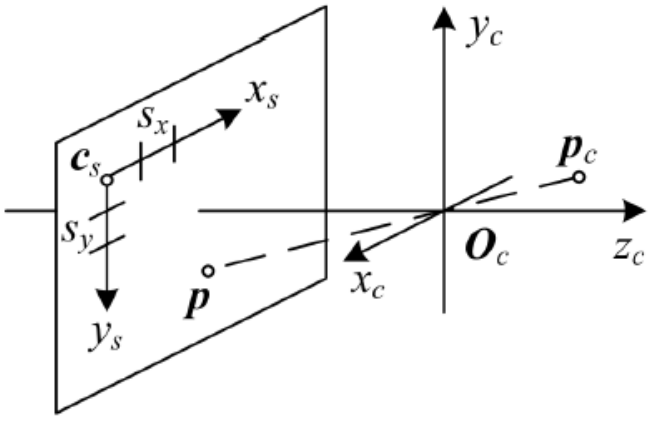

Figure 10: Pinhole camera model

[17] Ang, M.; Tourassis, V.: Singularities of Euler and Roll-Pitch-Yaw Representations (1987).

[18] Kühling, C.: Masterarbeit am Institut für Informatik, Fisheye Camera System Calibration for Automotive Applications (2017).

[19] Muñoz-Salinas, R. et al.: Mapping and localization from planar markers (2018).

[20] Mallon, J.; Whelan, P. F.: Precise radial un-distortion of images (2004).

[21] Zhang, Z.: A flexible new technique for camera calibration (2000).

The picture 10 illustrates the basic structure of the pinhole camera model. It shows a 3D-Point P_c, represented in the camera coordinate system with origin in O_c and the axes x_c, y_c and z_c. This point is transformed onto the image sensor plane by a projective transformation. The origin C_s and the scaling factors s_x and s_y determine the projection of the point on the sensor plane. The variable z_c represents the optical axis of the camera.

$$x' = \frac{X}{Z} \tag{2-3}$$

$$y' = \frac{Y}{Z} \tag{2-4}$$

In order to transform points from the real word to camera coordinate system the function (2-2) is used. If the relationship between the point from the real word and its image projection is analysed, it determines the following equation (2-5).

$$K = s \cdot \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} f_x & \gamma & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} r_{11} & r_{12} & r_{13} & \overrightarrow{t_1} \\ r_{21} & r_{22} & r_{23} & \overrightarrow{t_2} \\ r_{31} & r_{32} & r_{33} & \overrightarrow{t_3} \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{2-5}$$

The K matrix represents the camera matrix, it is also called intrinsic camera matrix and contain the intrinsic camera parameters. These are needed to determine the pixel coordinates of the projection point (u, v) and *s* represent a scalar scaling factor. In addition to the intrinsic parameters, the already mentioned rotation matrix R and translation vector $\vec{t}$. Both together represent the extrinsic parameters of the camera, and represent the orientation and the position of the camera in the world coordinate system. These parameters are explained in detail in the next sections (2.4.3), (2.4.4).

2.4.3 Intrinsic camera calibration

The aim of the project is to incorporate the cameras in some parts of the motorcycle. Due to the short distances between the rider and the cameras, wide-angle cameras or fisheye lenses will be used to capture all the possible movements of the rider. Basically, the advantage of using wide-angle cameras is the larger field of view, but also there are the disadvantages of distortion. This last phenomenon makes straight lines appear as curves lines instead, and impacts on the resolution of the image.

Intrinsic parameters are specific to a camera and are included in the K matrix, also known as intrinsic camera matrix. It includes information such as focal lengths (f_x, f_y), optical centers (c_x, c_y) and the skew (γ) betwenn the sensor and the principal point .All combined in the camera matrix determine the intersection of the camera Z-axis with the viewing plane. It depends on the camera only, once calculated, it can be stored for future purposes if the lens are not changed. The skew γ is usually

negligible in current cameras, for that reason it will be assumed to be zero. Usually the focal lengths are simplified as $f_x = f_y = f$ due to the assumption of square pixels. This means that in most of the cases focal length is suitable.

$$K_{\text{cam}} = \begin{bmatrix} f_x & \gamma & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2-6}$$

In the figure 11 two common types of radial distortion are displayed: the barrel distortion and the pincushion distortion. Radial distortion occurs when light rays bend more near the edges of lens than they do at its optical centre, the smaller the lens, the greater the distortion. These types are the most occurring distortions besides slight tangential distortion.

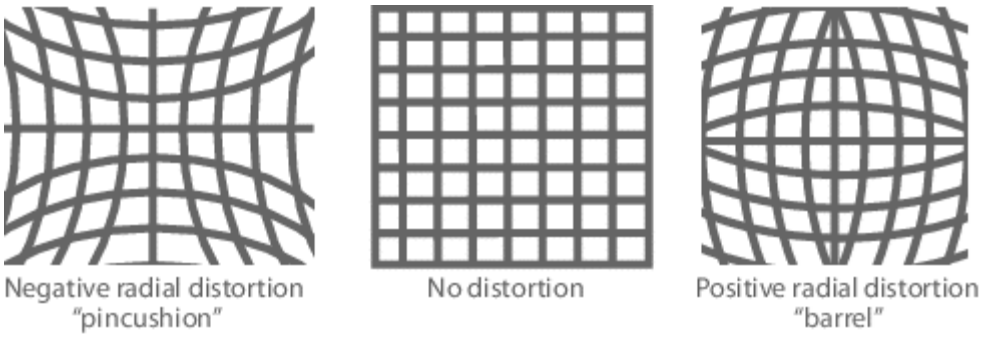

Figure 11: Two types of radial distortion

Hence new camera models that are able to handle this distortion are required. These models use parameters to describe distortions. The process to estimate the set of intrinsic parameters in a camera model is called intrinsic camera calibration. Following, it will be discussed three different options to achieve the intrinsic camera parameters and inspired by the projects of Söderroos[22] and Hofmann[23]

2.4.3.1 MATLAB Mei calibration toolbox

Mei developed a toolbox in MATLAB to provide the camera extrinsic, intrinsic and distortion parameters. This toolbox is capable of calibrating and evaluating the parameters from different types of cameras, such as parabolic, catadioptric and parabolic.

This toolbox has peculiarity, it is based on a specific projection model, Mei projection model[24]. This model is an extension of Barreto[25]. This method project the world points onto unit sphere before boing projected on the normalized plane. This capability allow the centre of the projection to be shifted on

[22] Söderroos, A.: Fisheye Camera Calibration and Image Stitching for Automotive Applications (2015).

[23] Hofmann, P.: Master's thesis, Object Detection and Tracking with Side Cameras and RADAR in an Automotive Context (2013).

[24] MeiProjectionModel. (2018).

[25] Barreto, J. P.; Araujo, H.: Issues on the geometry of central catadioptric image formation (2001).

the new plane. In the figure 12 it is shown an illustration describing the Mei projection method, and this model can be briefly explained by the following steps:

1. Project a world point Y in the camera coordinates onto the sphere

$$(x_m, y_m, z_m) = \frac{Y}{||Y||} \tag{2-7}$$

2. Change the rest of the points to a new reference frame centred in $C_p = (0,0,\xi)$. Where df is the distance between focal points and $4p$ is the cord through a focus parallel to the conic section directrix.

$$(x_s, y_s, z_s) = (x_m, y_m, z_m) + \xi \tag{2-8}$$

$$\xi = \frac{df}{\sqrt{df^2 + 4p^2}} \tag{2-9}$$

3. Project the points on to normalised plane. In the previous step, the centre has been shifted to ensure that the z_s coordinate is never zero. Thus, previous step makes the method work smoothly in every case.

$$M_u = \left(\frac{x_s}{z_s}, \frac{y_s}{z_s}, 1\right) \tag{2-10}$$

4. Include the radial distortion to the points. k_i are the radial distortion parameters, and *D* is the distortion function. In the distortion function, k_i are the distortion parameters; $k_i = 1, .., 5$ and $p = \sqrt{x^2 + y^2}$.

$$M_d = M_u + D(M_u, k_i) \tag{2-11}$$

$$D(p) = 1 + k_1 p^2 + k_2 p^4 + k_3 p^6 + k_4 p^8 + k_5 p^{10} \tag{2-12}$$

5. Project the points on the image plane using the intrinsic camera matrix *K*, introduced in the beginning of this chapter (1.7.3).

$$P = \begin{pmatrix} f & s & x_c \\ 0 & f & y_c \\ 0 & 0 & 1 \end{pmatrix} m_d \tag{2-13}$$

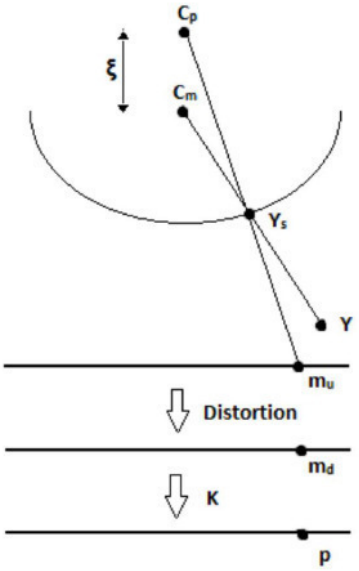

Figure 12: Mei projection model[26]

To use this toolbox is first needed to collect images of a chessboard calibration pattern. This pattern has to be shown in different angles to the camera. The user selects three non-radial points at least, it means not placed on the radii from the image centre, which belong to a line in an image, where the focal length is estimated from. The user also has to select four grid corners to initialize the extrinsic grid parameters. Afterwards, the pattern is projected again on the images and some global minimization can be done while calibrating the intrinsic camera parameters.

A disadvantage of this toolbox could be the need of the user to select the corners in every image, it means that this method does not work automatically. Another disadvantage could be relocating a displaced corner, the toolbox don not allow the user doing it. Whereas selecting the pictures to acquire the camera parameters, the faulty images which work poorly with the corner detection should be removed manually from the calibration dataset.

Finally, a great inconvenience is the requirement of a Matlab charged license to use this tool. Due to all of this obstacles, Mei calibration toolbox is discarded for this project.

2.4.3.2 Scaramuzza OcamCalibToolbox

Davide Scaramuzza[27] [28] developed another toolbox for MATLAB named OCamCalib. The used method is also based on his previous work. This toolbox allows the user to calibrate any omnidirectional cameras, it means any panoramic camera and including fisheye lenses as well. With a chessboard calibration pattern, the intrinsic and the distortion parameters can be obtained.

[26] Söderroos, A.: Fisheye Camera Calibration and Image Stitching for Automotive Applications (2015).

[27] Rufli, M. et al.: Automatic detection of checkerboards on blurred and distorted images (2008).

[28] Scaramuzza, D. et al.: A Toolbox for Easily Calibrating Omnidirectional Cameras (2006).

This toolbox has some similarities with the Mei's toolbox, for example both use a chessboard pattern to acquire the camera. In this method, the user also has to select the images to extract the parameters from. However, there is no need for the user to select the corners in each picture, the software does it automatically. Moreover, if the automatic detection of the corners is faulty, the corners position can be changed manually.

A great disadvantage of this toolbox is, that coordinate system is defined differently regarding not using the standard right-handed coordinate system, hence while working with this toolbox it has to be taken into account when evaluating the results.

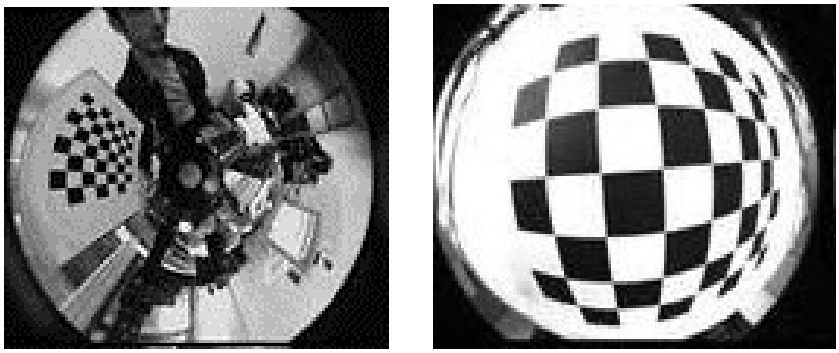

Figure 13: Pictures captured with a catadioptric (left) and a fisheye camera (right)

The simple pinhole camera model, even considering distortions, is not able to model fields of wide angle larger than 180º and problems may occur with much smaller angles. This is the reason why Scaramuzza developed a new camera model which allows higher wide angles, such as mirror lens, wide angle and fisheye lenses.

Distortion can be easily recognised in these types of cameras, in fisheye and wide angle lenses the central part has a higher zoom than the edges, and in the further areas from the centre straight lines become bent, such as the figure 14 shows.

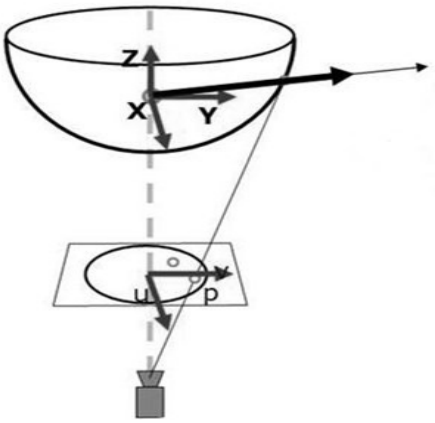

Figure 14: Omnidirectional camera model[29]

[29] Scaramuzza, D. et al.: A Toolbox for Easily Calibrating Omnidirectional Cameras (2006).

This model compute a vector $(x, y, z)^T$ from the single viewpoint to a spherical picture pointing in the direction of the incoming light ray for each pixel position $(u, v)^T$. To represent every camera coordinate point on a specific ray using spherical projection properties the following expression is used:

Where *s* represents a scalar number, and *g(p)* and *p*:

$$g(p) = a_0 + a_1 p \ + a_2 p^2 + a_3 p^3 + .. + a_n p^n \tag{2-14}$$

$$p = \sqrt{u^2 + v^2} \tag{2-15}$$

To projec from the camera coordinates onto an image, the polynomic ecuation (2-14) needs to be solved to find *u* and *v*. This method was not efficient enough, hence Scaramuzza depeloped an inverse Taylor polynomial. The radial positioning on the image plane based on the angle α of a light ray to the z axis is described by it.

$$f(\alpha) = b_0 + b_1 \alpha \ + b_2 \alpha^2 + b_3 \alpha^3 + .. + b_n \alpha^n \tag{2-16}$$

$$r = \sqrt{x^2 + y^2} \tag{2-17}$$

$$\alpha = \arctan\left(\frac{z}{r}\right) \tag{2-18}$$

$$u = \frac{x}{r} f(\alpha) \tag{2-19}$$

$$v = \frac{y}{r} f(\alpha) \tag{2-20}$$

However, leaving aside all these improvements, OcamCalib toolbox is also needs a MATLAB charged license, and there is no completely automatic chance to calibrate the camera with this toolbox, OcamCalib toolbox is also discarded for this project.

2.4.3.3 OpenCV camera calibration

OpenCV[30] stands for Open Source Computer Vision Library. It is a widely used open source library for machine learning and computer vision, and free charged license for both academic and commercial use. This library has C++, Python and Java and supports all of the most used operating systems. It is focused on real-time applications and was designed for computational efficiency.

Specifically, the camera calibration library is based on Brown's[31] distortion model .The coordinates x' and y' coordinates in the ideal pinhole camera coordinates are represented as x'' and y''taking into account the distortion parameters. These distortion parameters are named k_1, k_2,

[30] Bradski, G. R.; Kaehler, A.: Learning OpenCV (2011).

[31] Szeliski, R.: Computer vision (2011).

$k_3$and the tangential distortion coefficients p_1 and p_2. In case of strong distortions, those parameters are not enough and another additional distortion parameters k_4, k_5 and k_6 must be applied.

$$r^2 = x'^2 + y'^2 \tag{2-21}$$

$$x'' = x'(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + 2p_1 x' y' + p_2\left(r^2 + 2x'^2\right) \tag{2-22}$$

$$y'' = y'(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + p_1\left(r^2 + 2y'^2\right) + 2p_2 x' y' \tag{2-23}$$

$$u = f_x \cdot x'' + c_x \tag{2-24}$$

$$v = f_y \cdot y'' + c_y \tag{2-25}$$

If the distortion in is very high, the first term of the first two equations (2-21) and (2-22) are divided by $(1 + k_4 r^2 + k_5 r^4 + k_6 r^6)$. The following picture shows the effect to removing distortion. Points are elliptically displaced according to their distance to the center.

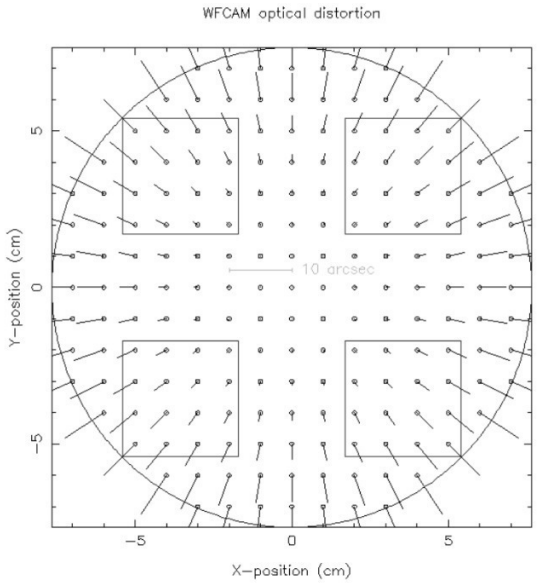

Figure 15: Distortion model[32]

OpenCv[33] has developed this method to work with wide angle and fisheye lenses. For a point P in 3D coordinates, X in the world coordinate system. The rotation matrix corresponding to the rotation vector $\overrightarrow{om}$: $R = Rodrigues(\overrightarrow{om})$[34].Then $\overrightarrow{x_c}$ is the coordinate vector of the point P in the reference frame.

$$\overrightarrow{x_c} = RX + T \tag{2-26}$$

[32] Muñoz-Salinas, R.: Camera Model.

[33] OpenCV: Fisheye camera model.

[34] Cheng, H.; Gupta, K. C.: An Historical Note on Finite Rotations (1989).

P is a 3D point, hence it has three coordinates: $= \overrightarrow{x_{c1}}$, $y = \overrightarrow{x_{c2}}$, $z = \overrightarrow{x_{c3}}$. The projection of the P point in the pinhole projection coordinates is $(a\ b)^T$ where $= x/z$, $b = y/z$, $r^2 = a^2 + b^2$ and $\theta = \arctan(r)$. The fisheye distortion can be describe as:

$$\theta_d = \theta\,(1 - k_1\theta^2 + k_2\theta^4 + k_3\theta^6 + k_4\theta^8) \tag{2-27}$$

Once solved, the distorted point coordinates $(x'\ y')^T$ can be found.

$$x' = \left(\frac{\theta_d}{r}\right) x \tag{2-28}$$

$$y' = \left(\frac{\theta_d}{r}\right) y \tag{2-29}$$

To finally find the position into pixel coordinates $(u\ \ v\ )^T$ that will depend on the optical centres C_x, C_y:

$$u = f_x(x' + \propto y') + C_x \tag{2-30}$$

$$v = f_y yy + C_y \tag{2-31}$$

To acquire the distortion parameters, OpenCV needs a particular method. A known pattern has to be printed, it is not a problem if the pattern is printed in a regular paper. Then it has to be attached to a rigid surface to make it reasonably planar. There are several types of patterns, hence the properties of the patter must be known. Some important information about the pattern is: the size of the squares or circles, the number of squares or circles, and the distances between them.

Figure 16: Calibration pattern. Circles (left), chessboard (Right)

Several images of the pattern in different orientations must be taken to find out the distortion parameters. Usually, the camera is fixed while moving the pattern one around. A specific feature detection algorithm is applied to the images to find the chessboard.

Once the parameters are known, the image can be rectified. A disadvantage of the fisheye cameras is the loss of the outer parts. When the image is rectified, it compensate the distortion curving the image. Thus, the rectified image is not a square and some parts are empty. Therefore the outer part is usually discarded, leaving a full inner rectified rectangle. However, this rectified rectangle is wider than a picture taken by a normal camera.

Figure 17: Fisheye camera image rectification[35]

The image can be cropped as much as wanted depending on the needs of the user. It is not mandatory to eliminate all the empty parts, but usually if some functions are applied to the video after the und-distortion process it may cause some errors.

OpenCV camera calibration is robust, work with multiple calibration patterns and runs completely automatically. Hence is a good method for camera calibration process

2.4.4 Extrinsic camera calibration

As in the intrinsic calibration, the process of acquiring the extrinsic parameters is called extrinsic camera calibration. The extrinsic parameters of a camera consist on the already used rotation matrix R and the translation vector $\vec{t}$. The rotation matric describes the orientation of the camera relative to the world coordinate system. The translation vector describes the position of the camera. The extrinsic parameters consist of six degree of freedom, three coordinates X, Y and Z and three angles α, β, γ. The most known names from these angles come from the established aviation nomenclature: yaw, pitch and roll.

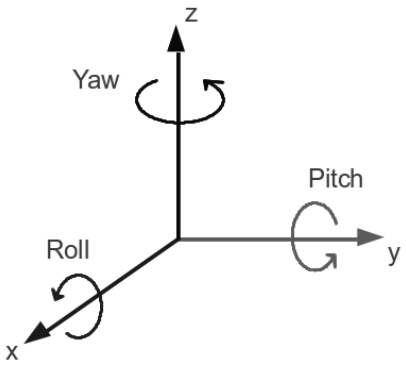

Figure 18: Roll, pitch and yaw

[35] opencv/opencv.

Rotations can be found using these equations:

$$(\text{Roll})\text{Rotation around x axis} \rightarrow R_x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\gamma) & -\sin(\gamma) \\ 0 & \sin(\gamma) & \cos(\gamma) \end{pmatrix} \tag{2-32}$$

$$(\text{Pitch})\text{Rotation around y axis} \rightarrow R_y = \begin{pmatrix} \cos(\beta) & 0 & \sin(\beta) \\ 0 & 1 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) \end{pmatrix} \tag{2-33}$$

$$(\text{Yaw})\text{Rotation around z axis} \rightarrow R_z = \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{2-34}$$

In result:

$$R = R_x R_y R_z \tag{2-35}$$

The fact of working with multiple cameras needs certain considerations. If the position relative to each other change, the extrinsic parameters need to be recalibrated. Fortunately, in this project, the relative position between the cameras will be defined and kept thanks to brackets fixed to the motorcycle. However, the extrinsic parameters must be estimated in relation to the same coordinate system or transform the results to the same coordinates.

In this project, due to the different methods that have been explained previously, the extrinsic parameters will be found only while calibrating the camera. With some of the already explained methods, the rotations and the position relative to the camera can be easily found. Hence further than understanding the concept of the extrinsic parameters it is not necessary to find the extrinsic parameters in every different situation.

2.5 Project model

For each project it has to be followed a specific and different process in each case. The aim of this project, is basically developing a tool as a software to detect the motion of a motorbike rider. Hence, a model based on software developing is followed in every possible step. There are several based on software development models, such as Balaji[36], Mathur[37] and, Nabil[38]

[36] Balaji, S.; Murugaiyan, M.S.: WATEERFALLVs V-MODEL Vs AGILE: A COMPARATIVE STUDY ON SDLC (2014).

[37] Mathur, S.; Malik, S.: Advancements in the V-Model (2010).

[38] Nabil Mohammed, A. M.; Govardhan, A.: Comparison Between Five Models Of Software Engineering (2010).

Two considered models for this project are Waterfall model and V-model. The second represents an upgrade of the first one. The main difference comes at the testing phase. Meanwhile in the Waterfall model all the phases are consecutives and depend only on the previous one, the V-model demonstrates the relationship between each phase of the development life cycle and its associated phase of testing.

2.5.1 Waterfall model

The Waterfall model is relatively a linear sequential approach for some branches of software design. This model is one of the oldest models and is used in many major of the companies. It tends to be not very flexible and not either iterative, because of the progress flows in one single direction. The steps follow a downwards sequence through the stages of: initiation, analysis, design, construction, testing, deployment and maintenance.

The waterfall development model was first originated in the manufacturing and construction industries, where the physical environments made the design changes prohibitively expensive much sooner in the development process. Although there are several variants of the waterfall model, the following figure shows the basic scheme that the waterfall model follows.

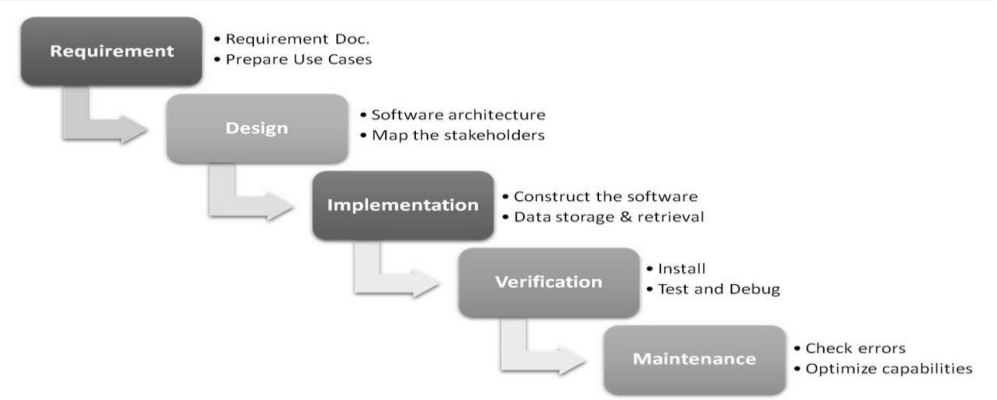

Figure 19: Waterfall model

The following steps shown in the figure 19 detail the steps for using the waterfall model:

1. Requirements: Establishes the components for building the system and the expectations for the software functionality. Includes all the necessary hardware, tools, and other components for the project. For the software functionality, it has to be defined which interactions are needed with other applications, databases, or any other source.
2. Architectural design: Determines the software framework to know the specific requirements. The design determines the major of the components and the interaction between those components. Besides, in this stage the defined components from the requirements stage are examined and specifies how each component is implemented.
3. Implementation: Basically consist of implement the detailed design into the required environment.
4. Verification: In this stage the implementation meets the requirements. Some tests are developed to find errors in the implementation.
5. Maintenance: Addresses the errors that may occur and enhancement requests after the software releases.

The Waterfall model fits with the project that is to be developed, but other options are also valued. Some of the stages of this model cannot be considered in this project, such as maintenance stage. The tool that is to be developed is not expected to be released, due to it no enhancement requests are contemplated. Instead of it, the tool will be tested and evolved based in the error that may affect the results after the testing stage.

Although the waterfall model has some weaknesses, it emphasizes in important stages of project development. Hence, even if this model is not applied, all of these stages must be considered.

Advantages:

- The requirements are clear before the development starts.
- Every stage is completed in a specific period of time.
- It is an easy model to understand and to implement.
- The resources to implement this model are minimal.
- The knowledge while using this model is widely known.

Disadvantages:

- Is not flexible.
- Due to it is one of the oldest models, is idealized and does not match the reality well.
- Does not match with the iterative reality of development.
- Difficult and expensive to make changes. Once in on stage, correcting errors from early stages costs rework and time.
- The problems with one phase are never solved completely in that phase, due to it some problems occur when the first stage is already signed and sometimes not possible to change.

Despite the pros and cons of this model, having a general idea about the stages to follow helps while developing a software tool, even if the model is not followed in each of the stages of the model.

2.5.2 V-Model

The V-Model is one of the most representative model for traditional software testing management. It is considered as an extension of the waterfall model and it was first proposed by Pau Rook[39]. The purpose of this variant is to improve efficiency and effectiveness of the developed software. This model reflects the relation between each phase of the development lifecycle and its associated phase of testing. Instead of moving down, such as the Waterfall model does, the process steps are bent upwards after the coding phase, to perform the V shape.

The V-Model deploys a well-structured method, in which each phase can be implemented by the documentation of the previous stage. Testing starts at the beginning of the project, well before coding. This method saves a high amount of time in front of the waterfall model, before the development is started, a test system plan is created. This test plan focuses on meeting the functionality specified in the requirements gathering.

[39] Rook, P.: Controlling software projects (1986).

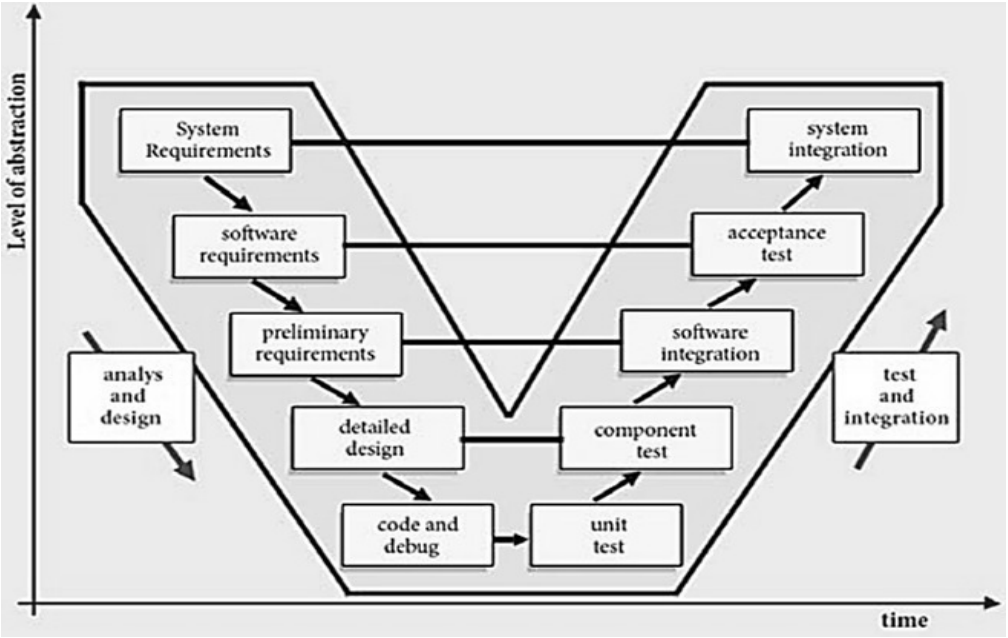

Figure 20: V-Model structure

The stages are very similar to the Waterfall Model, but in this case, software testing is too important to leave to the end of the project. Hence, V-Model incorporates testing into the entire software development life cycle. An integration plan is created in order to test the parts of the software system ability to work together. The model illustrates how each subsequent phase should verify and validate the work done in the previous phase, and how work done during development is used to guide the each of the testing phases.

The process shown in the figure 20, proceeds down and then up, from left to right depicting the basic stages of development and testing activities. This model highlights the existence of different levels of testing and represents how each relates to a different development phase. In the right of the diagram the tests plans developed earlier are put to use. The interconnectivity makes easier to detect important errors and omissions to prevent the process to be harm as soon as possible. As in the Waterfall Model, this model has some weaknesses and some strengths:

Advantages:

- It is simply to use and easy to understand.
- Each phase has specific deliverables to the next
- Every stage is completed in a specific period of time.
- Higher chance to success than in the Waterfall model because of the earlier development of the test plans.
- Requirements can be changed in every phase.
- The tester role will be involved in the requirement phase itself.
- Works well in small group projects where the requirements are quickly understood.
- The knowledge while using this model is widely known.

Disadvantages:

- Too rigid for some cases.
- Due to rigidity, any change becomes difficult and expensive, all the documentation for each step must be updated

- The model does not provides any path for found problems during the testing part.
- It is not recommended for short term projects, due to reviews take time at each stage.

Due to this thesis is not a product project, those both models are taken into account despite none of them can be totally followed. Hence, in this project the stages are followed as far as possible without developing some of these stages. For example, since the beginning, the tests to evaluate the tool are planned for every stage following the V-Model. Due to the size of this project, every small error in the requirements is simply to fix. But considering how the tests are implemented from the very beginning can make this project take into account some important details. Hence, the V-Model is the chosen to follow in this project as far as possible. Specifically, the followed stages in this project are system requirements, software requirements, detailed design, code and debugging, unit test, component test and software integration.

3. Methodology

In this chapter the methodology of this project is explained. All the technologies previously explained are taken into account to build the final tool. It is detailed in the first points which technologies are used, how each of those contributes and the reasons why it was chosen. An explanation of how the tool works is also detailed, emphasizing in specific requirements of each stage. To apply some of the chosen technologies the motorbike needs some structural changes, these changes and the reasons for them are also detailed in this chapter.

Due to the selected V-Model, in each part of this chapter the necessary tests are taken into account. Hence, each part of the tool is developed to fit in the entire set and able to be tested or changed in any point during the data acquisition. The developed tests to analyze the results of implementing this tool are illustrated in the final part of this chapter.

A general overview of how the tool works is also explained in this chapter, detailing how the user may use the tool and which specifications may change in every case.

3.1 Selected technology

In this section are specified the selected technologies as a summary to make clear which ones are used in this project.

V-Model:

The different stages of this model contribute to have clear the order of the different steps in this project. The reason why this model is chosen and not the traditional Waterfall model is the tests planning. Although most of the stages are the same, the tests planning at the beginning of the process was able to save time and took into account the tests while developing the tool and making the tests more profitable.

ArUco markers and library:

These markers represent a simple and reliable method to track the pose of the rider. With only adding this marker to the rider suit, any user can estimate the pose with no deep knowledge about image processing. The only limitation in this case is the focus width of the camera, but it is the same for the other considered technologies. Both Haar cascade classifier and Tensorflow are possible technologies to accomplish the purpose of this project, but due to the high amount of data needed to train the algorithms and the time to do it, are discarded.

OpenCV library:

These library provides the needed functions to perform the camera calibration without the need of a payment program. Furthermore this library is constituted of different functions that can help to develop the tool and to set different options up for the user and may help performing the tests. Some of the functions provided by this library allow the tool to: calibrate the camera, undistort the images, detect the markers, track the markers, get the results in a proper data file, and many other functions.

Python:

This programming language is chosen to use the OpenCV library. Another coding languages could be choses, but most of the information in the OpenCV sources are published in python and is a simply programming language to work with without being specialized in code programming. There is also the option to translate the coding language into some others which can use OpenCV, but this part depends on the user and is not contemplated in this project.

3.2 System requirements

Some of the needs from the initial project have changed. Due to the new selected technologies, some parts must be implemented in the motorbike, and some others need to be built to perform the tests.

Cameras:

In this project two different cameras are used, the Kodad Pixpro SP360 4k and the Sony Fdr-X3000. One is placed in the front of the motorbike and the other in the rear part. Both cameras are able to record in 4K resolution. In this project, the higher resolution of the camera the higher chances to detect the markers, due to the resolution in each frame. Both cameras are fisheye cameras, hence in the calibration stage it is needed a special treatment to eliminate the type of distortion that the fisheye cameras apply to the images. However, this type of distortion does not represent a disadvantage for this purpose. Although the image is cropped to avoid the distortion, due to the wide focus on this type of cameras, using a fisheye camera represents an advantage for the marker recognition.

Figure 21: The two fisheye camera models: Kodak Pixpro (left) and Sony Fdr-X3000 (right)

Both cameras are designed to withstand harsh conditions, both have protective cases in case of impact and stabilizers to record the recordings properly.

Fixings:

The used cameras need a specific fixings. Due to the required position in the motorbike, two anchor points are enabled in the rear and in the front part of the motorbike. In the front part, due to the reduced spaces, the front windshield was cut to place the fixing of the camera, and a small structure was built to fix the front camera.

Figure 22: Front camera fixing

In the rear part there is more space, hence a more robust system was built. An aluminum profile was used to build a small structure to hold the camera. This small structure needs to be able to move along the Z axis to regulate the distance to the rider in case that its height does not allow the camera to record it properly. For the same reason the camera needs to rotate to focus the important parts of the motorcyclist.

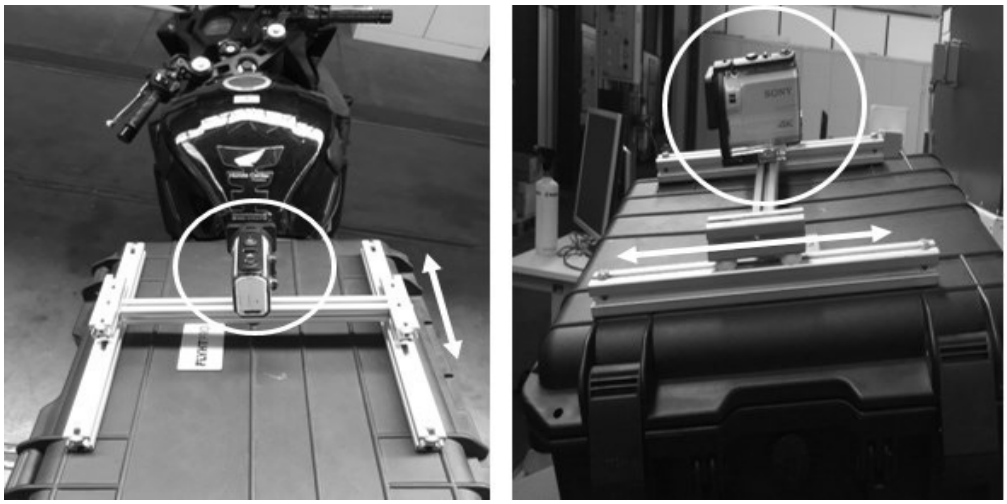

Figure 23: Rear camera fixing

Printed markers:

The markers were printed and plasticized to be more resistant to bending against the strong winds to which they will face on the road. A high resistant Velcro will be glued on the marker and in the rider jacket to fix the markers in the areas of interest for the pose estimation. The markers are 10x10 cm squares and belong to the ArUco dictionary 6x6_250. The size of the markers may help the cameras to detect it, depending on how far is one from each other.

Test system:

A main test was developed in this project. It consist of evaluating the real position of the marker and the estimated position calculated by the tool. Hence, a small structure was built to evaluate the results. It consist of a small aluminum frames to hold and keep stable the marker while recording them. It also needs to allow the movements in all the directions and all the rotations that the tool is able to detect.

Figure 24: Test system

As shown in the figure, the structure allows both translation and rotation movements. Hence all the motions and limits of the software detection can be evaluated.

Computer:

To develop the code and to apply them once developed a computer is indeed needed. In this project, it is tried not to need a great computation power to make any user able to use it. In this case an i5 2,3GHz processor is used with 4 GB RAM memory available. Hence it was demonstrated that a normal processor can run this tool.

Chessboard:

In order calibrate the cameras, to extract the intrinsic and extrinsic parameters, a chessboard pattern is used in this project. This patter need to be printed and glued into a plane surface to perform calibration. Once glued into a plane surface, either the camera or the patter need to be moved into different positions in front of the other to make the camera understand the distortion from all the possible perspectives. In this a project a 49,5 x 77 cm chessboard patter is used. Every white or black square of the pattern has a size of 5,5 x 5,5 cm, hence the pattern is composed 9 x 14 squares. Notice that

either in the horizontal and vertical, the patter has a white square in the corner. This cannot be detected with any function, due to it, the detected pattern is 8 x 13 squares.

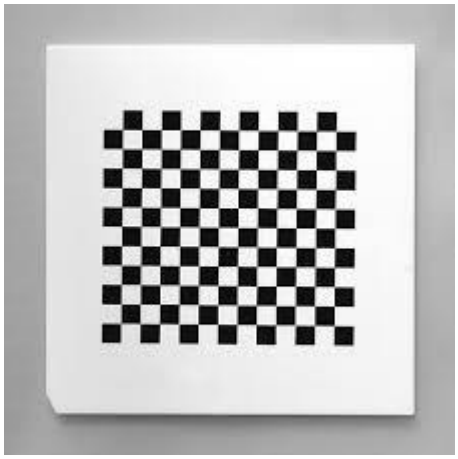

Figure 25: Example of a printed chessboard pattern

3.3 Software requirements

As explained before, the tool needs several software requirements to work properly. Due to the selected technologies, in this project, the following software are used in on or more stages.

- Python: This software is explained in previous chapter, it is used to write the code, to execute the algorithm and to treat the data extracted from the tool.
- OpenCV: These libraries are used to build the tool. The functions from these libraries are applied in several stages in the code of this project, either to analyze the data or to plot the results.

3.4 Detailed design

In this chapter the different parts of the code are explained. The code is divided according to the different processes that shape the tool: calibration of the camera, removal of the existing distortion in the recorded video, detecting the markers in the already undistorted video, tracking the markers throughout the video and extracting results from the recorded data. Not all the functions in the code are explained, but it is given a general idea of how the code works and which are the parameters that the tool needs in every case.

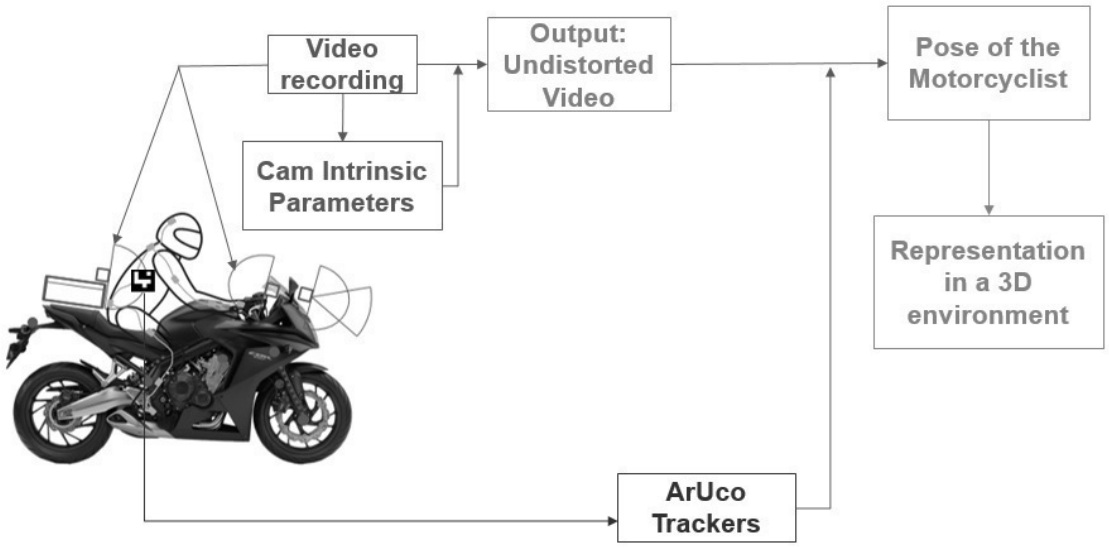

Figure 26: Proposed architecture of the tool

3.4.1 Calibration

As explained in several occasions, the camera calibration is needed to extract the camera parameters, which are used later to track the markers. The process of the camera calibration is described in the following diagram:

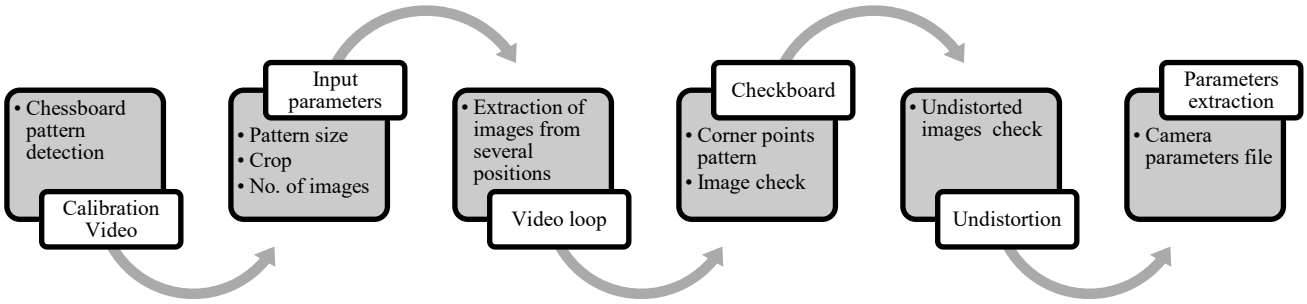

Figure 27: Calibration process

The first step to do, is recording a video of the chessboard pattern while moving it to a variety of positions in the field of view of the camera. Once is recorded, it has to be given to the program the specifications of the pattern to recognize it, such as: size of the squares, number of rows and columns, crop on the undistorted image, image resolution and number of images taken to perform the calibration. The user also has to write the location of the chessboard file and the location where the output will be written. The calibration process is based in the relation between the camera sight and the reality, which is represented by the parameters that are introduced in the code as inputs.

Table 1: Calibration parameters

	Units	Description
Size of the square	cm	Width and height of each square
Number of squares	-	Number of squares in the horizontal and the vertical. Must be an integer
Image resolution	Pixels	Dimensions of the image. It is not mandatory to give this parameter, but makes the code faster.
No. of images for the Calibration	-	Number of images that the user needs to select to perform the calibration. It is recommended 20 as minimum. Must be an integer
Crop	0 to 1	Value of the crop to apply to the undistorted image. A value of 0 will crop out all the black pixels and a value of 1 will leave all the pixels

Then the code takes the important information of the input video, such as: width, height, frames per second, number of frames, and some other properties. Usually these parameters are known, so the code works faster if the parameters are introduced manually. However, if the code takes all the needed information directly from the input video there cannot appear mistakes.

The program enters in a loop where the video is played and the user is able to see the video. In this point the user needs to select, with a simple command, the images required for the calibration. The user must select images of several different positions, distances and orientations of the pattern. All the selected images are stored in a specified folder to see with which images the calibration was performed for future uses.

Once the total number of images are selected, the code extracts the parameters from the selected images. The code initializes a loop through each image, turns it into a greyscale to detect better the borders, and detects the corners between all the squares. If the number of corner points matches with the correct number of corners, the true points and the measured points are stored. If the number of corners does not match, the image is discarded directly.

In this loop, the non-discarded images are displayed with a specific pattern relating all the found points, called checkboard. The user needs to press a simple command to continue with the next images if all the points are properly found. This step is useful for the user to check if the program has detected wrong some points, if this happens the process can be stopped. Also, if the user considers the rest of images enough for the calibration, can skip the wrong picture and continue with the process.

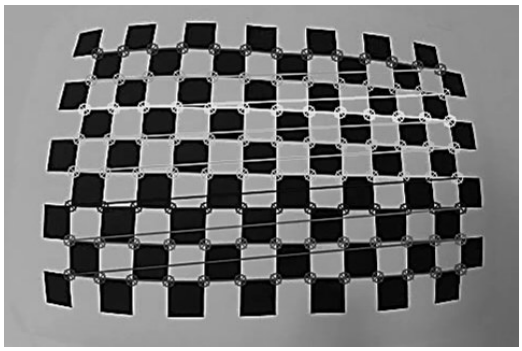

Figure 28: Checkboard pattern

When all the images are checked, the code takes all the object points and the image points and calculates the distortion coefficients and the camera matrix. For different types of cameras, such as fisheye cameras, some function change, hence the user needs to know which camera is using and which functions are required. Once the code has these parameters, the images are undistorted applying these parameters to show the user if the extracted parameters work well or does not work properly.

Some of these steps are not mandatory, for example showing to the user the checkboard or the undistorted images makes the program slower. However, this part of the tool is critical to have the correct parameters to estimate the pose of the markers, hence spending more time making sure that the obtained parameters are correct is a good method to avoid future errors.

Showing the images while the calibration process is executing comes from the V-model. While developing the requirements and functionalities of the calibration process, some tests were thought to be able to check if the results are correct or not, and concluded including this checking tests during the calibration.

Finally, the camera parameters are stored in a file for the next stages. Having these parameters in a file allows to call the data whenever it is required, and if it is the case, the parameters can be modified or the format can changed.

In this project there are two different cameras, and the tests are made with different configurations of the cameras, hence as many camera calibrations as different cameras and configurations have to be done changing the parameters according to each configuration.

The camera matrices and the distortion coefficients for each configuration of each camera are the followings:

- Pixpro in 4k mode:

$$K_{\text{pix4k}} = \begin{bmatrix} 730{,}53 & 0 & 1892{,}12 \\ 0 & 724{,}28 & 995{,}45 \\ 0 & 0 & 1 \end{bmatrix} \tag{3-1}$$

$$Dist.coeff_{\text{Pix4K}} = [-0{,}1183 \;\; 0{,}0096 \;\; 0{,}00171 \;\; -0{,}00022 \;\; -0{,}00029] \tag{3-2}$$

- Pixpro in MP4 mode:

$$K_{\text{Pix4K}} = \begin{bmatrix} 431{,}19 & 0 & 967{,}64 \\ 0 & 432{,}21 & 544{,}20 \\ 0 & 0 & 1 \end{bmatrix} \tag{3-3}$$

$$Dist.coeff_{\text{PixMP4}} = [-0{,}1083 \;\; 0{,}0074 \;\; 0{,}00038 \;\; 0{,}00038 \;\; -0{,}000075] \tag{3-4}$$

- Sony in 4k mode:

$$K_{\text{Sony4k}} = \begin{bmatrix} 1121{,}36 & 0 & 980{,}029 \\ 0 & 1123{,}53 & 521{,}38 \\ 0 & 0 & 1 \end{bmatrix} \tag{3-5}$$

$$Dist.coeff_{\text{Sony4K}} = [-0{,}2290 \;\; 0{,}1838 \;\; 0{,}00038 \;\; -0{,}00107 \;\; -0{,}14313] \tag{3-6}$$

- Sony in MP4 mode:

$$K_{\text{SonyMP4}} = \begin{bmatrix} 1122{,}16 & 0 & 982{,}059 \\ 0 & 1123{,}85 & 521{,}09 \\ 0 & 0 & 1 \end{bmatrix} \tag{3-7}$$

$$Dist.coeff_{\text{SonyMP4}} = [-0{,}2275 \;\; 0{,}1673 \;\; 0{,}00058 \;\; -0{,}00087 \;\; -0{,}1297] \tag{3-8}$$

3.4.2 Undistortion

After extracting the camera parameters, the tool is able to undistort any video recorded with the same camera. Nowadays the same camera can take a video in many different formats. Hence the user needs to make sure to use in every record, the same camera parameters, otherwise both the undistortion and the tracking programs will not obtain the correct results.

This stage of the tool takes the input videos and undistort them frame by frame. This process is shown in the following figure.

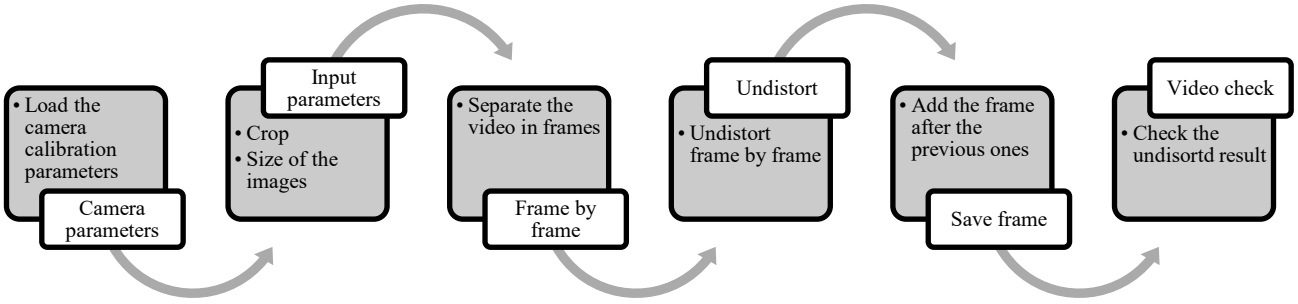

Figure 29: Undistortion process scheme

First of all, the code reads the input parameters. The user must write the location of the video to undistort, the location of the camera calibration parameters, where to write the undistorted file and the crop. The crop, as explained before, represents which percentage of the empty parts of the image are cropped. This parameter influences to a large degree on the final result and it is recommended to try more than one time to check which value works better for each case. The different results may vary due to the distance from where the video was recorded.

At the same time, the code reads the different characteristics of the video, for example the size, the resolution, frames per second, the total number of frames, and some other properties. This parameters can be written by the user, and the program will work faster. But as in the calibration stage, is defined as automatically read by the program. The goal of this project is to build an easy tool for the user,

therefore this parameters are automatically taken from the program to avoid mistakes from the user. It does not matter which type of video the user uses, the program will take the parameters properly.

Coming up next, the program starts with a loop where frame by frame, the video is undistorted. With the camera calibration parameters, the code is able to undistort each frame of the video. To be able to do it, the program need to split the video frame by frame. What the loop does is to separate the video in frames, then it takes the first one, undistorts it, saves it to the undistorted video and repeats the same process with the following frames until the video has no more frames.

After the first frames are undistorted, the program shows the undistorted video to the user. The goal of playing the video consist of letting the user know if the final result is properly undistorted. With this step, is enough to stop the process before it ends if the results are not correct, saving time for the user and for the tool.

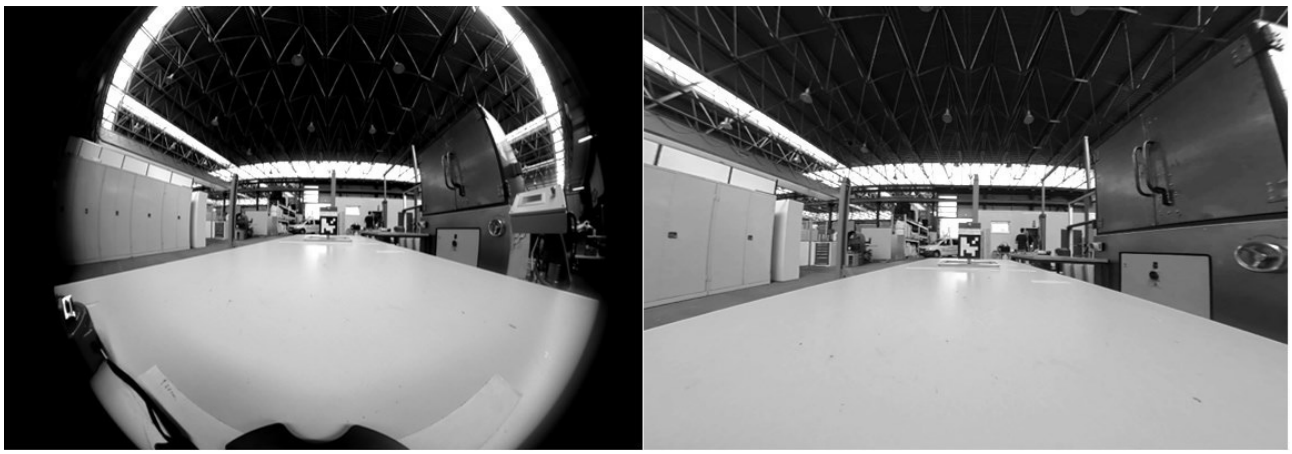

Figure 30: Example of a distorted image (left) and undistorted (right)

In the figure 30 can be appreciated the differences in the same image once undistorted. The most visual effect on the image, thanks to the crop function in the code, is the lack of a circular frame. The curved lines in the distorted image, once undistorted, appear as straight lines. Before detecting the markers, all the videos must be undistorted, otherwise the straight lines of the markers may appear as curved lines, degenerating in some errors while detecting the markers.

3.4.3 Detection and tracking

After all the previous stages are successfully completed, the video file is ready for the detection and tracking stages. This section is a critical point for this project. The main objective of this work is to detect markers, because of this reason this stage has to be meticulously done.

The ArUco library contains functions to estimate the position of the markers and build the ArUco marker detection algorithm. This algorithm may vary in each case, due to the different external conditions, the axes that the user considers or the number of markers to be detected. Some variations are based on applying different filters before the detection, to change the format of the image to make the markers easier to detect by the detection functions. In this project some filters are considered to make

the detection process more efficient. However, the filters are not always necessaries, hence this tool will allow the user to apply or not the applied filters depending on the conditions of the recording.

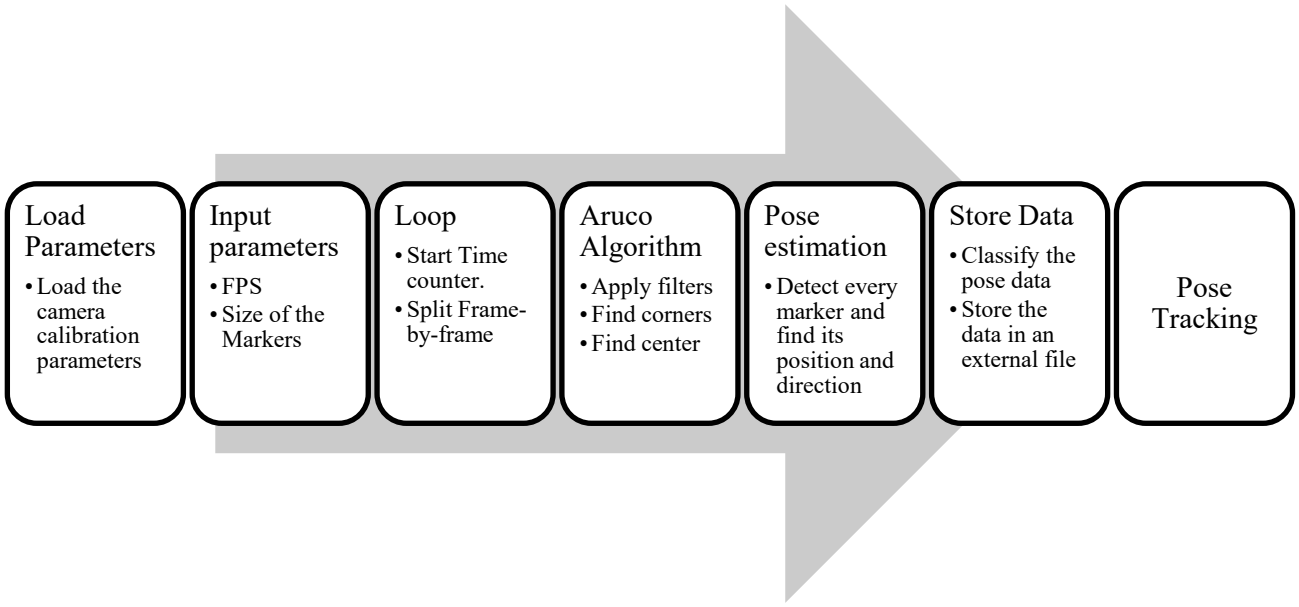

Figure 31: Pose estimation algorithm scheme

The figure 31 shows generally the performance of the de pose tracking algorithm. However, a more detailed diagram is attached in the annexes chapter.

The first step the algorithm does is collecting the data from all the needed parameters. Some of the inputs are: the undistorted video, the size of the video, the camera parameters, the size of the markers, the used dictionaries of the markers, the total duration of the video, and some other inputs. The camera parameters and the undistorted video are uploaded directly from the saved file. As in previous stages, the rest of the data can be either written manually or extracted by the program from the video file. As said before, it is considered not to introducing manually the parameters an advantage to not make mistakes. The program also need the size of the marker to detect only those markers with the required specifications. According to the type of marker, it is used a specific dictionary. If the dictionary or the size of the marker are not properly introduced in the code, the tool cannot detect the markers properly, it generates mistakes and probably the rest of the data cannot be considered correct.

After introducing all the required parameters, the program loop applies the ArUco detection algorithm the each single frame of the video. The detection algorithm needs to be applied to a single frame, hence the video is separated into single frames and the loop is applied form the first frame until the last one in the same order than the video.

To insure the correct order of the results, when the loop starts, a timer starts to measure the time of the video, to attach to each frame the specific time in the video. With this, the tool insure the coordination between the results and the specific time in the reality.

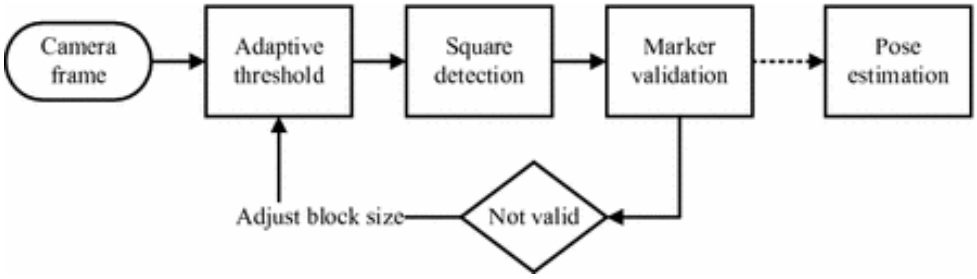

Figure 32: ArUco detection algorithm

As shown in the previous figure, the process of detecting the markers starts applying an adaptive threshold filter. After converting the each frame into greyscale, this filter converts each pixel into black or white depending on its pixel value. In the simple threshold, if the pixel value is greater than one specific threshold value the filter converts it into black, else it converts it into white. In this simplification the user defines the threshold value, but in the adaptive threshold the same filter scans which regions on the image are lighter and which ones are darker, assigning a specific threshold value for each of them. Applying the adaptive threshold, this tool reduces the need of having the same light in all the regions of the squared marker, and also makes the algorithm understands which are the requirements in different light conditions from the environment.

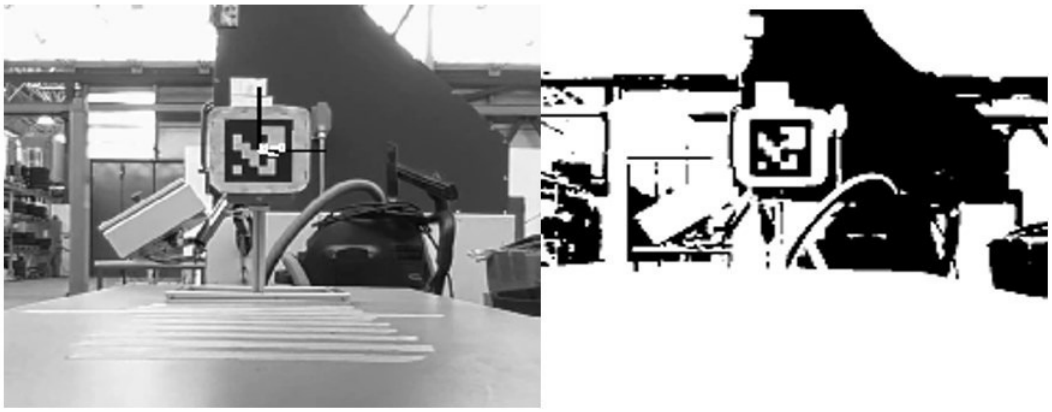

Figure 33: Normal image (left) and image with threshold filter (right)

There are several filters to be applied before the detection. According to the needs of the tool, the user can add another filters in the same point.

After the filters are applied, the program detects all the corners of the markers that appear in the current frame. This step represents a critical point in the program, if a single corner is not well detected, it may cause errors while extrapolating the results into real coordinates. Hence, in this part the corners results are validated. According to the size of the markers and with the camera calibration parameters, the algorithm understands if the corners are related with the real distance or not. If the program detects a wrong distance between the different corners, considers the detected object not a marker. This process helps the algorithm discarding the similar objects and the detection errors. This

process also increases the accuracy of the threshold filter, giving the filter feedback about the results with a specific configuration.

After the marker is validated as an existing marker, the tool continues with the pose estimation. The already detected positions of the corners are represented in the image coordinate system. The positions are given with the number of the pixel where the corner is located. The real coordinates must be extracted from this pixel coordinates. With the camera calibration parameters, the algorithm can convert the pixel coordinates into camera coordinates. The software extracts the translation and the rotation vectors of the markers, the coordinate position and the orientation in each frame.

Some functions are predefined to extract the position of the camera relative from the marker. For this project, the position of the marker relative to the camera is the wanted value. This process consist of transforming the coordinates into the real world coordinates, in this project these coordinates are the same as the motorbike coordinate system.

After changing the coordinate system, the next stage is to extract the positions and orientations in an external file to analyse the results and to draw conclusions. The data is in different formats, because each functions requires a specific format to work. Hence the tool processes the results to save only the specific relevant data, and to get the results in an easy to evaluate format. More than one marker can be tracked at the same time, hence a well organised output file is necessary to extract conclusions from the final file.

Once the data is well collected, the algorithm displays the video with some collected features, such as the borders of the detected markers, the centres, the axes and finally the position and rotations of each marker. This allows the user to have an immediate idea about if the program is working well or not. And allows the user to extract information without working with a data file, only watching the screen and comparing with the real position in the video.

Figure 34: Detected marker with coordinates

3.4.4 Test and data analysis

Once the tool is developed it is time to proof its accuracy and efficiency. It is known the general accuracy of the ArUco markers, there are several studies evaluating the error while tracking this type of markers in different conditions, such as those in the project of Rodrigo, S.[40].

Some of the studied tests in those sources are proved under different light conditions, from a range of different distances or simply increasing the number of markers.

This project is not focused on studying the accuracy of these markers, but a series of tests are developed to contrast the founded information and to evaluate the accuracy of the developed tool.

Due to the specific conditions in this project, different tests are developed to determine the ranges where the tool can work and its accuracy. Also the computation efficiency is evaluated for each code of the tool. In this case, it only depends on the computational power of the used device, but an approach with the used computer is given.

Every camera is tested with two different resolutions. In both cases, for the Kodak and the Sony, a first test with the highest resolution is done. In both cases, the highest resolution is 4k, and the second one in a lower resolution.

Table 2: Camera configurations

	Video size	Angle of view	FPS
Pixpro 4k	4K 3840x2160 16:9	235º	30p
Pixpro MP4	MP4 2048x2048	-	30p
Sony 4k	4k 3840x2160	"wide" ($f = 17\ mm$)	60p
Sony MP4	MP4 1920x1080	"Medium" ($f = 23\ mm$)	60p

From now on, the names of the different tests, are named as Pixpro 4K, Sony 4k for the highest resolutions and Pixpro mp4 and Sony mp4 for the lowers ones.

3.4.4.1 Translation test

The application of the markers in this project is focused in evaluating the pose of the rider, due to the position of the cameras in the motorbike the distances are limited in a specific range. This range limits the positions in the image where the markers can be detected. In this test, the positions for each axis is evaluated.

[40] Gonçalves, L. M.; Vision, W. o.: WVC 2017 (2017).

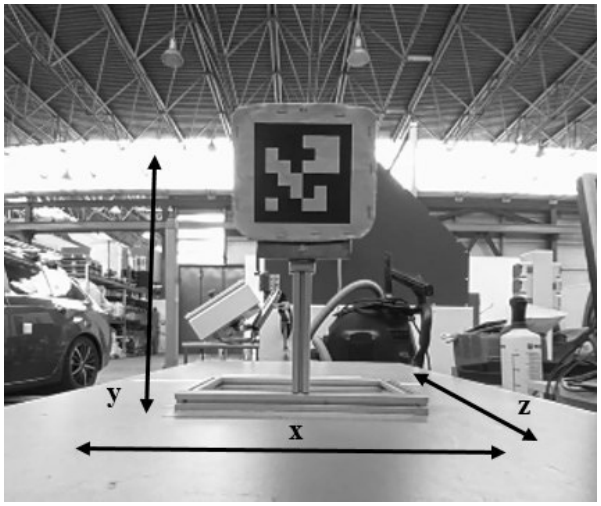

Figure 35: Translation tests movements

The longitudinal distance from the cameras to the rider vary between 40 - 80 cm. This distance is represented in the motorbike coordinate system in the Z longitudinal axis, due to it, the tests for these axes do not consider distances outside this range. For the other tests, the same distances ranges tested. Both X and Y axis are tested from a distance of 60 cm, this distance represents the middle point of the range where the cameras might work.

Table 3: Translation tests

	Range	Description
Axis X	-20 cm – 20 cm	Intervals of 5 cm
Axis Y	-20 cm – 20 cm	Intervals of 5 cm
Axis Z	40 cm – 80 cm	Intervals of 5 cm.

The measurement tool to set the distances before placing the markers have a specific accuracy. Understanding which accuracy requirements are needed for each case can help the project avoiding some high precision tests that may not be necessary. In this case the measuring tool has a precision of $\pm 0{,}5$ mm in the measurements.

3.4.4.2 Rotation test

The rotational position of the marker helps to understand how the marker is rotating from its last position. In this case there is a physical barrier. When the marker reaches a certain angle, the camera is not able to see, and thus to detect or to track. The rotational tests in this project are focused in finding the accuracy in a specific range of angles, but also searches for the maximum angle that can be detected with each configuration and for every camera. In the specified range, the marker is moved by 5º every time in a total range of 45º.

Table 4: Rotation tests

	Range	Description
Roll (x Axis)	180º- 135º	Intervals of 5 º
Pitch (y axis)	0º - 45º	Intervals of 5 º
Yaw (z axis)	0º - 45º	Intervals of 5 º

For these tests a digital protractor is used to measure the angles more dynamically. This tool has an accuracy of $\pm 0{,}5$ º.

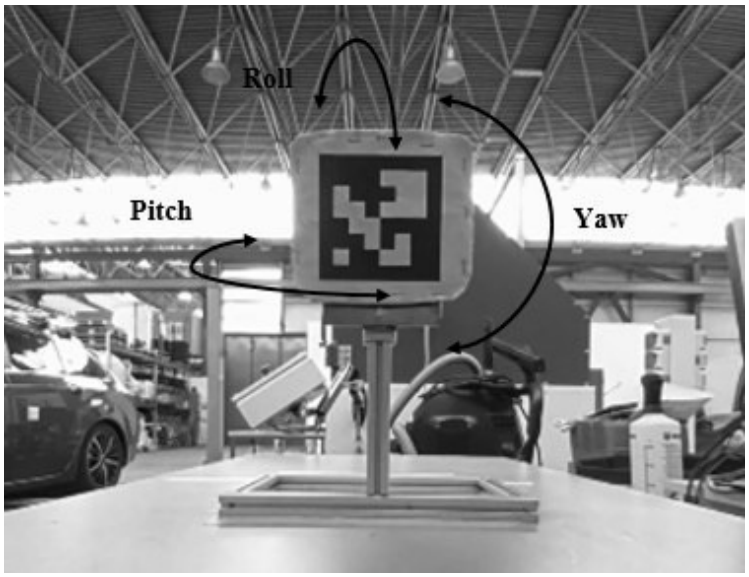

Figure 36: Rotation test movements

3.4.4.3 Test on the motorbike

In this part, evaluating the accuracy of the detection is difficult due to variable positions of the rider. However this test tries to determine if all the modifications on the motorbike in addition with the configurations of the camera are able to capture all the moves of the rider. In another hand, evaluating if the selected option to stick the markers in the jacket of the rider are a good solution.

Evaluating the accuracy of the tool with this test is highly difficult due to the lack of a reference point. Thus, the markers placed in the rider jacket must have a specific relation between them. With all the markers related between them, the results can be contrasted comparing them with the set positions between the markers.

Finally once the tool computes the results of the ride, the coordinates of each marker are represented in a graphic. Comparing the real recordings with the graphics, evaluating if it shows properly the position of the markers.

Figure 37: Rider with ArUco markers

3.4.4.4 Processing time estimation

The computation time estimation depends on the used computer, however an estimation for each of the parts of the tool is given to have a reference point about which computational power is required, and also how much time requires the tool to process all the recordings.

In order to develop this test, some options of the tool will be dispensed with, such as reproducing the video to see how the markers are retreated. However, being the fastest option, it will be tested with the highest resolution video, thus the information about the worst case within the fastest code is founded.

4. Results

In this chapter, the results from the tests are evaluated in order to extract conclusions. The results seem favorable at first sight, but to be able to evaluate them correctly they will be represented in different ways. First, the values extracted by the camera x_c will be plotted against the measured valuesx_{meas}, which are expected to be detected. Both for the translation results and the rotation results are represented in the same axis to compare them easily. In some cases, the difference between cannot be appreciated, hence the region with more differences between them is plotted to evaluate which one is better for each test. To have a better vision about the differences between the expected and measured results, these will be represented in a graph showing the differences in each point from which it has been taken. A polynomial approximation for the differences, in front of the distance or rotation, is also given. This approach allows to have a vision of the trend of these values in each point, and also avoid some points that are not representatives for the results. For both rotational and translation cases, the maximal difference is given in addition to the average of all the differences, to compare between every camera configuration. In the rotation cases, at some point, the camera cannot detect the marker. This is the limit point for the camera to detect the markers in the rotational axis, and is given as well in the next tables.

- Translation tests:

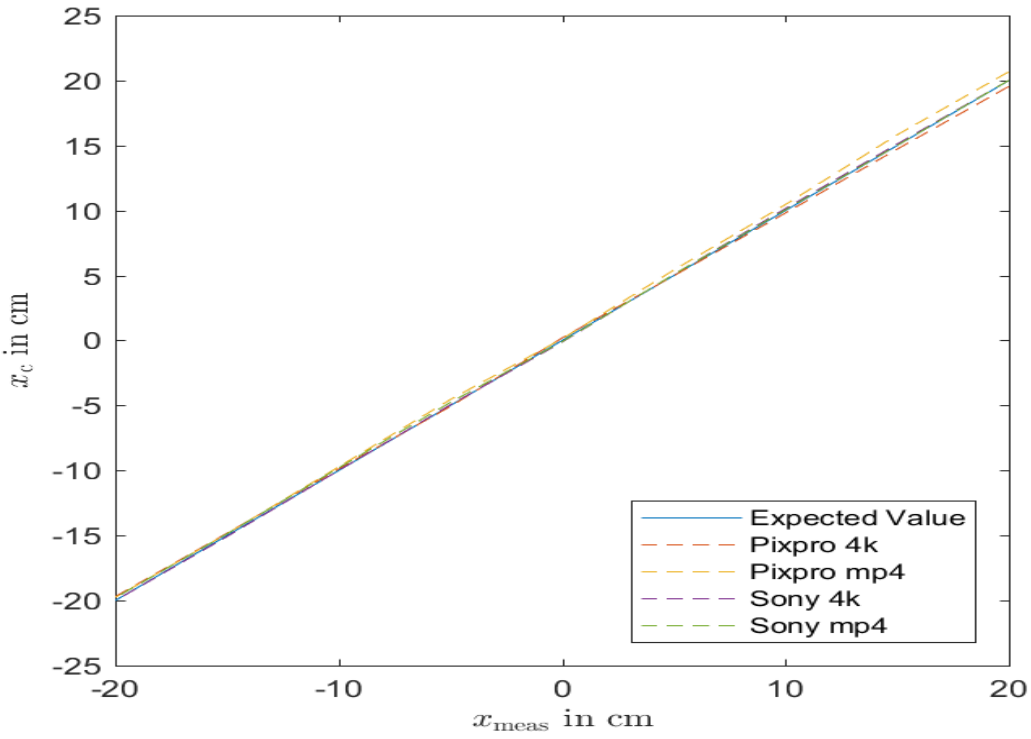

Figure 38: Results in the X axis

In the figure 38 it is shown the comparison between the expected values in the X axis for all the camera configurations. All the values for all the configurations are very close to the real ones. The

differences increase when the distance from the centre also does. Hence, the region with the highest values is evaluated again.

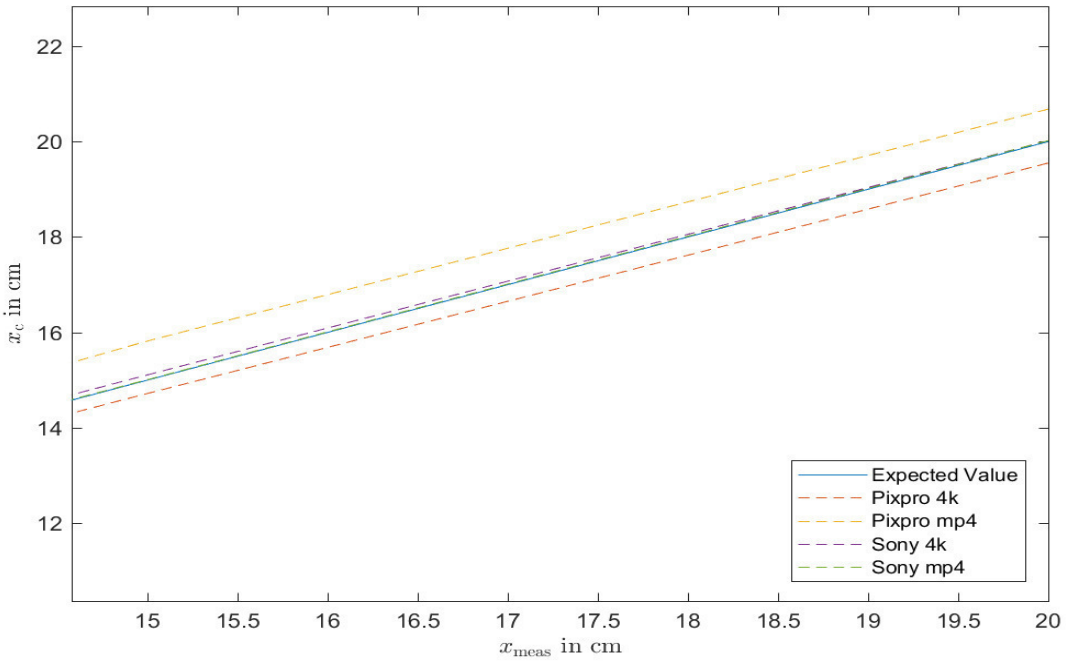

Figure 39: Maximal difference region for X axis results

The figure 39 shows that for the Pixpro camera, the results in higher resolutions are better, while for the Sony camera both results are similarly accurate. However the differences are not too much to consider all the configurations proper to estimate the pose of the rider in this axis.

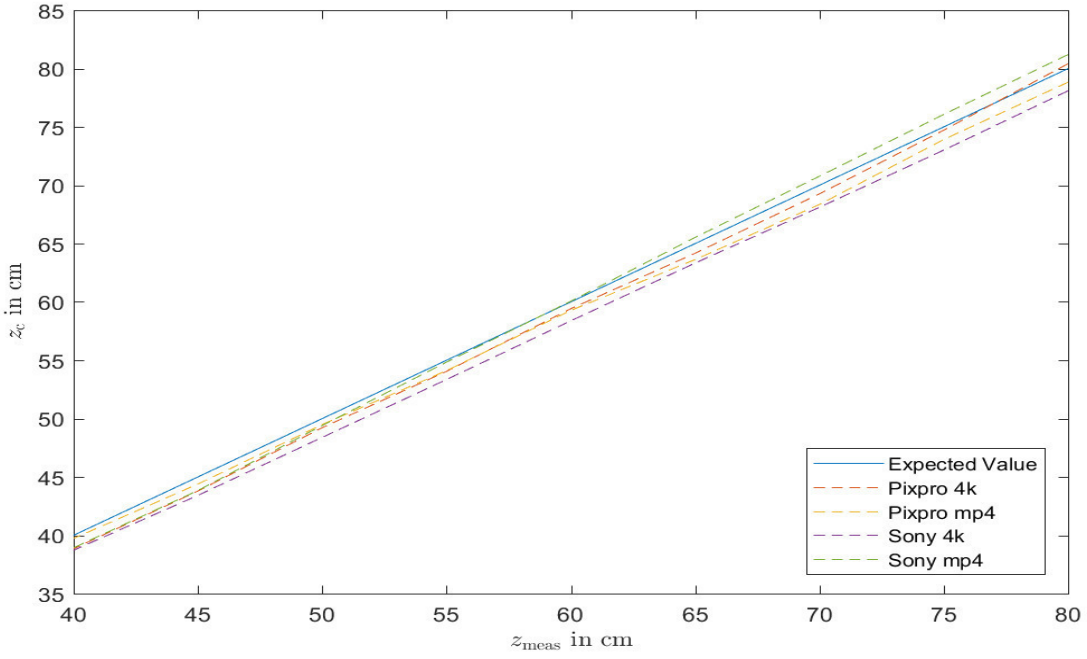

Figure 40: Results in the Z axis

In the figure 40 can be seen that the camera which captures better the position along the Z axis is the Pixpro. However, it happens the same as in the previous picture, for every camera configuration the results are accurate enough. As in the previous case, z_{meas} belongs to the expected results, the measured positions where the marker is placed, and z_c refers to the values extracted with the tool. The differences between them are positively accurate.

The results from the Y axis is not show in this chapter due to its similarity with the X axis. However, the graph from the Y axis is attached in the annexes chapter. Furthermore, the most representatives values from the data obtain in the Y axis test is shown numerical in a table with the rest of results.

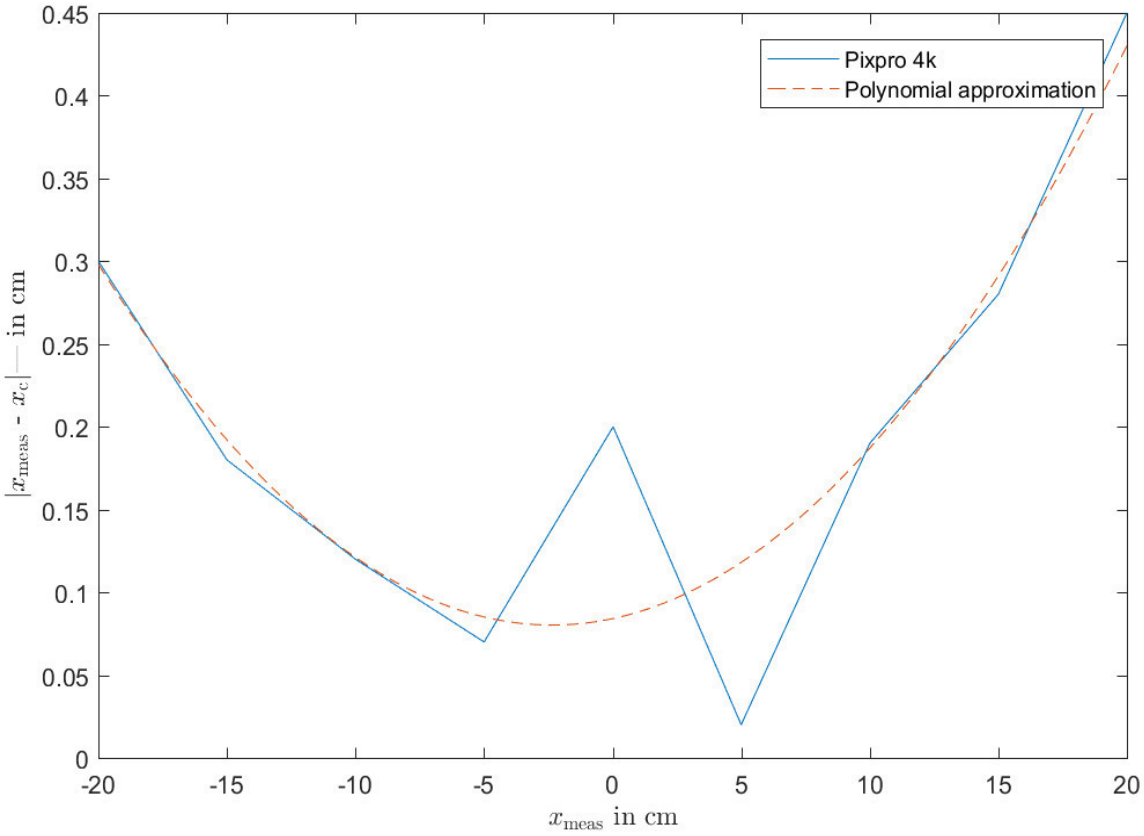

Figure 41: Differences between x_{meas} and x_c for the Pixpro 4k configuration

The figure 41 shows more clearly how the measurement from x_c loses accuracy as it moves away from the centre. A polynomial approximation is represented to understand better how the error evolves between the expected measurement and the obtained one. The expression of this function is given in the translation results table in the following pages. This function represents the trend of the difference between x_c and x_{meas}, smoothing the trend caused by some points whose value is outside the expected error.

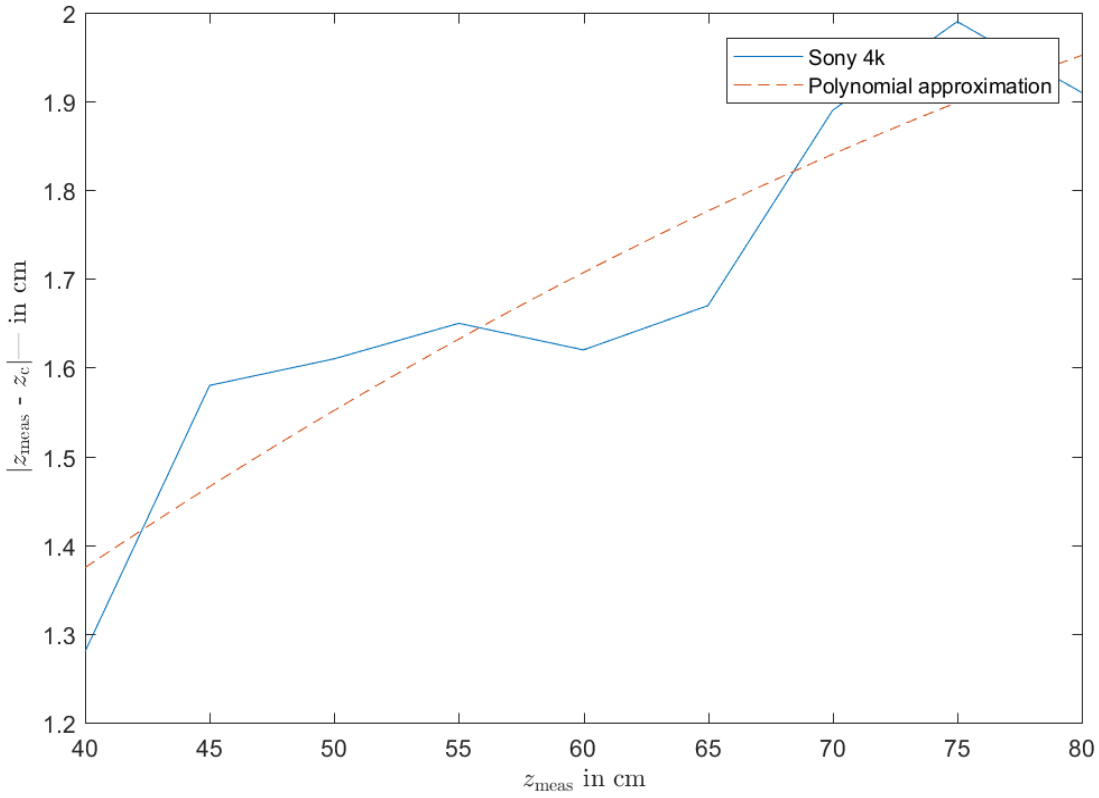

Figure 42: Differences between z_{meas} and z_c in the Sony 4k configuration

In the previous figure, it is shown the trend of the difference between the real position of the marker z_{meas} and the detected from the camera z_c. In the Z axis is shown clearly the tendency to increase the error as the marker moves far away from the camera. However in the worst cases the maximal difference is not greater than 2 cm.

Table 5: Translation results

	Axes	Max. *Diff.* (cm) $\lvert x_{meas} - x_c \rvert$	Average *Diff.* (cm)	Polynomial Approximation of *Diff* $diff(x_c) = a_0 x_c^2 + a_1 x_c + a_2$
Pixpro 4k	X	0,45	0,18	$diff(x_c) = 0{,}0007x_c^2 + 0{,}0033x_c + 0{,}0841$
	Y	0,42	0,23	$diff(x_c) = 0{,}0003x_c^2 - 0{,}0092x_c + 0{,}1865$
	Z	0,95	0,77	$diff(x_c) = -0{,}0203x_c + 1{,}9734$
Pixpro MP4	X	0,82	0,40	$diff(x_c) = 0{,}0004x_c^2 + 0{,}0137x_c + 0{,}3383$
	Y	0,72	0,31	$diff(x_c) = 0{,}0005x_c^2 + 0{,}0194x_c + 0{,}2295$
	Z	1,17	0,92	$diff(x_c) = -0{,}0008x_c^2 + 0{,}1202x_c - 3{,}3316$
Sony 4k	X	0,15	0,07	$diff(x_c) = -0.0001x_c^2 + 0{,}0276x_c + 0{,}4469$
	Y	0,22	0,14	$diff(x_c) = x_c^2 - 0{,}0004x_c + 0{,}1463$
	Z	1,99	1,69	$diff(x_c) = -0{,}0001x_c^2 + 0{,}0276x_c + 0{,}4469$

Sony MP4	X	0,24	0,11	$diff(x_c) = -0{,}0053x_c + 0{,}1191$
	Y	0,25	0,09	$diff(x_c) = 0{,}0002x_c^2 - 0.0022x_c + 0{,}0531$
	Z	1,16	0,73	$diff(x_c) = 0{,}0024x_c^2 - 0{,}2843x_c + 8{,}76$

In the previous table some values from the tests are given. The maximal values of error are greater for lower resolutions, it means using a better resolution a better accuracy is obtained. This fact is also proved by the average difference. Between the two cameras, the Pixpro is more accurate along the Z axis, while the Sony is more accurate in X and Y axes. However, both cameras need to be used at the same time, hence having a similar accuracy will make the future usages more trustful. In terms of translational precision, none of the precisions of the different configurations is imprecise enough to be discarded.

- Rotational tests:

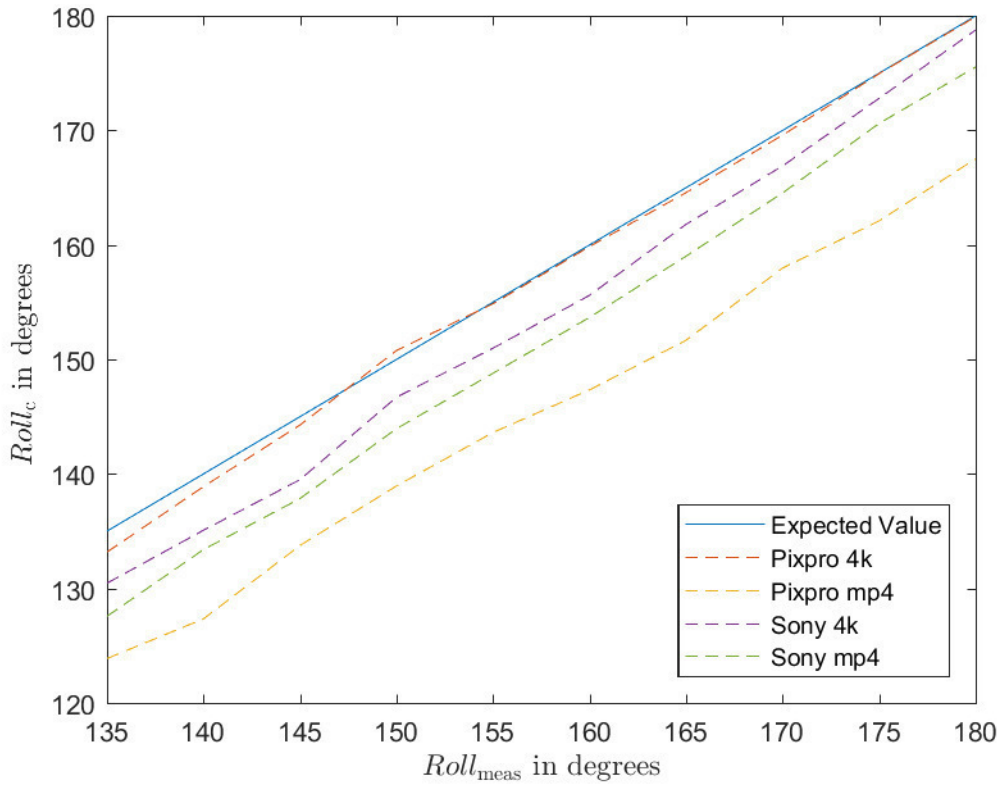

Figure 43: $Roll$, results

As in the translation graphics, the results for the $Roll_c$, $Pitch_c$ and Yaw_c from the camera detection, are represented in front of the expected values $Roll_{meas}$, $Pitch_{meas}$ and Yaw_{meas} . In the figure the $Roll$ results from all the configurations is shown. In this case, the lower resolutions are noticeably more imprecise than the higher resolutions. As in the translations results, the $Pitch$ graphs is similar to the $Roll$ one due to the similar movement. Hence the graphic is attached only in the annexes.

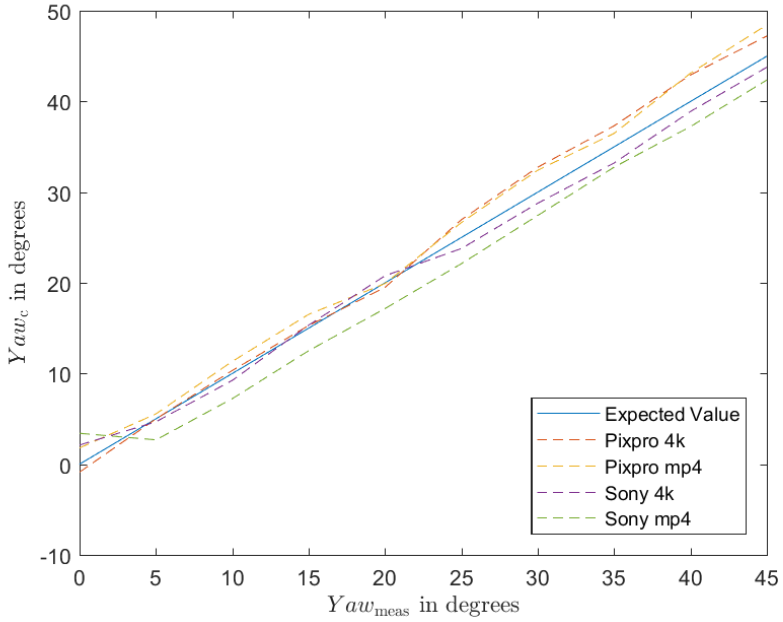

Figure 44: *Yaw* Results

The Yaw reveal for this rotation, a similarity between the higher resolution between the higher and lower resolutions. However, for all the configurations, the Yaw rotation values are precise enough to satisfy the purpose of this poject.

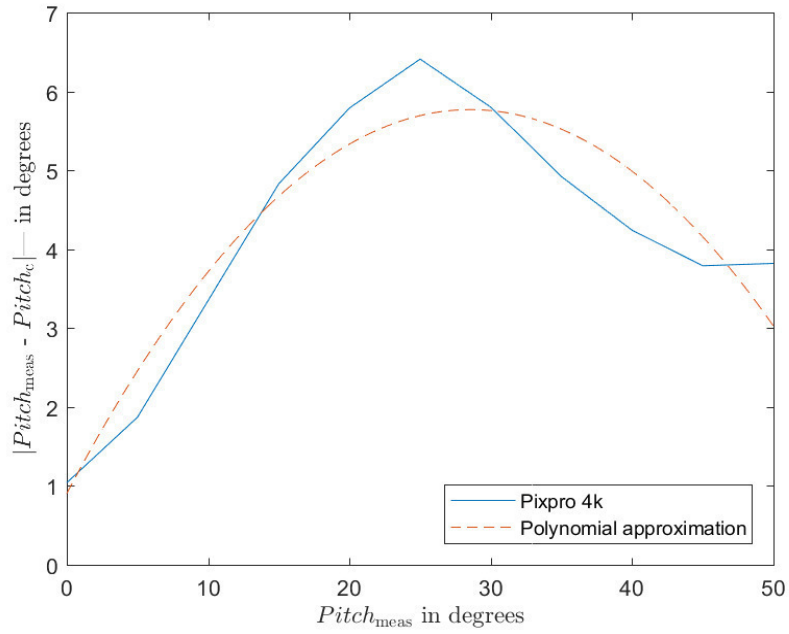

Figure 45: *Pitch* difference in the Pixpro 4k configuration

As in the previous graphics, the difference $|Pitch_{meas} - Pitch_c|$ is approximated in a polynomial function to understand better the trend of the error. In this case, the evolution of the difference do not give much information about tendency of the measures. However, the greatest difference in the worst case is around 6 degrees, a value that is acceptable for the purpose of this tool.

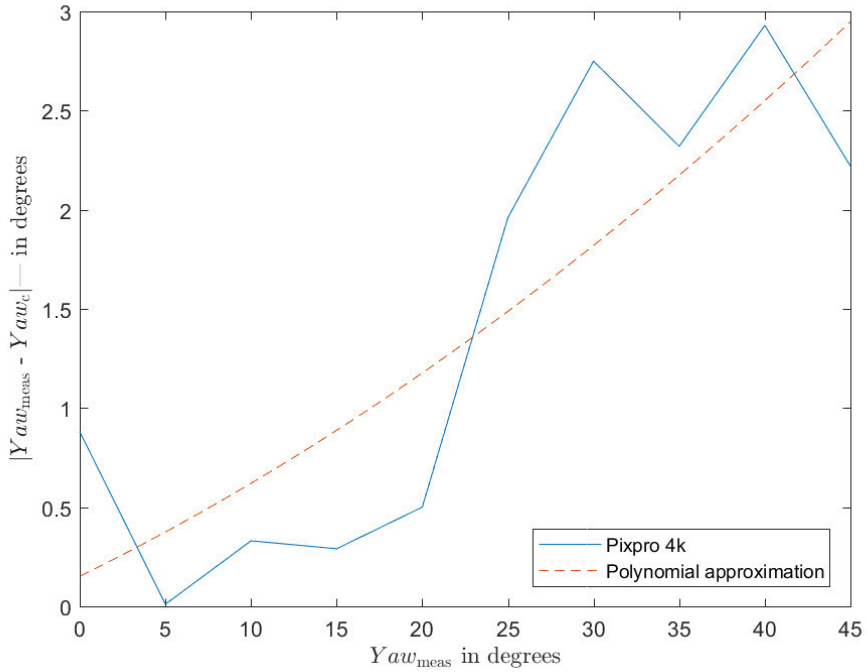

Figure 46: *Yaw* differences in the Pixpro 4k configuration

In the Yaw difference representation is shown the tendency of the measurement system to increase the error as the marker moves away from the origin point. However the difference is never higher than 3º, hence in this axis and with this configuration, the tool is precise enough.

Table 6: Rotational results

	Axes	Max. *Diff.* (cm) $\lvert x_{meas} - x_c \rvert$	Average *Diff.* (º)	Polynomial Approximation of *Diff* $diff(x_c) = a_0x_c^2 + a_1x_c + a_2$	Limit value (º)
Pixpro 4k	X (Roll)	6,19	1,77	$diff(x_c) = 0{,}0012x_c^2 - 0{,}4070x_c + 34{,}8453$	100,44
	Y (Pitch)	2,02	0,99	$diff(x_c) = -0{,}0060x_c^2 + 0{,}3408x_c + 0{,}9032$	71,57
	Z (Yaw)	2,93	1,42	$diff(x_c) = 0{,}0004x_c^2 + 0{,}0426x_c + 0{,}1506$	-
Pixpro MP4	X (Roll)	13,33	12,08	$diff(x_c) = 0{,}0213x_c + 7{,}9433$	112,43
	Y (Pitch)	2,31	1,43	$diff(x_c) = -0{,}0072x_c^2 + 0{,}4336x_c + 0{,}4434$	68,54
	Z (Yaw)	3,45	1,74	$diff(x_c) = 0{,}0026x_c^2 - 0{,}0713x_c + 1{,}4841$	-
Sony 4k	X (Roll)	4,92	3,64	$diff(x_c) = -0{,}0019x_c^2 + 0{,}54x_c - 32{,}6758$	110,94
	Y (Pitch)	1,53	1,03	$diff(x_c) = -0{,}0033x_c^2 + 0{,}2197x_c + 2{,}291$	70,39
	Z (Yaw)	2,11	1,08	$diff(x_c) = 0{,}0012x_c^2 - 0{,}0454x_c + 1{,}2803$	-
Sony MP4	X (Roll)	7,44	8,08	$diff(x_c) = -0{,}0008x_c^2 + 0{,}1989x_c - 4{,}625$	106,38
	Y (Pitch)	3,57	1,39	$diff(x_c) = -0{,}0034x_c^2 - 0.1818x_c + 2{,}055$	74,1
	Z (Yaw)	3,4	2,64	$diff(x_c) = 0{,}0005x_c^2 - 0{,}0295x_c + 3{,}0091$	-

In the tables, the most significant values about the rotational tests are given. The values for the higher resolutions is more precise than for the lowers. Due to de capability of creating more clear images, the markers can be detected better for these configurations. For the highest resolutions, the values of the average difference and the maximum difference are precise enough to evaluate properly the rotation of the markers. Furthermore, the limit angle that each configuration can detect are also favorable to the highest resolutions.

4.1 Test on the motorbike

Once the markers are tested on the motorbike, the position can be extracted taking into account the accuracy calculated in the previous chapter. Due to the size of the markers, in some occasions the camera is not able to detect them. Also because the markers are not in a totally plane surface, they fold, being undetectable for the tool.

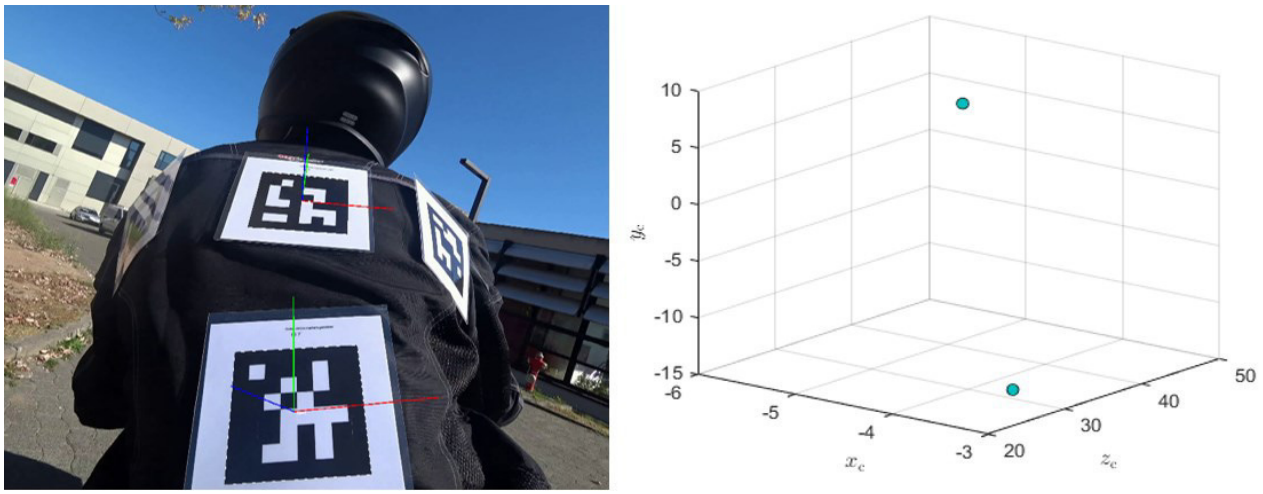

Figure 47: Pose estimation on the motorbike, rear view

In this case, it is shown the positions of the markers, in this situation, there are two markers that cannot be detected due to the position and also because of the size of the marker.

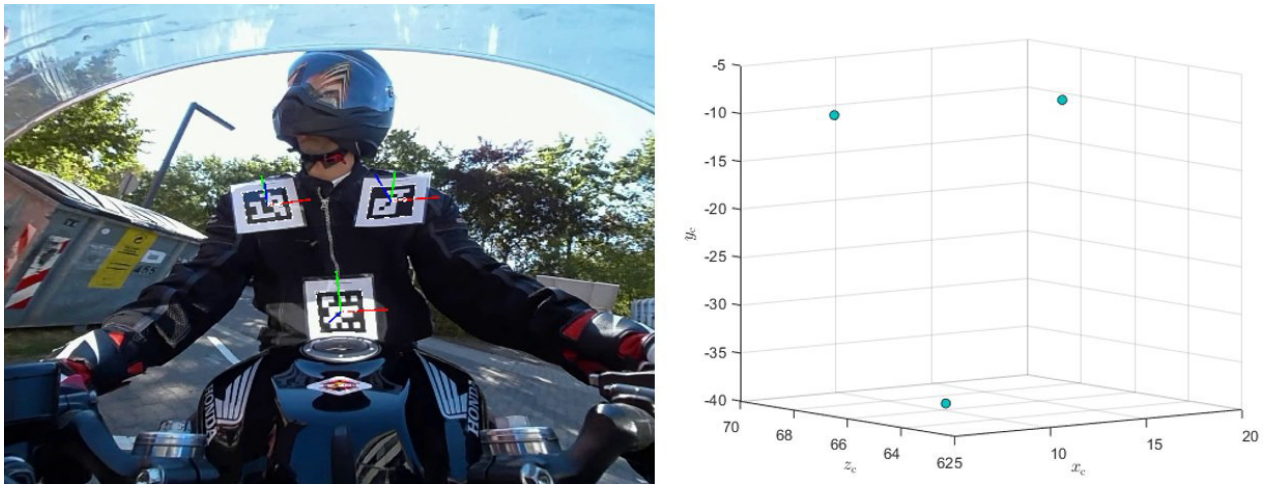

Figure 48: Pose estimation on the motorbike, front view

In the figure 48 is shown how the three markers at the same time are perfectly detected and their positions.

4.2 Processing time estimation

The different stages of the tool present the processing times shown in the table 7. Not all the stages use the same video, thus the processing time per every minute in the input video is calculated. However in all the stages the videos are recorded in the Pixpro 4k configuration.

Table 7: Processing time of the different stages

	Time of the input video (min:sec)	Processing time (min)	Processing time (min) / input min	Processing time (sec) / frame
MP4 to .avi	2:02	17:03	8:38	0,288
Calibration	1:00	2:12	2:12	0,073
Undistortion	3:29	28:05	8:40	0,289
Detection and tracking	2:02	26:19	13:34	0,452
Total	-	73:39	-	-

The total amount of time required by the tool to do the whole process is reasonable. Using a faster computer this processing times could be easily decreased.

5. Conclusions

Before the start of this thesis, the project of the Technische Universität Darmstadt was not able to track the position of the rider. The tracking suit was not giving accurate results and the other options were not evaluated yet. Besides the several data provided by the different sensors installed in the motorbike, the pose and orientation of the rider was unknown. Hence this project was not able to relate the results from the state of the motorbike, the surrounding views and the pose of the rider.

The main goal of this thesis was then to provide a technology to track the pose of the rider. A review of the latest technologies about image processing was done, providing a base of knowledge for future possible variation in the tool.

Due to the fact, that the installed cameras are equipped with fisheye lenses, their images distortions had to be removed to get straight lines, necessary for computer vision algorithms. Therefore an intrinsic calibration was needed, resulting in an easy-to-use camera calibration script. This script is based on the OpenCV intrinsic camera calibration. Only for each camera, one recording of images showing a calibration pattern is required to gain precise intrinsic camera parameters.

Using the new calculated intrinsic parameters, the distortions of the original fisheye cameras were removed, undistorting each frame of the recordings.

Based on the ArUco markers technology, the position and location of the fiducial markers were verified through measurements made by hand. This markers where attached to the body of the rider, to prove the viability of this tool for the main purpose of the project.

Overall, the final outcome of this thesis is a promising base for further works, using the installed fisheye camera system with the ArUco based tracking system. This work can be seen as a starting point on top of which most diverse simple computer vision tracking algorithms, with a remarkable accuracy that for sure can be improved with development.

6. Future works

After analyzing the results and taking into account every conclusion, this chapter introduces the future works that this project allows to do and some aspects to develop.

About the technologies, it would be interesting to be able to contrast the results with the results that could be extracted from other technologies such as convolution pose estimation network. Due to the high demands in terms of hardware, it has not been possible to prove in this project, but it would be a method to take into account, due to no marker being needed.

In this project the size of the marker is not changed in the different tests. With the considered ranges of detection, a reasonable size should be properly detected in any case, and the goal of this project is not to test the accuracy in every single variant in the markers objects. However, decreasing the size of the detected objects can help the markers adapt better to the rider's jacket or even to better adapt to all the possible heights of the riders and tracking more points from the position at the same time. These upgrades are not difficult to test, changing the marker size in the code allows the algorithm to detect this size of markers. Hence, printing markers with different sizes and changing the size specifications in the code may allow to test it easily.

In this project, has been noticed that with de current size of 10 cm, the markers are too big to track several points. Hence for further usages, printing the markers in a smaller size would help to estimate the pose of more points. Furthermore, with this size, the marker does not adapt properly to the size of the body, hence it folds in some parts and cannot be properly detected in some cases.

Another aspect that could be improved is the light influence on the markers. This project has not taken into account the effect of light on the markers, mostly due to the time it takes to test all possible light scenarios. Hence several tests under different light conditions must be developed, either inside with artificial light or outside with natural light. In another hand, the tool has a filter that adapts the light of each region depending on the contrast with its neighbors. This means that the algorithm should not have any problem to detect the markers under different light conditions, but to make sure this theory, it must be tested.

There are several filters to be applied. Some of them have been applied in this project, but surely there are some others that could help increasing the precision of the tool.

The set of tests that have been carried out to determine the accuracy of the algorithm can be completed for a more extensive range of positions and angles. With this, the tool would have information about the precisions in each point, and could estimate in a more efficient way the error that it commits in each case.

About the code, a loop closure update could increase the precision of the detections. This type of algorithm compare the current position with already detected poses, verifying the current positions with the coordinates acquired in the previous ones.

The different parts of the tool are separated in different scripts, this takes time for the user to set the parameters for each part of the tool. Each part of the tool does not take a long time to compute the

data, however combining the undistortion and the tracking part could make the process faster. The process of analyzing the data also needs some time for the user. A pre-treatment of the output coordinates data may help analyzing the results.

Another point to consider for future works is the data fusion of the markers from both sides of the rider. To give an accurate position of the considered part, placing the marker in the surface of the body of the rider does not give an accurate position of the selected part. A good optimization for this project could be fusing the data from both sides to find the intermediate point, which would represent better the position of the part.

The process of analyzing the data needs some time for the user. A pre-treatment of the output coordinates data may help analyzing the results.

Appendixes

Pose estimation diagram:

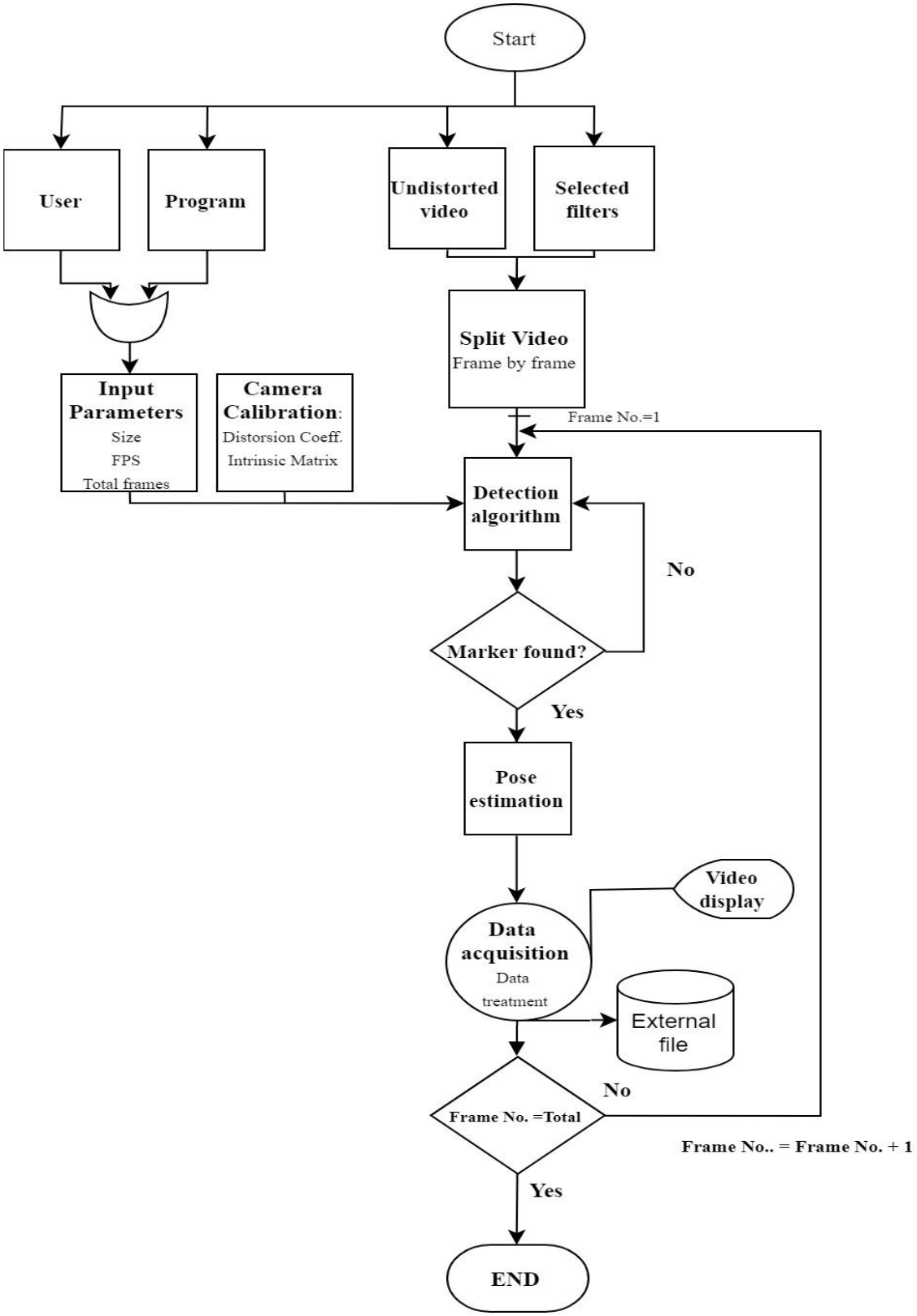

Figure 49: Pose estimation diagram

Translation tests:

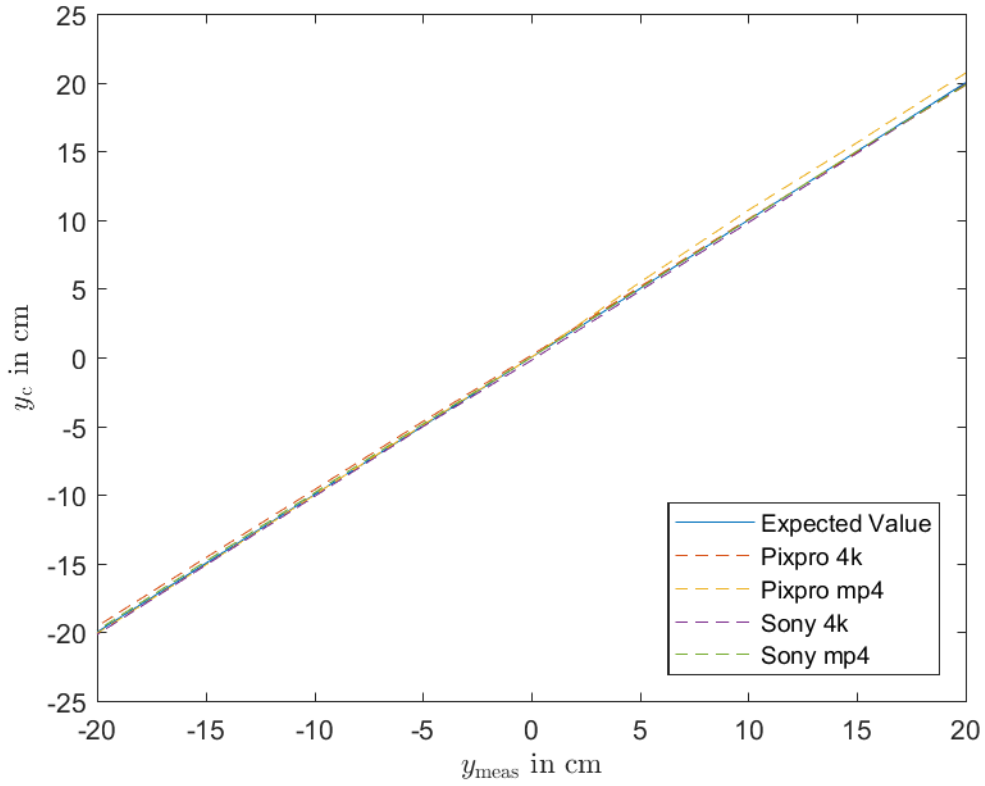

Figure 50: Y axis results

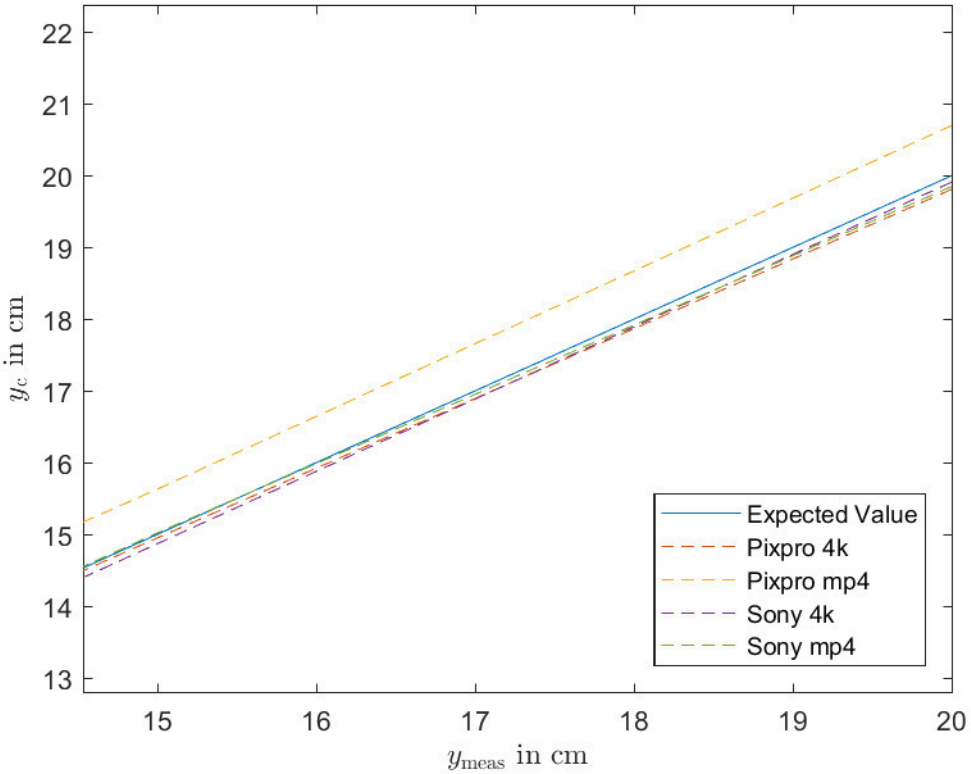

Figure 51: Y axis results in detail

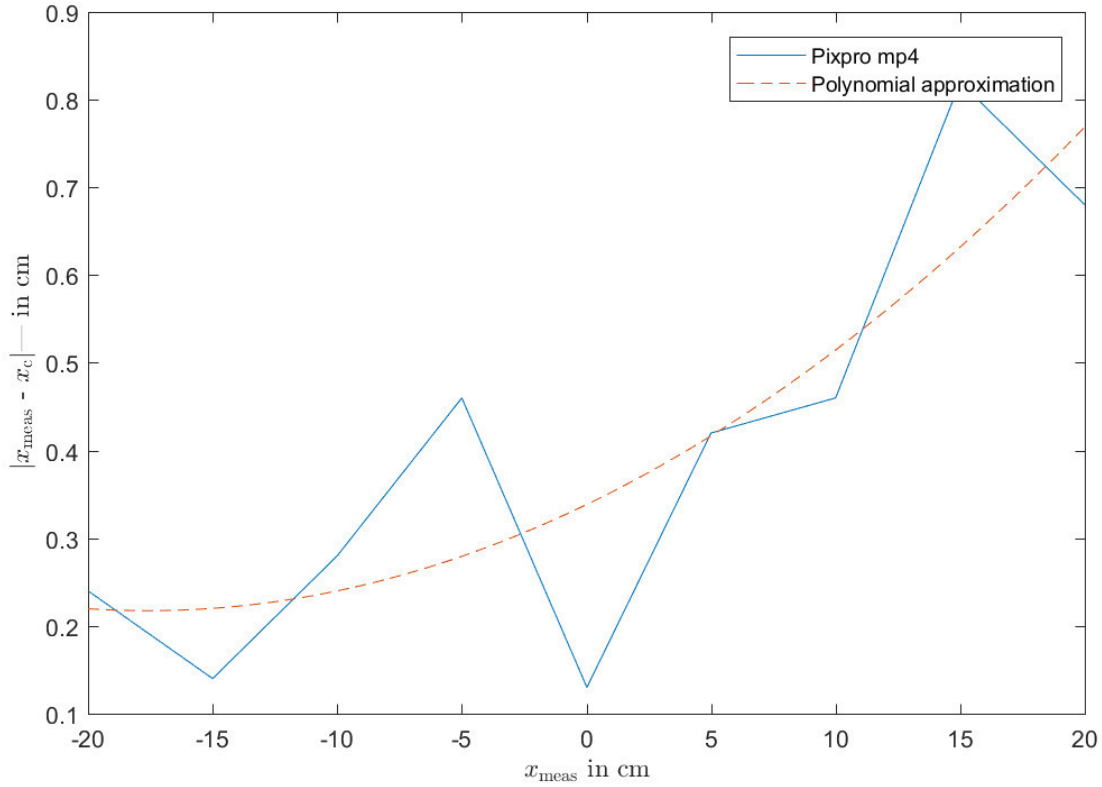

Figure 52: Pixpro mp4 X axis differences results

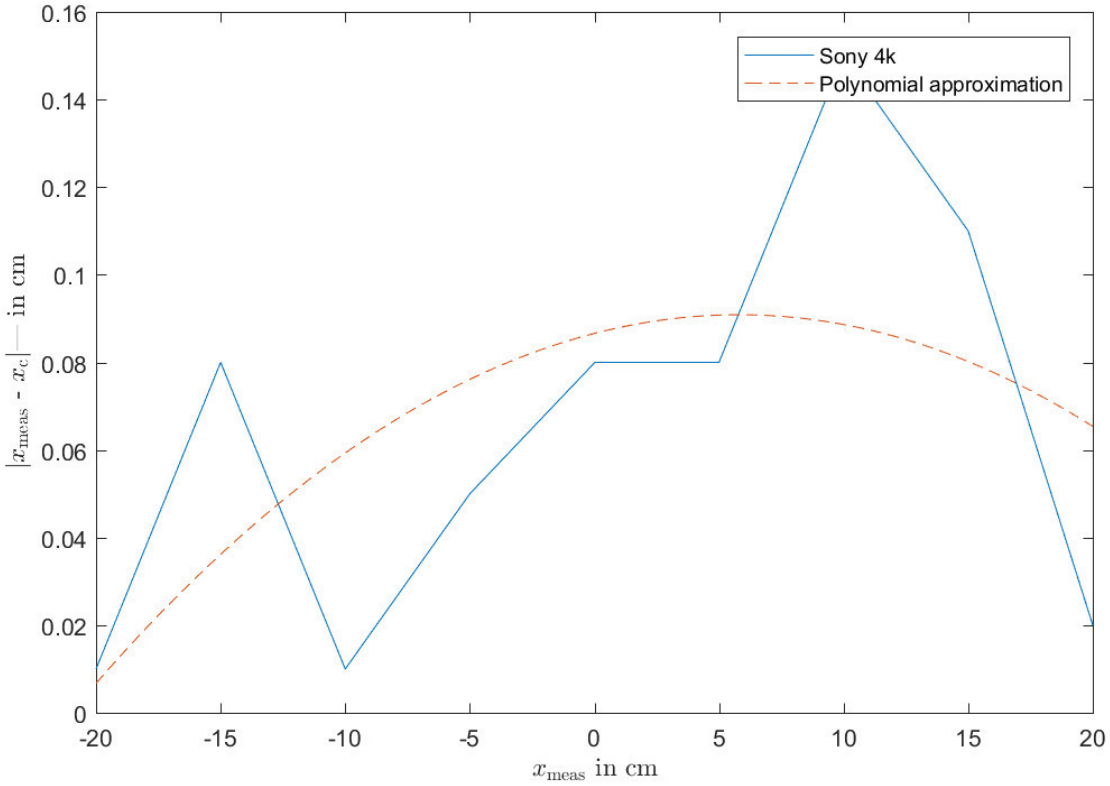

Figure 53: Sony 4k X axis differences results

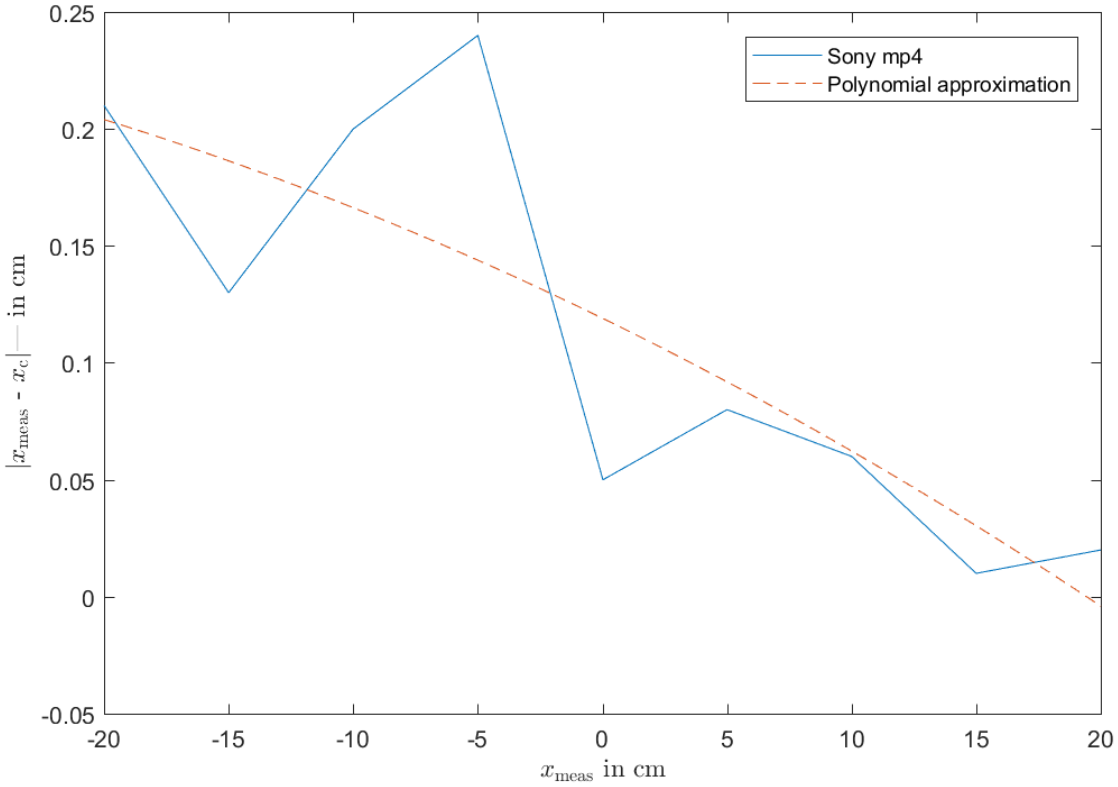

Figure 54: Sony mp4 X axis differences results

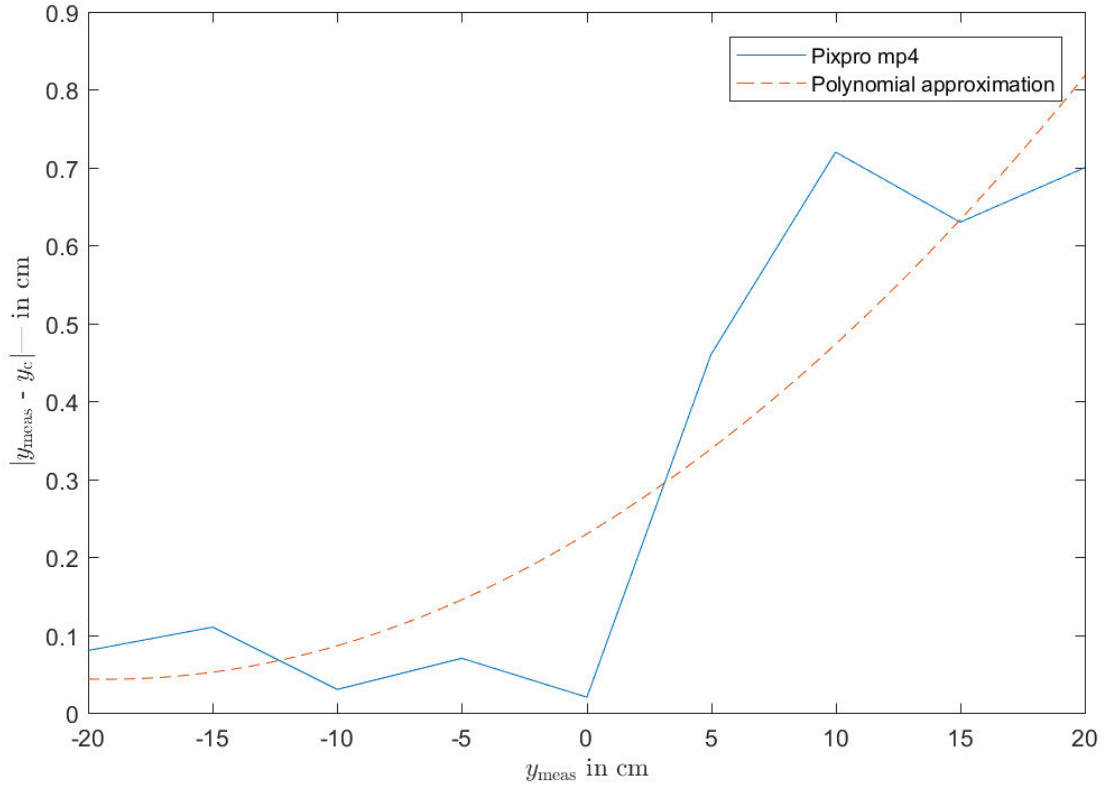

Figure 55: Pixpro mp4 Y axis differences results

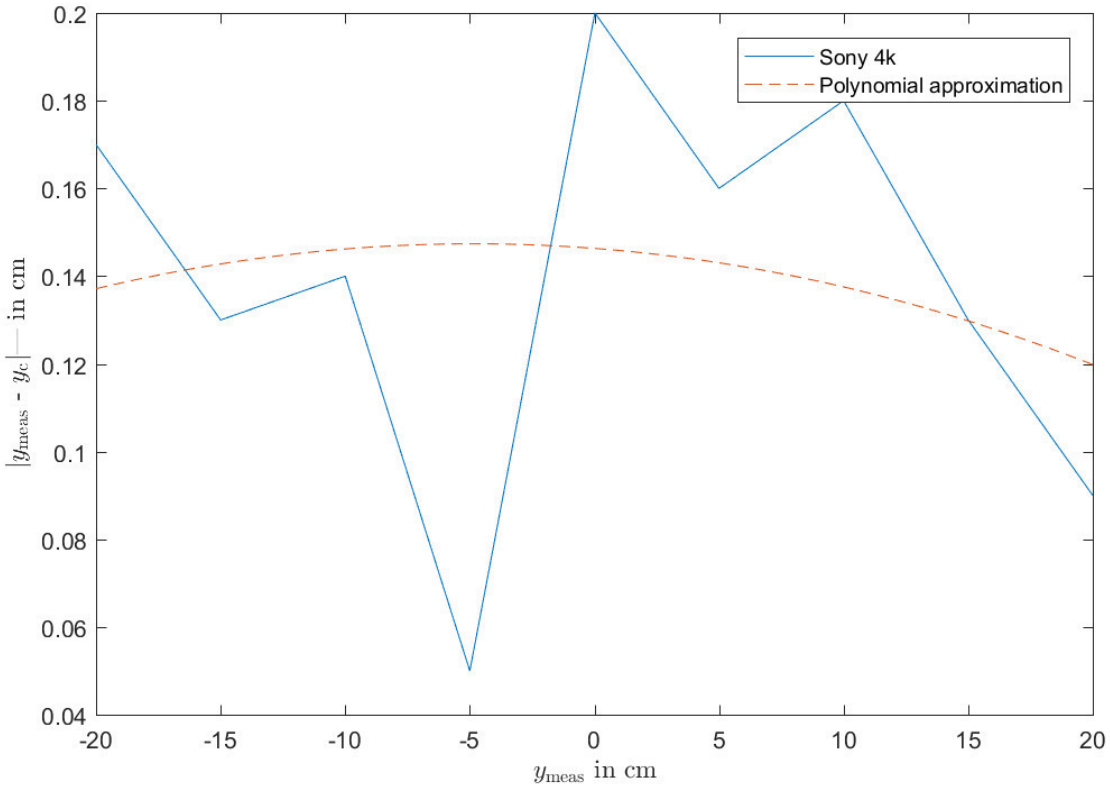

Figure 56: Sony 4k Y axis differences results

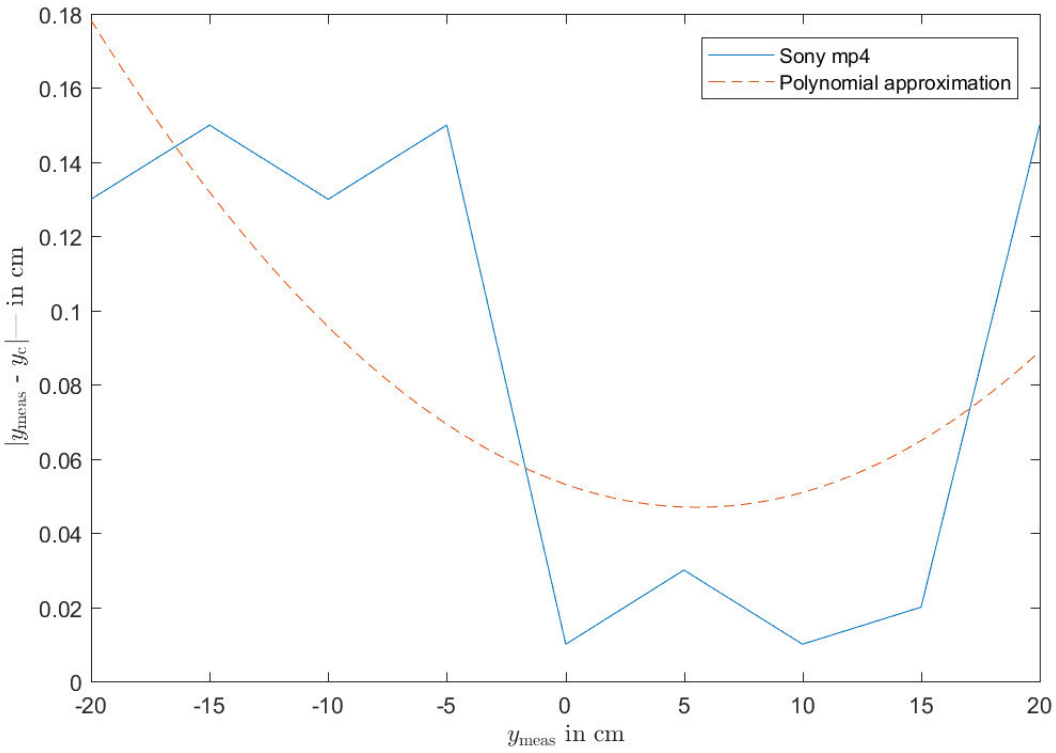

Figure 57: Sony mp4 Y axis differences results

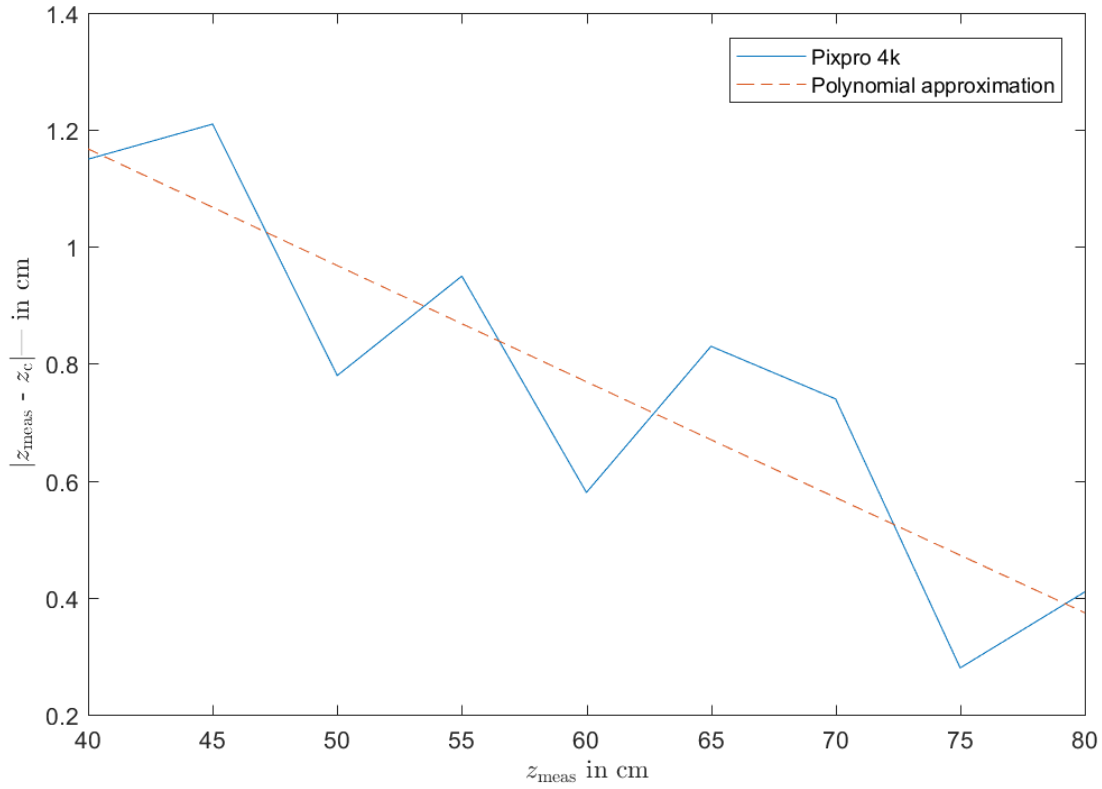

Figure 58: Pixpro 4k Z axis differences results

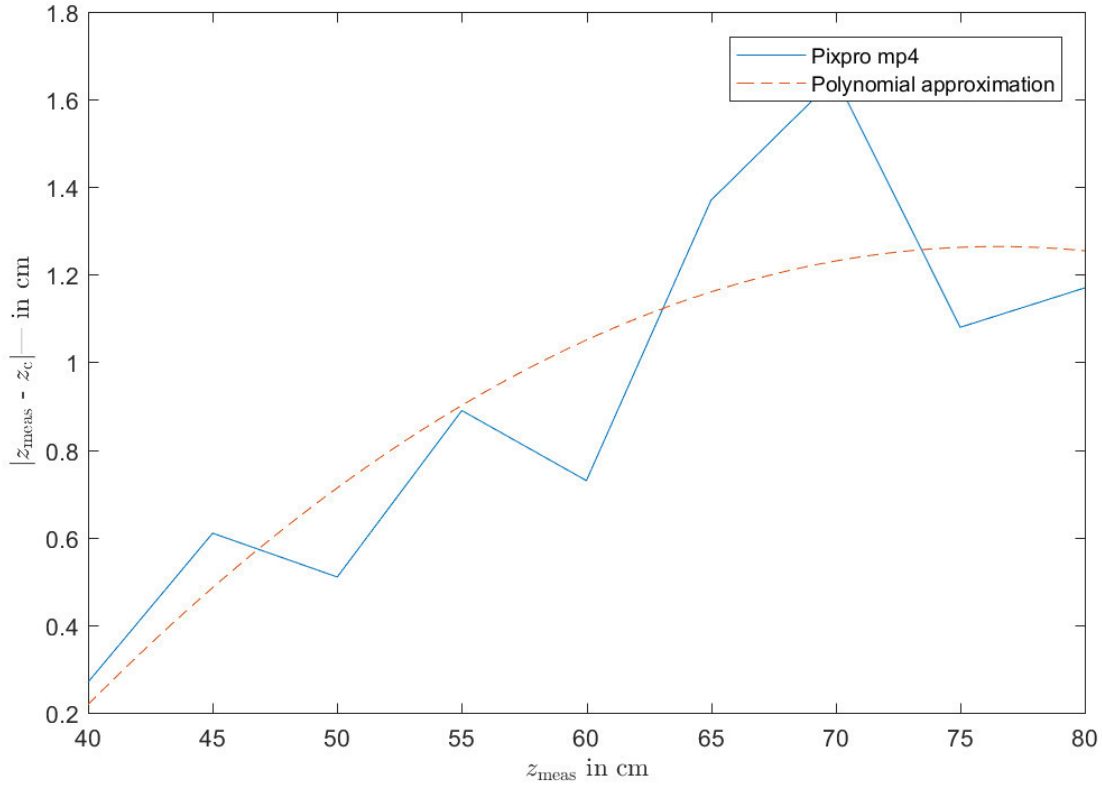

Figure 59: Pixpro mp4 Z axis differences results

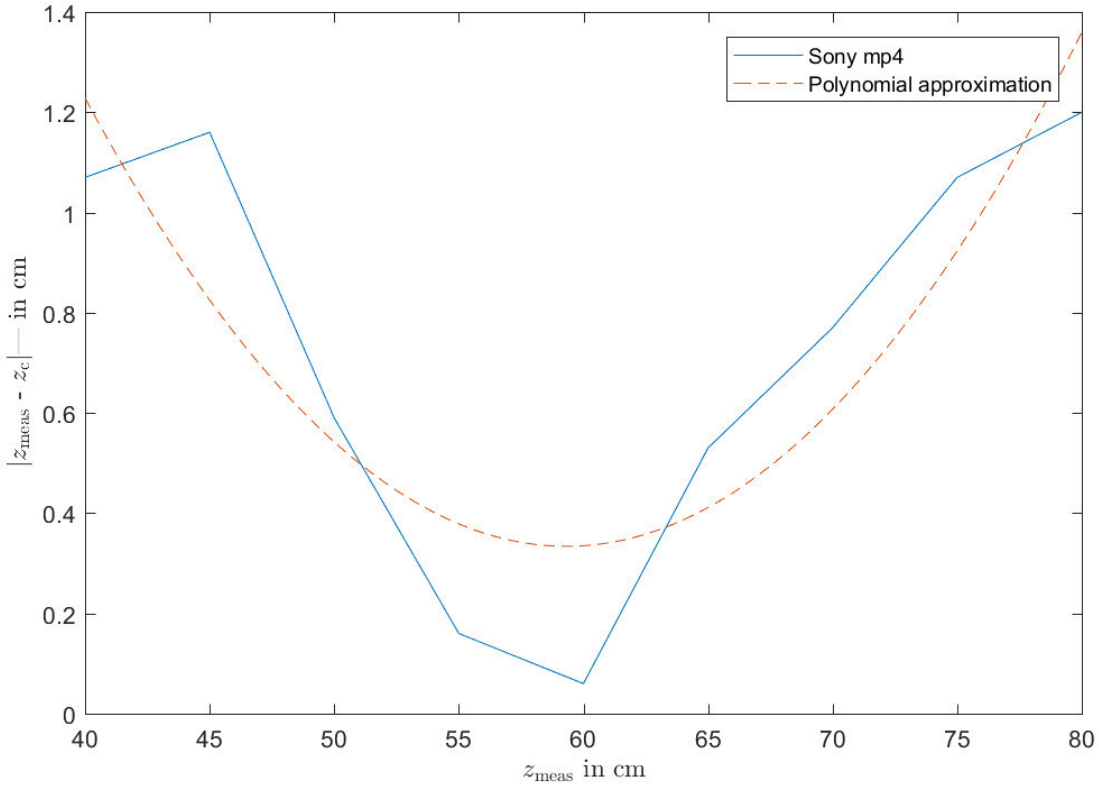

Figure 60: Sony mp4 Z axis differences results

Rotational tests:

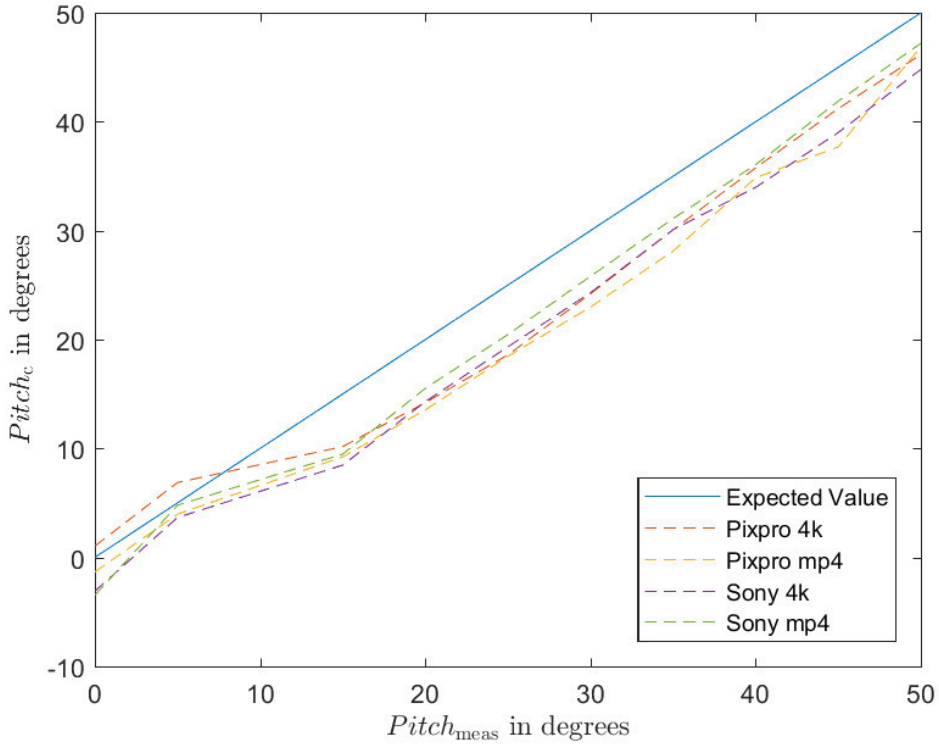

Figure 61: Pitch results

References

Abadi, M. et al.: TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems (3/14/2016)
Abadi, Martín; Agarwal, Ashish; Barham, Paul; Brevdo, Eugene; Chen, Zhifeng; Citro, Craig; Corrado, Greg S.; Davis, Andy; Dean, Jeffrey; Devin, Matthieu; Ghemawat, Sanjay; Goodfellow, Ian; Harp, Andrew; Irving, Geoffrey; Isard, Michael; Jia, Yangqing; Jozefowicz, Rafal; Kaiser, Lukasz; Kudlur, Manjunath; Levenberg, Josh; Mane, Dan; Monga, Rajat; Moore, Sherry; Murray, Derek; Olah, Chris; Schuster, Mike; Shlens, Jonathon; Steiner, Benoit; Sutskever, Ilya; Talwar, Kunal; Tucker, Paul; Vanhoucke, Vincent; Vasudevan, Vijay; Viegas, Fernanda; Vinyals, Oriol; Warden, Pete; Wattenberg, Martin; Wicke, Martin; Yu, Yuan; Zheng, Xiaoqiang: TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems, 3/14/2016

Agnus, V. et al.: Illumination Independent and Accurate Marker Tracking Using Cross-Ratio Invariance (2015)
Agnus, Vincent; Nicolau, Stephane; Soler, Luc: Illumination Independent and Accurate Marker Tracking Using Cross-Ratio Invariance, in: IEEE computer graphics and applications (5), Issues 35, pp. 22–33, 2015

Ang, M.; Tourassis, V.: Singularities of Euler and Roll-Pitch-Yaw Representations (1987)
Ang, Marcelo; Tourassis, Vassilios: Singularities of Euler and Roll-Pitch-Yaw Representations, in: IEEE Transactions on Aerospace and Electronic Systems (3)AES-23, pp. 317–324, 1987

Balaji, S.; Murugaiyan, M.S.: WATEERFALLVs V-MODEL Vs AGILE: A COMPARATIVE STUDY ON SDLC (2014)
Balaji, S.; Murugaiyan, M.Sundararajan: WATEERFALLVs V-MODEL Vs AGILE: A COMPARATIVE STUDY ON SDLC, in: nternational Journal of Information Technology & Business Management, Issues 22, 2014

Barreto, J. P.; Araujo, H.: Issues on the geometry of central catadioptric image formation (2001)
Barreto, J. P.; Araujo, H.: Issues on the geometry of central catadioptric image formation, in: Computer Vision and Pattern Recognition (CVPR 2001), I E E E [Imprint]; IEEE Computer Society Press, Los Alamitos, Jan. 2001

Bradski, G. R.; Kaehler, A.: Learning OpenCV (2011)
Bradski, Gary R.; Kaehler, Adrian: Learning OpenCV, Software that sees, 1. Edition, O'Reilly, Beijing, 2011

Cheng, H.; Gupta, K. C.: An Historical Note on Finite Rotations (1989)
Cheng, Hui; Gupta, K. C.: An Historical Note on Finite Rotations, in: Journal of Applied Mechanics (1), Issues 56, p. 139, 1989

Computer Vision and Pattern Recognition (CVPR 2001) (2001) Computer Vision and Pattern Recognition (CVPR 2001), I E E E [Imprint]; IEEE Computer Society Press, Los Alamitos, 2001

Garrido-Jurado, S. et al.: Automatic generation and detection of highly reliable fiducial markers under occlusion (2014)
Garrido-Jurado, S.; Muñoz-Salinas, R.; Madrid-Cuevas, F. J.; Marín-Jiménez, M. J.: Automatic generation and detection of highly reliable fiducial markers under occlusion, in: Pattern Recognition (6), Issues 47, pp. 2280–2292, 2014

Garrido-Jurado, S. et al.: Generation of fiducial marker dictionaries using Mixed Integer Linear Programming (2016)
Garrido-Jurado, S.; Muñoz-Salinas, R.; Madrid-Cuevas, F. J.; Medina-Carnicer, R.: Generation of fiducial marker dictionaries using Mixed Integer Linear Programming, in: Pattern Recognition, Issues 51, pp. 481–491, 2016

Gonçalves, L. M.; Vision, W. o.: WVC 2017 (2017)
Gonçalves, Luiz M. G.; Vision, Workshop o. C. (Eds.) WVC 2017, IEEE, Piscataway, NJ, 2017

Hartley, R. I.: Theory and Practice of Projective Rectification (1999)
Hartley, Richard I.: Theory and Practice of Projective Rectification, in: International Journal of Computer Vision (2), Issues 35, pp. 115–127, 1999

Hofmann, P.: Master's thesis, Object Detection and Tracking with Side Cameras and RADAR in an Automotive Context (2013)
Hofmann, Peter: Object Detection and Tracking with Side Cameras and RADAR in an Automotive Context, Master's thesis
Freie Universität Berlin, Berlin, 2013

Insafutdinov, E. et al.: ArtTrack: Articulated Multi-person Tracking in the Wild (2016)
Insafutdinov, Eldar; Andriluka, Mykhaylo; Pishchulin, Leonid; Tang, Siyu; Levinkov, Evgeny; Andres, Bjoern; Schiele, Bernt: ArtTrack: Articulated Multi-person Tracking in the Wild, 2016

Insafutdinov, E. et al.: DeeperCut: A Deeper, Stronger, and Faster Multi-Person Pose Estimation Model (2016)
Insafutdinov, Eldar; Pishchulin, Leonid; Andres, Bjoern; Andriluka, Mykhaylo; Schiele, Bernt: DeeperCut: A Deeper, Stronger, and Faster Multi-Person Pose Estimation Model, 2016

Kühling, C.: Masterarbeit am Institut für Informatik, Fisheye Camera System Calibration for Automotive Applications (2017)
Kühling, Christian: Fisheye Camera System Calibration for Automotive Applications, Masterarbeit am Institut für Informatik
Freien Universität Berlin, Berlin, 2017

Lienhart, R.; Maydt, J.: An extended set of Haar-like features for rapid object detection (2002)
Lienhart, R.; Maydt, J.: An extended set of Haar-like features for rapid object detection, in: Proceedings. International Conference on Image Processing, IEEE, 22-25 Sept. 2002

Mallick, S.: Head Pose Estimation using OpenCV (2016)
Mallick, Satya: Head Pose Estimation using OpenCV; https://www.learnopencv.com/head-pose-estimation-using-opencv-and-dlib/, 2016, Access 07.08.2018

Mallon, J.; Whelan, P. F.: Precise radial un-distortion of images (2004)
Mallon, J.; Whelan, P. F.: Precise radial un-distortion of images, in: Kittler, Josef (Ed.): Pattern Recognition, IEEE Computer Society Press; IEEE [distributor], Los Alamitos, Piscataway, Jan. 2004

Martinez, J. et al.: A simple yet effective baseline for 3d human pose estimation (2017)
Martinez, Julieta; Hossain, Rayat; Romero, Javier; Little, James J.: A simple yet effective baseline for 3d human pose estimation, 2017

Mathur, S.; Malik, S.: Advancements in the V-Model (2010)
Mathur, Sonali; Malik, Shaily: Advancements in the V-Model, in: International Journal of Computer Applications (12), Issues1, pp. 30–35, 2010

MeiProjectionModel. (2018) MeiProjectionModel.; http://www.robots.ox.ac.uk/~cmei/, 2018

Muñoz-Salinas, R.: Camera Model
Muñoz-Salinas, Rafael: Camera Model, Spain

Muñoz-Salinas, R. et al.: Mapping and localization from planar markers (2018)
Muñoz-Salinas, Rafael; Marín-Jimenez, Manuel J.; Yeguas-Bolivar, Enrique; Medina-Carnicer, R.: Mapping and localization from planar markers, in: Pattern Recognition, Issues 73, pp. 158–171, 2018

Nabil Mohammed, A. M.; Govardhan, A.: Comparison Between Five Models Of Software Engineering (2010)
Nabil Mohammed, Ali M.; Govardhan, A.: Comparison Between Five Models Of Software Engineering, in: International Journal of Computer Science Issues, Issues 7, 2010

opencv/opencv opencv/opencv; https://github.com/opencv/opencv, Access 08/09/2018

OpenCV: Fisheye camera model OpenCV: Fisheye camera model; https://docs.opencv.org/trunk/db/d58/group__calib3d__fisheye.html, Access 08/09/2018

Rook, P.: Controlling software projects (1986)
Rook, Paul: Controlling software projects, in: Software Engineering Journal (1), Issues1, p. 7, 1986

Rufli, M. et al.: Automatic detection of checkerboards on blurred and distorted images (2008)
Rufli, M.; Scaramuzza, D.; Siegwart, R.: Automatic detection of checkerboards on blurred and distorted images, in: IEEE/RSJ International Conference on Intelligent Robots and Systems, 2008, IEEE Service Center, Piscataway, NJ, 2008

Scaramuzza, D. et al.: A Toolbox for Easily Calibrating Omnidirectional Cameras (2006)
Scaramuzza, Davide; Martinelli, Agostino; Siegwart, Roland: A Toolbox for Easily Calibrating Omnidirectional Cameras, in: IROS 2006, IEEE, [Piscataway, N.J.], 2006

Söderroos, A.: Fisheye Camera Calibration and Image Stitching for Automotive Applications (2015)
Söderroos, Anna: Fisheye Camera Calibration and Image Stitching for Automotive Applications, 2015

Soo, S.: Object detection using Haar-cascade Classifier (2015)
Soo, Sander: Object detection using Haar-cascade Classifier
University of Tartu, Tartu, 2015

Szeliski, R.: Computer vision (2011)
Szeliski, Richard: Computer vision, Texts in computer science, Springer, London, 2011

Viola, P.; Jones, M.: Rapid object detection using a boosted cascade of simple features (2001)
Viola, P.; Jones, M.: Rapid object detection using a boosted cascade of simple features, in: Computer Vision and Pattern Recognition (CVPR 2001), I E E E [Imprint]; IEEE Computer Society Press, Los Alamitos, Jan. 2001

Wei, S.-E. et al.: Convolutional Pose Machines (1/30/2016)
Wei, Shih-En; Ramakrishna, Varun; Kanade, Takeo; Sheikh, Yaser: Convolutional Pose Machines, 1/30/2016

Wilson, P. I.; Dr. John Fernandez: Facial feature detection using Haar classifiers (2006)
Wilson, Phillip I.; Dr. John Fernandez: Facial feature detection using Haar classifiers, in: Journal of Computing Sciences in CollegesVolume 21, pp. 127–133, 2006

Zhang, Z.: A flexible new technique for camera calibration (2000)
Zhang, Z.: A flexible new technique for camera calibration, in: IEEE Transactions on Pattern Analysis and Machine Intelligence (11), Issues 22, pp. 1330–1334, 2000