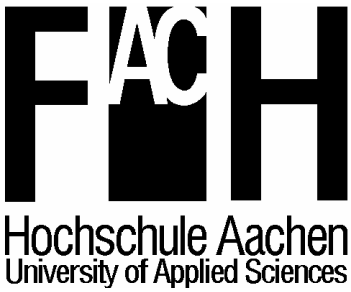


Fachhochschule Aachen

Fachbereich
Maschinenbau und Mechatronik

Mechatronics

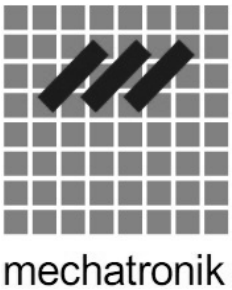
Human / Robot Interaction in Pick & Place Operations with Optical Tracking

Miguel Angel Garcia Figueroa

Matr.-Nr.: 346041

Referent : **Prof. Dr.-Ing. Günther Starke**

Korreferent : **M. Sc. Thomas Kunkel**



August 2009

In collaboration with APS - European Centre for Mechatronics, Aachen, NRW.

Declaration

Hereby I declare that this thesis is my own work and that I did not use other sources than the literature and websites listed in the appendix.

Passages taken verbatim or analogously from published or non-published sources are denoted accordingly.

Pictures and drawings were created by myself or are referred to the used sources correspondingly.

This work was handed in neither in equivalent nor similar form at any other examination authority.

Secrecy

Acknowledgment

I would like to thank my parents for all their invaluable support, guidance, encouragement and confidence in me that have been strong motivators along my life but especially on this project.

I would like to thank Prof. Dr-Ing. Günther Starke to give me the opportunity to realize this thesis in APS - European Centre for Mechatronics, and all the guides for complete this work.

I am thankful to M.Sc. Thomas Kunkel for all the support in programming, especially with Qt, and carrying out this thesis work.

Furthermore, I want to thank all other employees for their friendliness and the pleasant atmosphere they created, which I enjoyed very much during my work at APS.

Lastly, I would like to thank to all my friends in Mexico and in Germany and to all the people that were part of my life.

Abstract

This thesis presents a human/robot interaction in pick & place operations with optical tracking. The optical tracking consisted of in a complete vision system. It was developed using OpenCV Library integrated in Qt creator. A graphical user interface was designed, which contains the necessary requirements in order to achieve an optical tracking. The whole project is based on three main algorithms: Camera Calibration Algorithm, Object Detection Algorithm and Tracking algorithm. A webcam is used for capturing video. Hence the camera calibration has to be carried out. The parameters from the calibration are applied for solving the radial distortion. Handling of objects is the main goal. An object detection method is realized, which is based on geometry of the objects. The geometry is analyzed through a contour processing. Therefore the Canny Edge Detection algorithm is applied. The detection for the mark is the same method used for the objects. Optical tracking is achieved using the CamShift algorithm. CamShift permits to track the mark through camera view, and it is always possible to know the actual position of the mark. The main functions are provided from OpenCV Library, tests and comparisons between them were realized in order to obtain a better response.

The robot system consists of an industrial robot with 6 degrees of freedom. This robot is mounted to a gantry robot. A gripper for the pick & place operation has been mounted onto the robot. The communication within the system robots and user interface was based on network technology. The final results obtained from each algorithm were completely successfully. The camera calibration algorithm was able to solve the radial distortion. Therefore the response of the object detection algorithm was a linear response. Due to undistortion the system presented high accuracy. The tracking based on CamShift algorithm is one of the functions that OpenCV Library provides. The definition of the color as feature to tracking is a powerful option for this application.

- Table of Contents -

1 Introduction

Nowadays the applications of the robots are more common. Principally the interaction human/robot takes an important part in this field. The most relevant related are the remote application. The user is able to control certain activity from another place. When the activity consisted of to enable robots to assist human in handling of objects and pick & place operations without explicit programming, appropriate human-to-machine communication concepts with multimodal input as well as adaptive functions are required to meet these goals.

One approach towards advanced human-to-machine communication with multimodal input is the use of gestures to guide and command a robot system. Therefore the applicability of optical tracking (with one camera) could be evaluated to detect and interpret gestures. Focus could be given to gestures provided by the human hands to command a robot to pick up objects pointed at and to place them at pointed target positions. Additionally a functionality could be available which allows to track the motion of the hands in order to enable teach-in by demonstration.

Another approach is to define different features for instance the color. It is possible to achieve when a color distribution is calculated. Therefore the optical tracking could be evaluated to detect and interpret a flesh hue distribution. The mean shift algorithm operates on probability distributions. To track colored objects in video frame sequences, the color image data has to be represented as a probability distribution using color histograms to accomplish this. Color distributions derived from video image sequences change over time, hence the mean shift algorithm has to be modified to adapt dynamically to the probability distribution it is tracking.

An optical tracking was implemented using the color as feature. The detection of the mark is based on the geometry of it. To attain these tasks, the functions provides by OpenCV Library were the base of this project.

2 State of Art

2.1 Open Source Computer Vision Library and Qt Creator

The method proposed for this application is based on C++ programming environment. The objective is to design a graphical user interface, which is going to contain the necessary requirements and tools in order to keep a friendly environment for the user but at the same time, its response and utilities must be powerful. Considering these characteristics, Qt creator is defined as the one of the best options for developing this application, even Qt offers many good qualities for programmers, because provides the Qt designer keeping the Object Oriented Programming (OOP) structure. Allowing that an image processing is needed for this application, Source Computer Vision Library (OpenCV Library) is a very good option, because its main library provides many different functions, all of them are related with image and video applications. Given these both proposes the idea is to integrate the OpenCV libray to Qt creator. Therefore, every process related with images and video capturing is going to develop applying OpenCV library functions, all of them running on Qt environment.

2.1.1 Qt Creator

Qt Creator is a complete integrated development environment (IDE) for creating applications with the Qt application framework. Qt is designed for developing applications and user interfaces once and deploying them across several desktop and mobile operating systems.

Qt Development tools:

- An advanced C++ code editor and integrated Qt Designer
- Project and build management tools
- Integrated, context-sensitive help system
- Visual debugger using a graphical user interface with increased awareness of Qt class structures
- Code management and navigation tools

New project wizard for Qt4:
The most important characteristic and the easiest way for creating a new project in Qt Creator it is aided by a wizard that guides the user through the project creation process in consecutive steps. In the first step the user selects the type of the project from the categories: Qt console application, Qt GUI (graphical user interface) application, or Qt library. Next, the user can select a location for the project, Qt specific settings (such as select the needed modules), before specifying the details of the first class of the application. When the steps have been completed, Qt Creator

will automatically generate the project with required header, source, user interface and project files as defined by the wizard.

2.1.1.1 Object Oriented Programming

The program structure is based on Object Oriented Programming, hence it is suitable to do a briefly explanation about the basic concepts. As already mentioned in the preceding section, the software used for this proposed is Qt creator, its environment is used for creating the graphical user interface and in combination with OpenCV library, which the image processing and video capturing is to achieve through this powerful library.

The Object Oriented Paradigm is divided in four basic principles [21]:

- Abstraction
- Encapsulation
- Modularity
- Hierarchy

Abstraction allows managing complexity by concentrating on the essential characteristics that makes an entity different from others. Object Orientation (OO) uses abstraction to depict classes and objects in a system.
Encapsulation separates the implementation of objects behavior from its public interface. The “information hiding” allows the behavior of the objects to be used without knowing its implementation. Encapsulation offers two kinds of security: protects the internal state of the objects from being changed from outside users. And changes can be done to the behavior without affecting other objects
Modularity can be defined as the process of breaking up of a big system into small, self contained pieces that can be managed easily. Packages support the definition of the modularity. Classes are independent modules.
Hierarchy means that any ranking or ordering of abstractions into a tree-like structure

Types of hierarchies:

- Class
- Aggregation
- Inheritance
- Specialization

Basic Concepts of Object Orientation:

- Object
- Class
- Attribute
- Operation

- Polymorphism
- Component
- Relationships
- (Package)

Object has different definitions because it is a concrete manifestation of an abstraction. It is an entity with a well-defined boundary and identity that encapsulates state and behavior, and also it is an instance of a class.
An object has:
– State: set of conditions in which an object exists, implemented by a set of properties, which are known as attributes, with their values and the relationships with other objects.
– Behavior: the way the object interacts with other objects, its public interface
– Identity: makes two objects different even in case they have the same state and behavior
Classes are in general, but in particular a class is a description of a set of objects that share the same attributes, operations, relationships, and semantics. A class can have objects, which are known as instances.
A class is an abstraction:
– emphasizes relevant characteristics
– hides other characteristics

In Figure 2-1, an example of a class is depicted. It is the representation according with Unified Modeling Language (UML) notation.

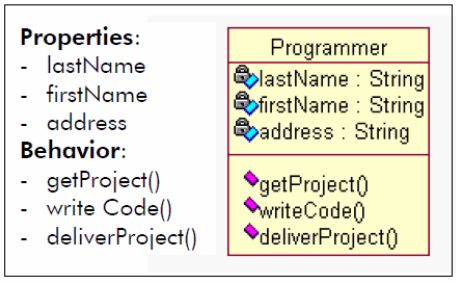


Figure 2-1: Example of a class [21]

Attributes, in particular an attribute is a named property of a class that describes the range of values that instances of the property may hold. An attribute has a data type that defines the type of its instances. Only the object itself should be able to change the value of its attributes. The values of the attributes define the state of the object.
Operation is the implementation of a service that can be requested from any object of the class to affect behavior.
The representation of both is depicted in Figure 2-2. It is according with UML class diagram.

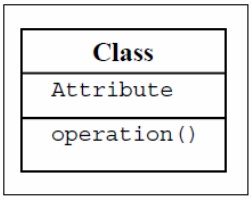


Figure 2-2: Representation of a class with its corresponding parts [21]

A *relationship* is a connection among things. UML uses these kinds of relationships:
– Association
- Aggregation
- Composition

– Generalization

- Association is a structural relationship that describes a set of links, in which a link is a connection among objects. It represents the semantic relationship between two or more classifiers that involves the connections among their instances. The association allows bi-directional navigation between two classes. Classes involved in an association can be the same.

Association: Aggregation
- Models a whole/part relationship
- Both classes are conceptually at the same level
- Represents a “has-a” relationship
- The whole does not own the part

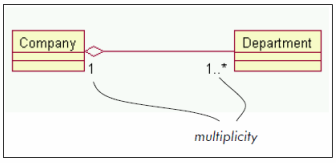


Figure 2-3: Representation of Aggregation relationship [21]

Figure 2-3 shows an aggregation relationship. The program structure is based on this type of relationship. In principle, these concepts are sufficient in order to understand the principal idea of the program structure proposed, it is shown in Chapter 3. Source gives more details about OO, and in general any book about C++ programming.

2.1.2 OpenCV Library

OpenCV means Intel® Open Source Computer Vision Library. It is a collection of C functions, and few C++ classes that implement some popular algorithms of Image Processing and Computer Vision. OpenCV is cross-platform middle-to-high level API,

that consists of a few hundreds (>300) C functions. It does not rely on external numerical libraries, though it can make use of some of them at runtime, if they are available. OpenCV is free for both non-commercial and commercial use.

2.1.2.1 Features

OpenCV has many capabilities, and the major functionality categories as Image Processing, and Computer Vision algorithms. These include edge, line, and corner detection, ellipse fitting, image pyramids for multiscale processing, template matching, various transforms (Fourier, discrete cosine, and distance transform), and more, which are located in the manual reference. Also OpenCV includes several high-level capabilities such as face detection, recognition and tracking as well, optical flow (using camera motion to determine 3D structure), camera calibration, and stereo. Artificial Intelligence (AI) and machine learning methods, computer vision applications often require machine learning, or other AI methods. Some of these are available in OpenCV Machine Learning package. Image sampling and view transforms, it is often useful to process a group of pixels as a unit. OpenCV includes interfaces for extracting image subregions, random sampling, resizing, warping, rotating, and applying perspective effects. Methods for creating and analyzing binary (two-valued) images, binary images are frequently used in inspection systems that scan for shape defects or count parts. A binary representation is also convenient when locating an object to grasp. Methods for computing 3D information, these functions are useful for mapping and localization - either with a stereo rig or with multiple views from a single camera. Math routines for image processing, computer vision, and image interpretation, OpenCV includes math commonly used algorithms from linear algebra, statistics, and computational geometry. Include also Graphics, these interfaces write text and draw on images. In addition to various fun and creative possibilities, these functions are useful for labeling and marking. GUI methods, OpenCV includes its own windowing interfaces. While these are limited compared to what can be done on each platform, they provide a simple, multi-platform API to display images, accept user input via mouse or keyboard, and implement slider controls. Data structures and algorithms, with these interfaces, you can efficiently store, search, save, and manipulate large lists, collections (also called sets), graphs, and trees. Data persistence, these methods provide convenient interfaces for storing various types of data to disk and retrieving them later.

2.1.2.2 Modules

OpenCV has several modules. Each module contains the specific functions already mentioned as features. The next paragraphs are a briefly description about them.
CV or CVREFERENCE is one of the most important and complete module. It contains the major quantity of the image processing functions, related with the

specific functions used for structural analysis, as well motion analysis and object tracking reference. Also it provides pattern recognition, camera calibration and 3D reconstruction. More details about this module they are given by OpenCV manual reference [1], and also directly on line, internet resources.

CXCORE contains basic data type definitions. For example, the data structures for image, point, and rectangle are defined in cxtypes.h. CXCORE also contains linear algebra and statistics methods, the persistence functions, and error handlers. Somewhat oddly, the graphics functions for drawing on images are located here as well.

CVAUX is described in OpenCV documentation as containing obsolete and experimental code. However, the simplest interfaces for face recognition are in this module. The code behind them is specialized for face recognition, and they are widely used for that purpose. Its main sections are stereo correspondence functions, view morphing functions, 3D tracking functions, eigen objects functions and embedded hidden markov model functions.

HIGHGUI is just an addendum for quick software prototypes and experimentation setups. The general idea behind its design is, to have a small set of directly useable functions to interface your computer vision code with the environment. It is located in the directory named “otherlibs”. HighGUI contains the basic I/O interfaces, also contains the multi-platform windowing capabilities and video interfaces.

It is important to mention, these modules permit to see a detail description about all the functions already developed. These modules also are very helpful in order to know all the members for each function, and how to have access to each one.

2.1.2.3 Functions and Types of Variables

During development process of the project many tasks are required. Functions and variables from OpenCV library were implemented. Therefore a short explanation about them is mentioned. Table 3 and Table 4 in Appendix show the most important variables and functions respectively, also in the next Chapters a short description is explained at the moment when the function or the variable are applied. OpenCV reference manual gives detail information [1].

2.1.2.4 Integration in Qt Creator

Qt's Build System Projects are described by .pro files that contain terse, but readable descriptions of source and header files, Qt Designer forms, and other resources. These are processed by the qmake tool to produce suitable Makefiles for the project on each platform.

For using OpenCV library on Qt creator based on Qt 4.5.0, open the file *.pro already mentioned.

Before to type the code lines, firstly it is necessary to add the libraries, just the *.lib, these libraries are located “C:\Program Files\OpenCV\lib”, or it depends where the user has installed the OpenCV.

Copy the next files:

cv.lib cvaux.lib cxcore.lib highgui.lib

These files must be paste in the next folder:

C:\ 2009.01\qt\lib

this is required because of the next code lines in the *.pro file.

A new Qt project in the *.pro file normally contain these code lines:

```
TARGET = Test_video_recognition_tutorial
TEMPLATE = app
SOURCES += main.cpp\
           mainwindow.cpp
HEADERS += mainwindow.h
FORMS += mainwindow.ui
```

And these code lines are added for using OpenCV library:

```
LIBS += -lcvaux
LIBS += -lcv
LIBS += -lcxcore
LIBS += -lhighgui

INCLUDEPATH += "C:\PROGRAM FILES\OPENCV\CV\INCLUDE"
INCLUDEPATH += "C:\PROGRAM FILES\OPENCV\CXCORE\INCLUDE"
INCLUDEPATH += "C:\PROGRAM FILES\OPENCV\CVAUX\INCLUDE"
INCLUDEPATH += "C:\PROGRAM FILES\OPENCV\OTHERLIBS\HIGHGUI"
```

And then with this, it is possible to declare the headers in the *.cpp file and *.h, as this:

```
#include <cv.h>
#include <cvaux.h>
#include <highgui.h>
#include <cxcore.h>
```

The procedure is not complicate, and also it is needed, when an OpenCV function is used in some file, in some class, whether the file is *.cpp or *.h, it does not care, the OpenCV headers must be written.

At the moment when Qt recognizes the OpenCV functions, there are no errors in compilation, after that, when a valid OpenCV function is typing one label appears, it shows the correct syntax, for instance, all the elements that the function needs, and what type of elements are them.

2.2 Hardware

The application is integrated by different elements. Motoman SV3X, XRC 2001 controller, gantry robot, and gripper fixed to the motoman, camera and the object defined for this task. The objects are presented in order to how the objects are. The motoman SV3X consist of an industrial robot. It has full 6 axis capability, hence provides high flexibility. There mountable on the floor, wall and ceiling. The Motoman SV3X has advanced robot motion control, which is XRC 2001 controller. It provides powerful tools for operating the motoman. It also offers optional DeviceNet, ControledNet, Profibus-DP and Interbus –S. Therefore it is possible to integrated this advance robot motion control to a network and thorough the connection with others devices a particular task can be realized. For the application proposed the robot is instructed from to remote station. According with the objective of the project, handling operations can be to achieve. Hence a gripper was added to the system in order to do this kind of tasks. The motoman is mounted in a gantry robot, which is depicted in Figure 2-5. It provides also high accuracy to the motoman, in this case, for handling the position of the motoman is very helpful. Figure 2-4 shows the motoman robot SV23X and the gripper already fixed to the motoman. Technical specifications data for Motoman are listed in Table 6 and Table 7 in Appendix.

Figure 2-4: Motoman Robot SV3X

Figure 2-5: Gantry Robot

With the gripper shown it is possible to attain handling of certain objects. In particular for this application the objects already defined have the next characteristics: diameter of 50mm, height of 50mm. There are different colors for the objects. They are blue, red and green. Figure 2-6 illustrates the object, and it is possible to see the characteristics of them.

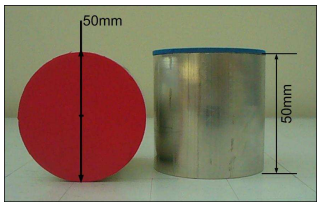


Figure 2-6: Objects

For handling operation the objects are detected, hence the camera used is a webcam Logitech Pro 9000, which offers different advantages. It has a higher megapixel performance and also it is possible to get more detail and clarity. Some further details about the specifications and technical requirements are listed at the Appendix (see Table 5 in Appendix). Figure 2-7 illustrates the camera employed.

Figure 2-7: Webcam Logitech Pro 9000

The objects as well as the camera already were defined. Concerning with the network communication the modules were already developed. Hence the communication with the robot was easy to achieve.

3 Development Processes

The method proposed is an interactive software tool with graphical user interface, where the main goal is to develop a complete system to enable robots to assist human in handling of objects and pick & place operations with optical tracking. To attain the main goal, the complete process is divided in different stages, which have a common objective. The algorithms are presented. Their development process is explained separately, this is in order to give details about the steps and considerations involves in each one.

Firstly the overall architecture is presented, the tools used for this proposed and also their requirements and constraints take an important part of the system. The final response and also the expected behavior both are directly depending of the devices employed for realizing the different tasks.

3.1 Overall Architecture

The elements of the overall architecture have been briefly described in the preceding Chapter. Therefore, robot system is depicted in Figure 3-1.

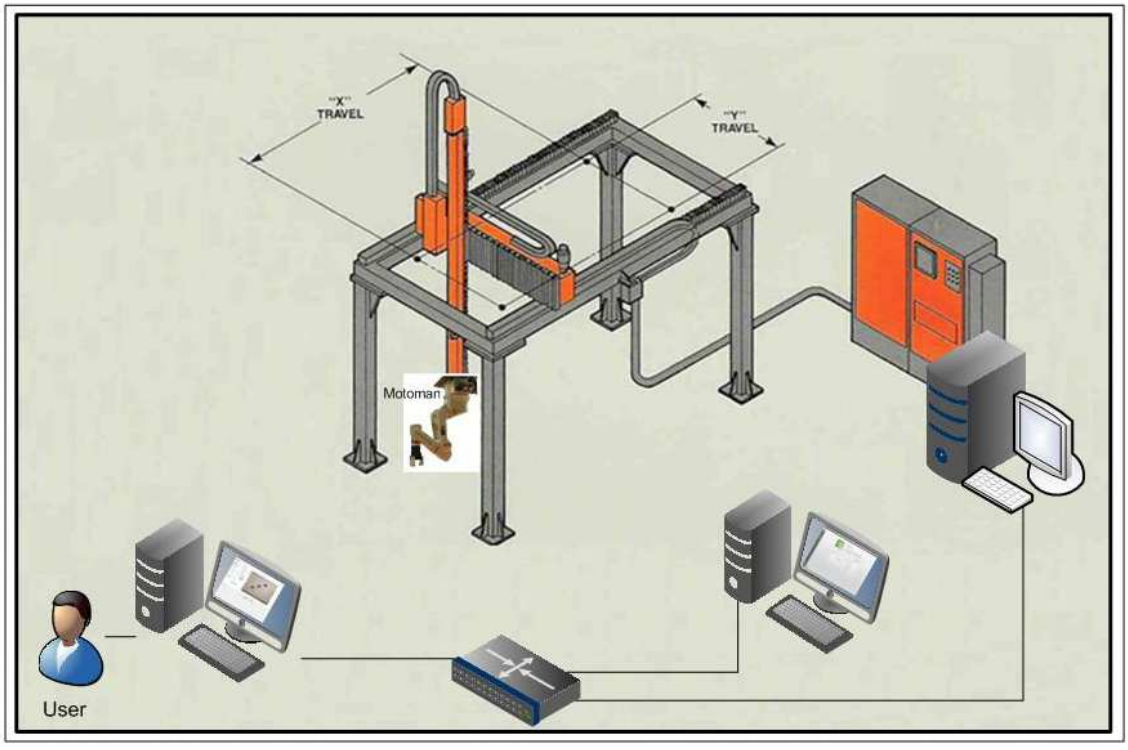


Figure 3-1: Overall Architecture of the system

When an object is detected, the data is sent. Figure 3-1 permits to appreciate the complete system, and with the robot grantry. This is a general idea of the

architecture. The communication between different remote stations is already established trough a local network. The tracking depends of the detection. The area for detecting is limited by the camera view. A place for the camera was defined because the camera is fixed as is shown in Figure 3-2. Hence the area for the robot is limited. If the camera is fixed at object detection, the camera is watching the objects in a different perspective according with the Figure 2-6 in Chapter 2. Now only the camera is able to see the surface of the objects because it is fixed. Regarding this important constrain a distance value for the camera related with the object plane also had to be defined. The value for the distance was considered because of the resolution of the camera. The camera has high quality is terms of resolution. The problem is that, the resolution is limited by OpenCV library. The resolution with the functions from OpenCV library work is 320x240. Due to this value of resolution, if the distance of the camera related with the object plane increase, the detection and tracking algorithms could be presented problems because the size of the objects under that condition is smaller. Also is not the best idea to increase the distance because it is not comfortable to show in the GUI the video with a short resolution. Figure 3-2 depicts the distance considered principally for detection and tracking algorithm. It is important to regard if this distance change, the system must be rescaling for sending the correct values to the robot in world coordinates system (this transformation is explained in Chapter 4).

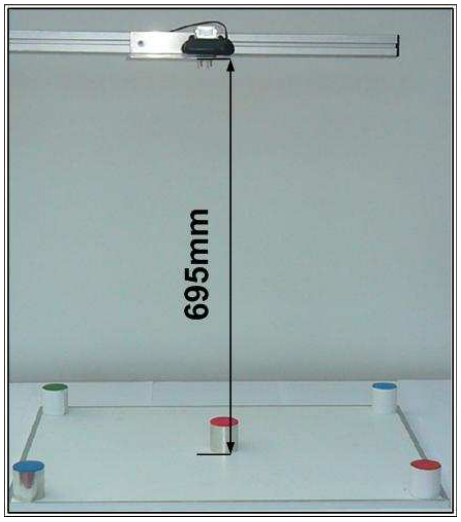


Figure 3-2: Physical distance used between the camera and object plane for this application

3.2 Graphical User Interface

The Graphical User Interface (GUI) is designed making allowance for all possible requirements that the final user needs. User modeling is normally defined to as a model containing relevant assumptions about the user of the target system. Thinking about this concept and also considering the main goal of this application, the basic

idea of the complete structure of the program is depicted in Figure 3-3. It is possible to see that, the complete structure is divided in different blocks, which represents one task.

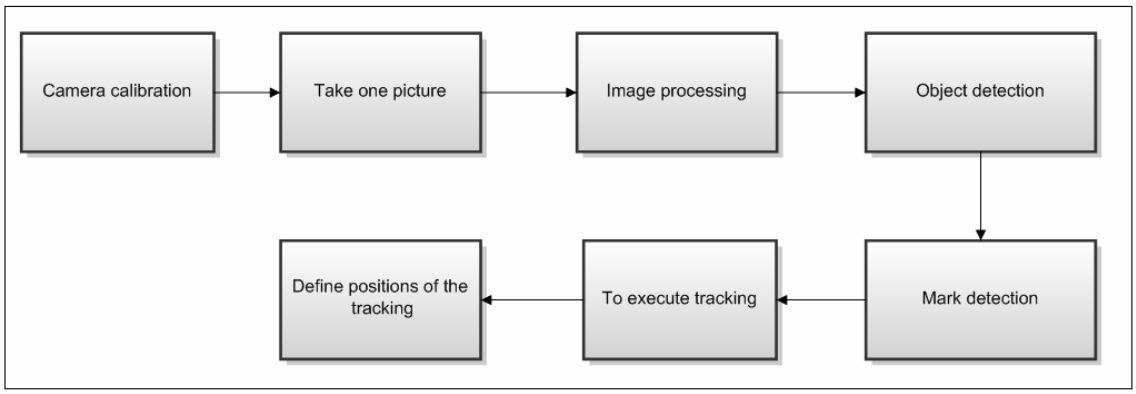


Figure 3-3: Basic model of the program structure

A webcam is used, therefore a calibration stage is needed. It is because of lens distortion, this is explained in the next section, and the final results are shown in Chapter 4. Hence the GUI is divided in two windows. One window contains all the necessary tools related with the camera calibration. It is shown in Figure 3-4. And the other window is the main window where the objects are detected and also the tracking system can be initialized. Also the GUI has certain permissions and restrictions. For example when the calibration is executed it is not possible to execute the tracking system at the same time.

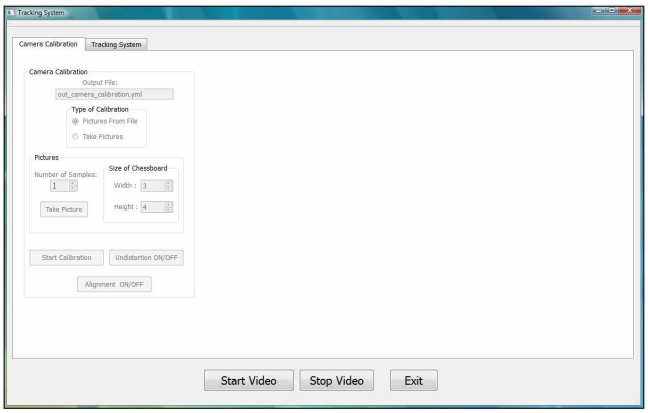


Figure 3-4: Graphical User Interface. Camera Calibration

The detail explanation about the content of both windows it is explained and described in Chapter 4.

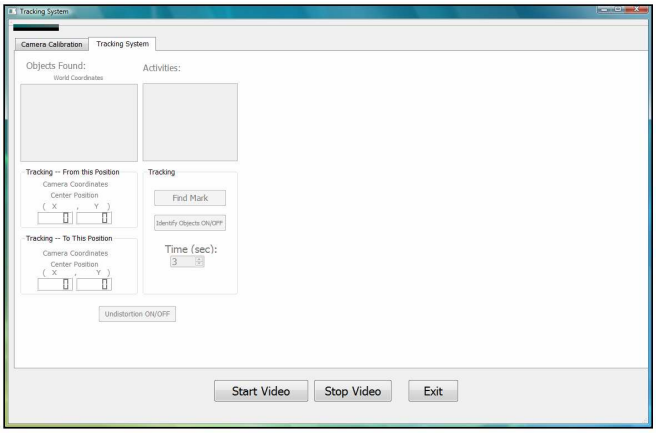


Figure 3-5: Graphical User Interface. Tracking System

Figure 3-6 shows the moment when the object detection is running, in this case it is after camera calibration because there is no lens distortion. This tracking window contains two different lists, in left side the list with the information about the position of the objects is shown, also their color is specified. The list in right side is related with the activities already executed. The information concerning with the tracking system also is located here. The actual position of the center mark is shown when the tracking is running and it is possible to define the time in seconds in order to load the positions of the tracking. The User is able to see if the position of the mark is valid or not, and at positions valid the time starts. The conditions and details about the tracking are deal in Chapter 4 when the experiments are to achieve.

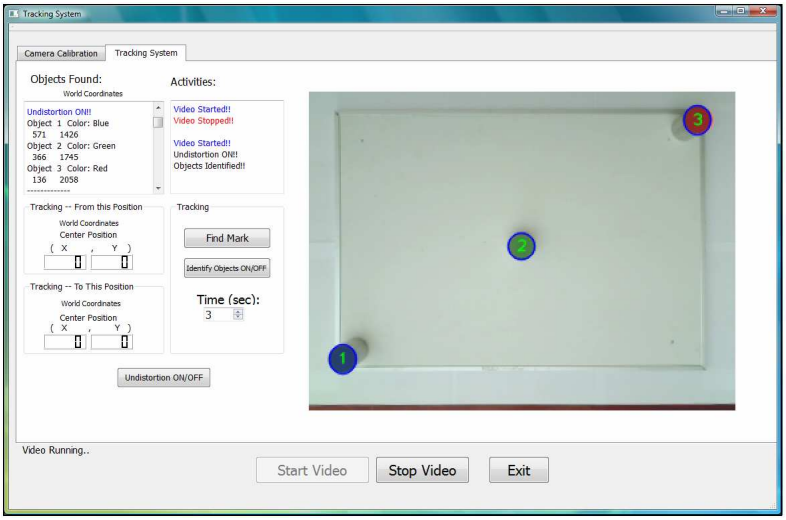


Figure 3-6: Program running. GUI Object detection

The GUI was developed used Qt creator, which is based on Object Oriented Programming. The OpenCV library was added to Qt creator. The resolution of the video supported by OpenCV functions is 320x240 pixels. OpenCV provides one module called HIGHGUI already explained in Chapter 2. This module contains the functions in order to generate the windows for showing the video, or even a button

can be created. But in this case Qt is powerful development software. It permits to create and also design a comfortable and complete GUI. Therefore, it was possible to adapt the video in Qt, thereby at the moment when the video is running before to show it, a change of the resolution of the video in the GUI is achieved. This new resolution is just in order to do more comfortable the GUI because the all the analysis are to do using a resolution of 320x240.

The structure of the program is divided in 7 different classes, and 1 main class called “MainWindow”. The 4 classes are related directly with the main class, which are CamCalibrator, ObjectDetector, MarkTrack, and qopencvwidget. The last one of these classes is used for integrating the video in Qt, thus the video can be shown in Qt environment.
In the case of CamCalibrator, it is a particular case, because only camera calibration is executed in this class. Therefore, the video is required in order to show the results given by calibration algorithm. Afterwards CamCalibrator is not used any more. ObjectDetector invokes EllipticObject and ColorDefiner. All of these classes are used for detecting. ObjectDetector find the contours and they are sent to EllipticObject, which analyze the contours and returns the objects located in the contour processing. After that ObjectDetector invokes ColorDefiner for sending the objects that have been found and ColorDefiner returns the color of each object. Lastly ObjectDetector is invoked for MainWindow and the objects that have been found are shown in the GUI. When the tracking is executed by the user, firstly the mark must be found. Hence the process is the same that was explained for detecting objects, with the difference that the ObjectDetector is searching the mark. When the mark has been found ObjectDetector delivers to the MainWindow the position and the region of the mark, and MainWindow send those data to MarkTrack. MarkTrack initialize the tracker for the mark, thereby the mark is tracked. CamShift algorithm is executed in MarkTrack, thus MarkTrack invokes to PositionEvaluator and it is going to verify the actual position of the mark. PositionEvaluator returns a true value when the mark is in a valid position and also it decides the correct moment when the positions must be stogared. In general that is the structure and the logic of the program. The ObjectDetector is always executed when the program is running. This is one of the advantages of the Object Oriented Programming, that the classes only are executed when they are required. And they are used for specific tasks.

In the next page Figure 3-7, the UML model diagram of the program is depicted. It permits to see the classes and their relation between them. In the next section the algorithms designed for each class are explained. Doing more emphasis is the most important processes of this project.

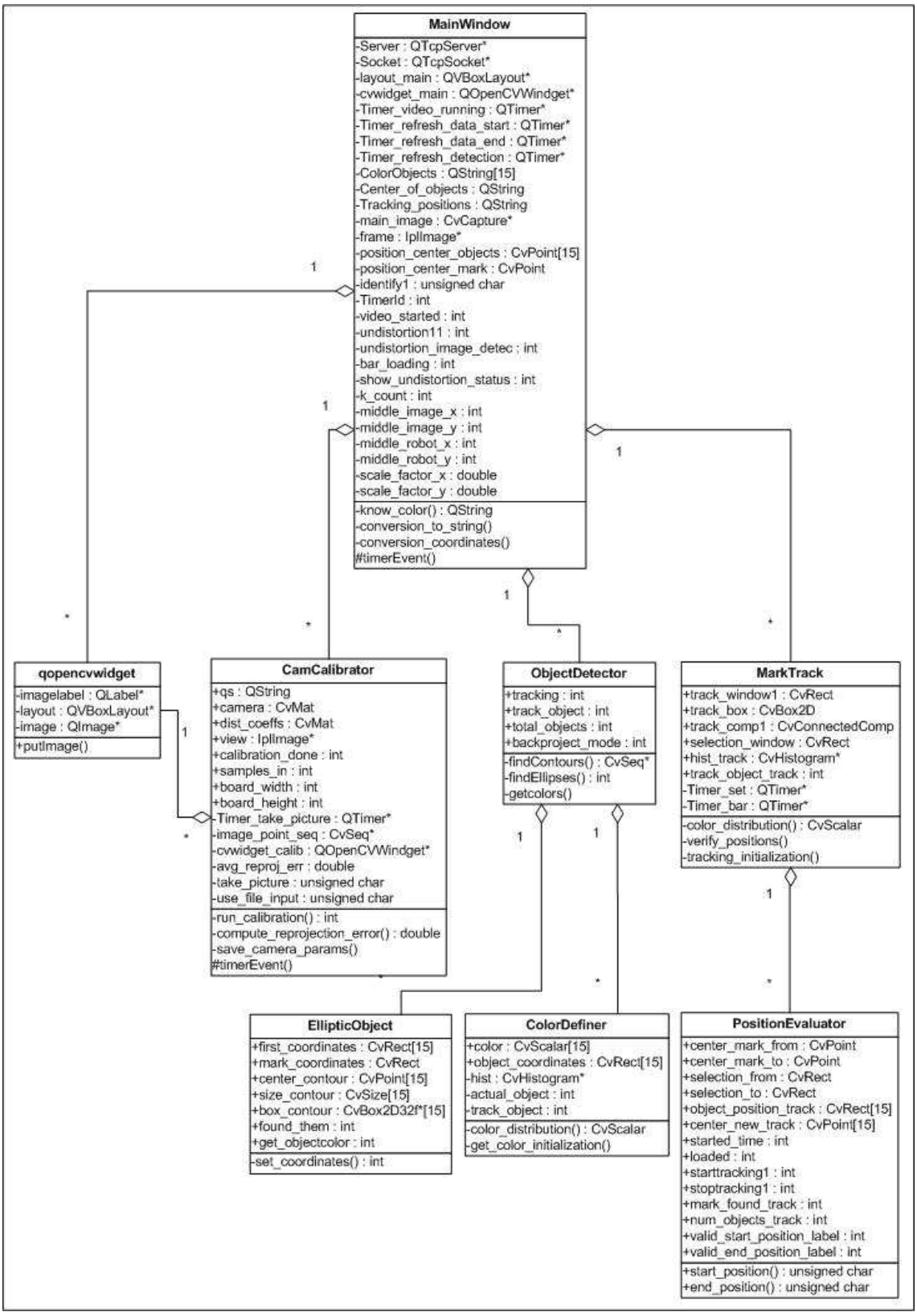


Figure 3-7: UML model diagram of the program

3.3 Camera Calibration Algorithm

The objective of camera calibration is to determine the interior geometry quantities of a camera used for image acquisition. Principally a camera calibration is required in order to achieve accuracy and precision when the application involves quantitative measurements. The calibration procedure essentially defines a mapping between points in the world coordinate system and their corresponding point locations in the image plane. In other words, it is the transformation from three-dimensional word to two-dimensional world, and vice versa. The optical characteristics can be known, they are the intrinsic or internal parameters. These include the effective focal length, which is the distance between the camera lens and the image plane, the location of the image center in pixel coordinates, the effective pixel size, and distortion coefficients of the lens [6].

Also the extrinsic or external parameters, which corresponding to the 3D position and orientation of the camera frame relative to a certain world coordinate system, and camera. These values are represented by the rotation matrix and translation vector.
Therefore a camera is considered calibrated if the principal distance, principal point offset and lens distortion parameters are known. In many applications, especially in computer vision, only the focal length is recovered while for precise photogrammetric measurements all the calibration parameters are generally employed.

OpenCV library contains a function called “cvCalibrateCamera2()”, which is specially developed for calculating the parameters of the internal camera geometry. OpenCV also provides data structures for Camera Calibration and associated functions. The function above mentioned is able to calculate both the intrinsic and extrinsic parameters. This function requires certain information, which is given by other functions. The method of calibration used by “cvCalibrateCamera2()” is to target the camera on a known structure that has many individual and identifiable points. The procedure consist of by viewing that structure from a variety angles, it is possible to achieve the relative localization and orientation of the camera at the time of each image, these values corresponding to extrinsic parameters. And also with the same procedure is possible to know the intrinsic parameters. In order to do this task a calibration pattern or calibration grid is required. The pattern consists of black and white squares on white background. For OpenCV library the pattern is known as “chessboard”. The geometry of the pattern must be known.

Regarding the general description about the camera calibration, in this section is presented the algorithm proposed for doing the calibration stage. This algorithm is designed according with the “cvCalibratedCamera2” function. Pictures are needed for calibration, thus the proposed algorithm can be executed by definition of the number of samples that want to be used, or take the pictures from a file with the pictures

already defined and saved. Figure 3-8 depicts the algorithm proposed for camera calibration. It is shown in the next page.

The calibration method used for “cvCalibrateCamera2” from OpenCV library is based on Zhang’s calibration [5]. Therefore the method recommends that the planar checkerboard grid above mentioned must be placed at different orientations, and the number of samples should be more than 2, and they should be in front of the camera [5]. The developed algorithm from OpenCV library uses the extracted corner points of the checkerboard pattern to compute a projective transformation between the image points of the *n* different images, up to a scale factor. Afterwards, the camera interior and exterior parameters are recovered using a closed-form solution, while the third- and fifth-order radial distortion terms are recovered within a linear least-squares solution. A final nonlinear minimization of the reprojection error, it is solved using a Levenberg-Marquardt method, refines all the recovered parameters [10].

The theory behind of these methods is explained in the next sections, also the camera model used for the main function from OpenCV library. The results obtained are applied and shown in Chapter 5, principally the results when the distortion coefficients are applied, and their effect that they produce in the detection algorithm related with the accuracy.

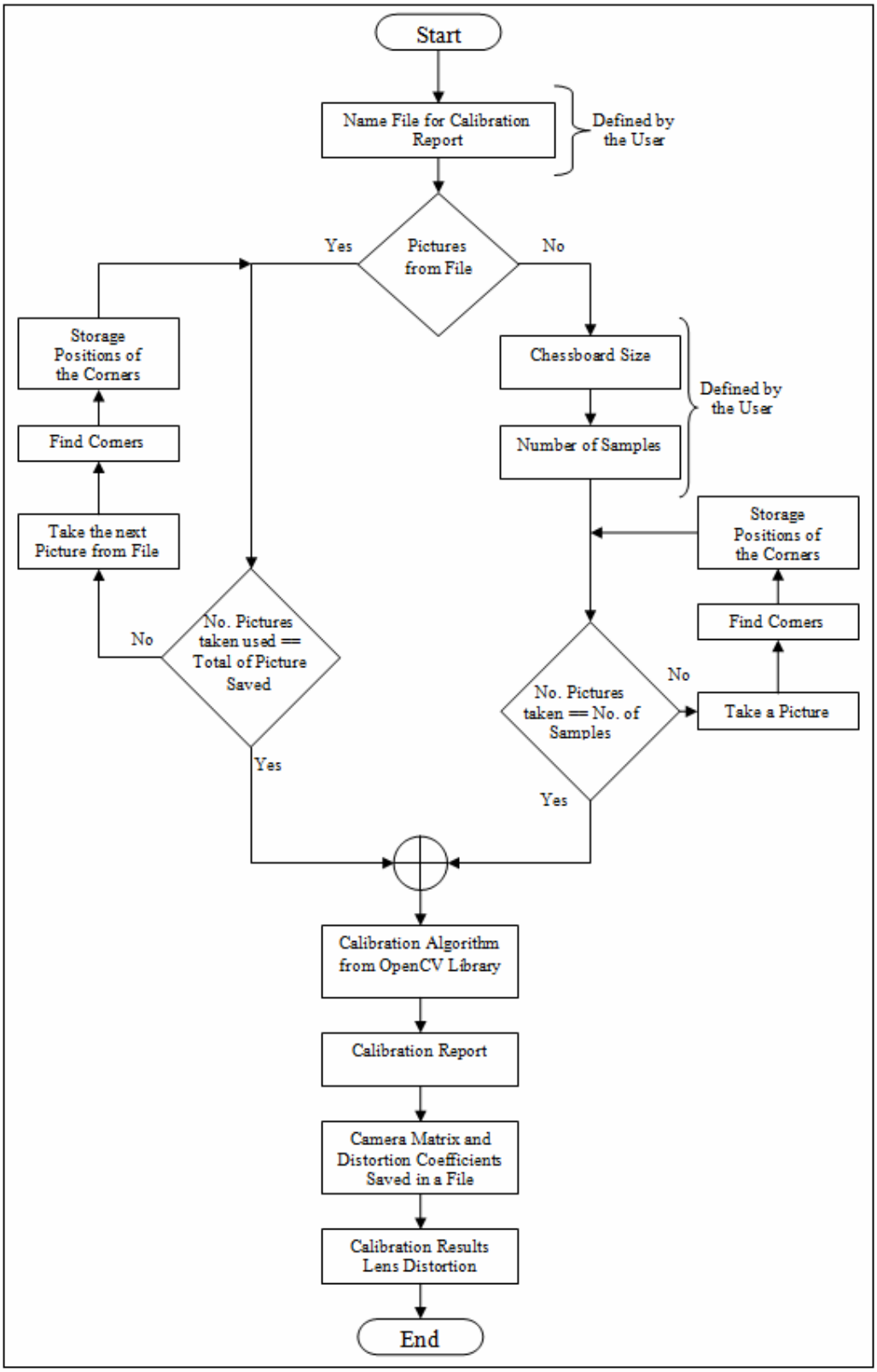


Figure 3-8: Flowchart Camera Calibration Algorithm

3.3.1 Camera Model

In certain instances use of a complex camera model is obviously preferable, for example, if the camera exhibits significant distortions, or if the task requires extremely high precision. However, in many situations a simple camera model is quite sufficient. A camera is a mapping between the 3D world (object space) and a 2D image. Camera models are matrices with particular properties that represent the camera mapping. This is important because scenes are not only three-dimensional. They are also physical spaces with physical units. Hence, the relation between the natural units of the camera (pixels) and the units of the physical word (e.g. meters), it is a critical component in any attempt to reconstruct a three-dimensional scene.

The cameras modeling central projection are specializations of the general projective camera. The anatomy of this most general camera model is examined using the tools of projective geometry. The geometric entities of the camera, such as the projection centre and image plane, can be computed quite simply from its matrix representation [3].

The *pinhole model* is the simplest approximation, which is suitable for many computer vision and computer graphics applications. A pinhole camera performs a perspective transformation. This simple model, usually it is modeled by central projection, which consist of a ray of light is envisioned as entering from the scene or a distant object, but only a single ray enters from any particular point. In a physical pinhole camera, this point is then “projected” onto an imaging surface [2]. The ray intersects a specific plane in space chosen as the image plane [6]. The intersection of the ray with the image plane represents the image of the point. The size of the image relative to the distant object is given by a focal length parameter of the camera [2].

Figure 3-9 depicts certain properties of a pinhole camera model. The pinhole camera contains the following properties [6]:

- Image plane
- Optical axis, also known as the principal axis
- Focal point C, also called the optical center or the center of projection

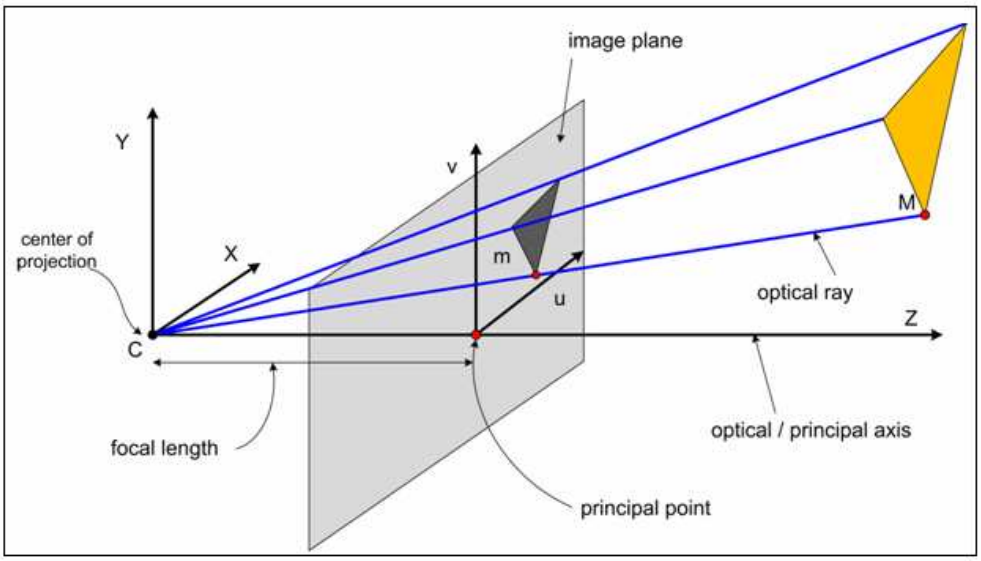


Figure 3-9: The basic Pinhole model [23]

Figure 3-9 also permits to see that the distance from the pinhole aperture to the screen is precisely the focal length, where f is the focal length of the camera, and also it is a parameter of the lens, and Z is the distance from the camera to the object.

For the central projection of points in space onto an image plane, the following consideration is regarded. Let the center of projection be the origin of a Euclidean coordinate system, and consider the image plane at $Z = f$. A point in space with the coordinates $M = (X, Y, Z)^T$ is mapped to the point to the image plane m, where the line joining the point M to the center of projection meets the image plane. In fact, this line is an optical ray. The relationship obtained is by similar triangles. This is illustrated in Figure 3-10 [6].

$$\frac{X}{u} = \frac{Z}{f}, \frac{Y}{v} = \frac{Z}{f} \tag{3.1}$$

Therefore, the projection property is that the point $M = (X, Y, Z)^T$ is mapped to the point $m = (f.\frac{X}{Z}, f.\frac{Y}{Z}, f)^T$ on the image plane. Observing the image of the scene on the image plane, the third dimension is ignored and thus, the following equation is obtained, which describes the central projection mapping from the world to image coordinates.

$$M = \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \mapsto m = \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} f.\frac{X}{Z} \\ f\frac{Y}{Z} \end{pmatrix} \tag{3.2}$$

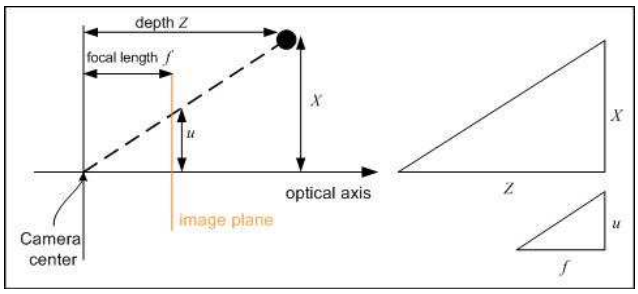


Figure 3-10: Similar triangles of a pinhole camera model [23]

Using the homogeneous coordinates the central projection can be expressed. If the world and image points are represented by homogenous vectors denoted as $\tilde{m}$ and $\tilde{M}$, respectively, thereby central projection is very simply expressed as a linear mapping. Equation 3.2 may be written in terms of matrix multiplication as

$$\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \mapsto \begin{pmatrix} f.X \\ f.Y \\ Z \end{pmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{3.3}$$

The point in space is extended to a vector in four-dimensional space where the fourth component is set to one. Note that this projection does not work if the homogeneous vector $\tilde{M}$ corresponding to 3D point in space M is not mapped to a plane in 4D, where the fourth component is one.

The matrix in Equation 3.3 may be denoted with P and is called the *camera projection matrix*. It is written compactly as:

$$\tilde{m} = P\tilde{M} \tag{3.4}$$

The *camera projection matrix* P contain the camera intrinsic parameters, here only the focal length value f, that are also stored by the *camera calibration matrix* K. The matrix P can be expressed in terms of K as:

$$P = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & f \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} = K\left[I_3 \mid 0\right] \tag{3.5}$$

The matrix $\left[I_3 \mid 0\right]$ represents a matrix divided up into 3 x 3 block which contains the identity matrix and a vector, in this case the three-dimensional vector. Normally the matrix P contains also the external camera parameters. For simplification, in this

case it is assumed that the camera is located at the origin of Euclidean coordinate system with the principal axis of the camera pointing straight down Z-axis [6].

3.3.2 The principal point offset

According with the previous analysis the point where the optical axis meets the image plane is referred to as the principal point. In practice is not possible to assume that the origin of coordinates in the image plane is at the principal point.

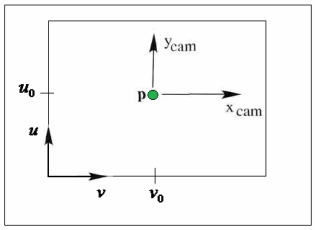


Figure 3-11: Image (u, v) and camera (x_{cam}, y_{cam}) coordinate systems [6]

Given an image of a camera, the origin is most times located at the top left corner, but it is expect that the center of projection is nearly centered in front of the camera sensors. Thus, the principal point is often located somewhere around the center of an image, but note that this need not be so. Hence, a translation is added to each image coordinate according to the coordinates of the principal point $(u_0, v_0)^T$, which is depicted in Figure 3-11.

$$M = \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \mapsto m = \begin{pmatrix} f.\frac{X}{Z} + u_0 \\ f.\frac{Y}{Z} + v_0 \end{pmatrix} \tag{3.6}$$

This formula may be expressed as a linear equation using homogeneous coordinates:

$$\tilde{M} \mapsto \tilde{m} = \begin{pmatrix} f.X + Zu_0 \\ f.Y + Zv_0 \\ Z \end{pmatrix} = \begin{bmatrix} f & 0 & u_0 \\ 0 & f & v_0 \\ 0 & 0 & 1 \end{bmatrix} [I_3 \mid 0] \tilde{M} \tag{3.7}$$

Finally, the camera calibration matrix K is defined as:

$$K = \begin{bmatrix} f & 0 & u_0 \\ 0 & f & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.8}$$

3.3.3 Non-uniform scaling

Normally at the moment when the camera matrix is known two different values for focal length are obtained, their value is very closer to each other, but is not the same. The main reason for this is that the individual pixels on a typical low-cost imager are rectangular rather than square, this because may be the sensors of a camera can have non-squared dimensions. Due to this and additional properties of the electronics of acquisition, the extra effect of non-equal scale factors in both axis direction of the image plane is obtained. Therefore, the camera calibration is rewritten as follows:

$$K = \begin{bmatrix} \alpha & 0 & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.9}$$

Assuming that only there are non-squared sensors with dimensions s_x, s_y, and there is no additional source like sampling errors of digital-analog converts which cause a non-uniform scaling of the image coordinate axis, the values f/s_x and f/s_y for α and β, where f is the focal length of the lens expressed in meters $[m]$ and s_x, s_y is expressed in meters per pixel $[m/pixel]$. Generally, the resulting scale parameters α, β can be interpreted as the size of the focal length in horizontal and vertical pixels, respectively. The behavior of the results is clearly visible at the moment when the conversion from camera coordinates to world coordinates is realized. In Chapter 4 in order to calculate the scale factor, in fact there are two factors, when only one scale factor is expected, hence the two factors must be consider.

3.3.4 Camera Transformation

According with the function provides by OpenCV library, its method of calibration as already mentioned consists of to target the camera on a known structure that has many individual and identifiable points. Viewing this structure from a variety of angles, it is possible to then compute the relative location and orientation of the camera at the time of each image, in other words the extrinsic parameter as well as the intrinsic parameters of the camera. In general points in space are expressed in terms of world coordinates system. The world and camera coordinates system is given in Euclidean space with unit $X-Y-Z$ and unit axes $X'-Y'-Z'$, respectively, Figure 3-12 depicts the relation between two coordinates system.

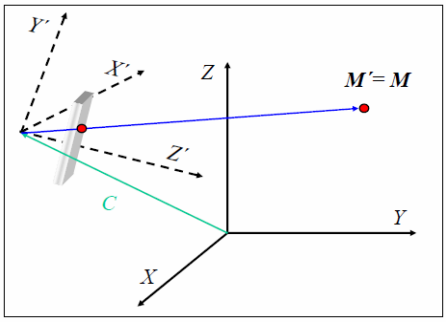


Figure 3-12: The Euclidean transformation between the world and camera coordinate systems [23]

The two coordinate systems are related via a rotation and a translation, using a 3 x 3 rotation matrix and 3D translation vector. A point $M = (X, Y, Z)$ given in world coordinates can be transformed to a point $M' = (X', Y', Z')$ relating to the camera coordinate system as follows.

If R represents the rotation matrix between world coordinate system towards the camera coordinate system, and the camera center is given in world coordinates as a vector C, therefore the relation of a point in world coordinates M respecting to a point in camera coordinates M' is:

$$M' = R(M - C) \tag{3.10}$$

If M is given in homogeneous coordinates, it is written as:

$$\breve{M}' = \begin{bmatrix} R & -RC \\ 0 & 1 \end{bmatrix} \breve{M} \tag{3.11}$$

Assuming that a point $\breve{M} = (X, Y, Z, W)$ given in homogeneous coordinates is projected on a plane with $W = 1$, the general mapping between a point in space $\breve{M}$ and a point $\breve{m}$ on the image of the camera plane given by the pinhole camera is:

$$\breve{m} = P\breve{M} \qquad \text{where } P = KR[I \mid -C] \tag{3.12}$$

for which the calibration matrix K is of the form of Equation 3.9. The rotation and translation do not depend on the intrinsic or internal camera parameters, which stay the same between views, and they are referred to as the extrinsic or external camera parameters. Each, the orientation as well as the translation, has 3 degrees of freedom. In fact, a camera of the form of Equation 3.12, hence it is called a finite projective camera and has 11 degrees of freedom.

Also the general equation may be written as:

$$P = K[R \mid t] \tag{3.13}$$

instead of $P = KR[I \mid -C]$ where $t = -RC$. This is because, it is often not do make the camera center explicit, hence this is used whenever the camera center in modeled not explicitely.

3.3.4.1 Calibration Pattern

The theory about the camera calibration is known, thereby according with the algorithm proposed, the calibration can be executed using the OpenCV library method, which is a flat grid of alternating black and white squares that it is usually known as “chessboard”.

In principle, any appropriately characterized object could be used as a calibration object, but flat chessboard patterns are much easier to deal for this purpose, especially when the calibration methods rely on three-dimensional objects, and generally, it is difficult to make precise 3D calibration objects. Hence OpenCV opts for using multiple views of planar object, in this case a chessboard, rather than one view of a specially constructed 3D object.

The pattern used of alternating black and white squares, which ensures that there is no bias toward one side or the other in measurement. Given an image of a chessboard, which can present the scenario with a person holding a chessboard, or any other scene with a chessboard and a reasonably uncluttered background, an example is depicted in Figure 3-13, thereby, it is possible to use the function from OpenCV library called “cvFindChessboardCorners()” to locate the corners of the chessboard.

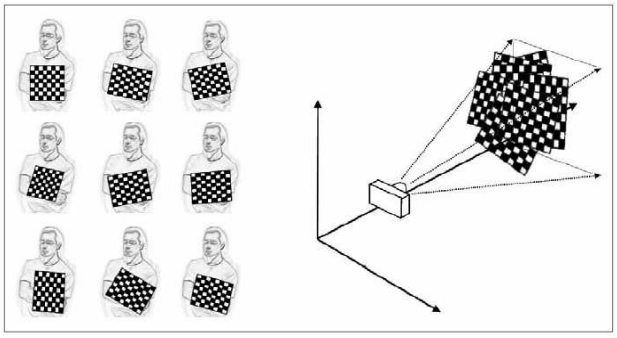


Figure 3-13: Images of a chessboard being held at various orientations (left) provide enough information to completely solve for the locations of those images in world coordinates (relative to the camera) and the camera intrinsic parameters [2]

The “cvFindChessBoardCorners()” function need certain information. Concerning with the most important data for this function, firstly it takes as arguments a single image containing a chessboard, which must be an 8-bit grayscale. That means, after the moment when the picture, a gray scale is applied.

After the gray scale is applied, the next is the geometry of the chessboard, which must be defined. The geometry is according with the quantity of the corners in each row and column of the board. This quantity is the number of interior corners. OpenCv book [2] recommends that in practice is more convenient to use an asymmetric chessboard grid and even and odd dimensions, for example (5 x 6). It is clear visible in Figure 3-14. The reason is that using even-odd asymmetry yields a chessboard that has only one symmetry axis. Thereby the board orientation can always be defined uniquely.

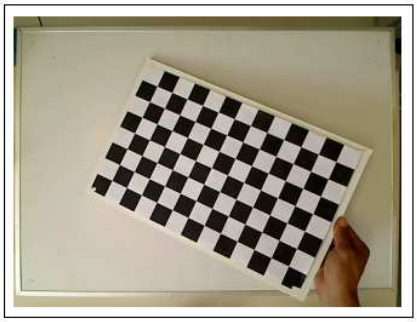


Figure 3-14: Size of Chessboard defined as: width=7, height=12

These arguments are the most important in order to execute successfully the camera calibration. The function has an argument which is a pointer to an array, which contains all the locations of each found corner in the image, this array must be larger enough for storing all of them. The values are expressed in pixels coordinates. If the function is successful at finding all of the corners, that means, the requirement is slightly stricter, because not only must all the corners be found, they must also be ordered into rows and columns as expected. Therefore, only if the corners are found and ordered correctly the returned value of the function is nonzero [2]. In contrast, if the function fails, the returned value is equal to zero. OpenCV reference manual gives detail information [1].

Concerning with the algorithm proposed, when the positions of the corners are known the calibration function can be executed, but the problem is that the corners returned by “cvFindChessBoardCornes()” are only approximate. Hence OpenCV contains a separate function which is able to compute the exact locations of the corners to subpixel accuracy, this is achieve using the previous locations given by “cvFindChessBoardCorners()”. The function is called “cvFindCornerSubPix()”. This function needs an image also with a gray scale applied, and also previous locations

of the corners, these locations are expressed in integer pixels, which are taken as the initial guesses for the corner locations. Also the number of total corners found must be defined. Thereby the function knows how many points there are to compute.

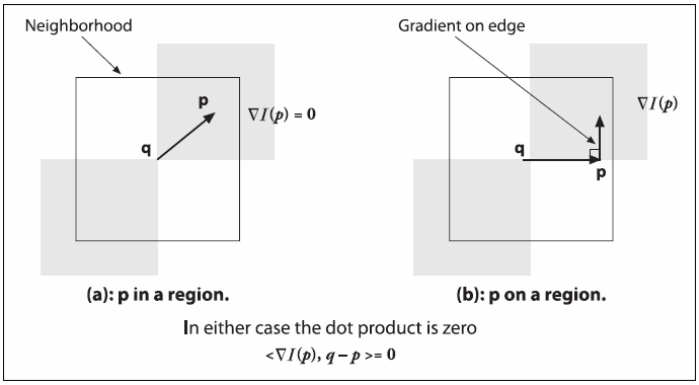


Figure 3-15 : Finding corners to pixel accuracy. (a) The image area around the point *p* is uniform and so its gradient is 0. (b) The gradient at the edge is orthogonal to the vector *q-p* along the edge. In either case, the dot product between the gradient at *p* and the vector *q-p* is 0 [2]

The actual computation of the pixel location uses a system of dot-product expressions that all equal 0, this method is illustrated in Figure 3-15, where each equation arises from considering a single pixel in the region around *p*. The method is briefly described in the next paragraph. In Figure 3-15 the assumption of a starting corner location q that is near the actual sub pixel corner location is consider. The vectors are examined starting at point q and ending at *p*. When *p* is in a nearby uniform or “flat” region, the gradient there is 0. In the other example if the vector *q-p* aligns with an edge then the gradient at *p* on that edge is orthogonal to the vector *q-p*. Therefore, in either case, the dot product between the gradient at p and the vector *q-p* is 0. The method consist of to assemble many such pairs of the gradient at nearby point *p* and the associated vector *q-p*, set their dot product to 0, and solve this assemble as a system of equations. The solution obtained yield a more accurate subpixel location for *q*, the exact location of the corner.

The function “cvFindCornerSubPix()” also has a field called “win”, which specifies the size of the window from which these equations will be generated. These equations form a linear system. It can be solved by inversion of a single autocorrection matrix. In practice, this matrix is not always invertible due to small eigenvalues obtained from the pixels very close to *p*. It is common to simply reject from consideration those pixels in the immediate neighborhood of *p.* Once a new location is found for *q*, the algorithm will iterate using the value found as a starting point and this is realized until the terminations criterion is reached. OpenCV reference manual and OpenCV book give more details about the valid values used for criteria and also about the values that the function need [1] [2]. The process is achieved for every image, this image

can be taken at the moment when the calibration algorithm is executed, or also the images can be taken from a file already saved and defined. Neglecting to call subpixel refinement after the first definition of locations the corners can cause substantial errors in calibration. An important tool that OpenCV offers from calibration functions is called “cvDrawChessboardCorners()”. As its name indicates, this function draws the corners found. This is helpful usually for the image that is used in the first place. Also this function permits to see whether the projected corners match up with the observed corners. In the case if not all of the corners were found the available corners are represented as small red circles. Only if the functions for finding corners have found the corner s this function is able to draw them, which are painted into different colors (one different color is assigned to each row) and connected by lines representing the identified corner order. Figure 3-16 shows the result obtained when the entire corners have been found and it indicates also that is possible to achieve the camera calibration.

Figure 3-16: Result of “cvDrawChessboardCorners()”

3.3.4.2 Homography

Calibration objects supply 2D plane coordinates with regard to an arbitrarily oriented calibration grid in multiple image frames. When a 2D calibration grid is used multiple images of a moving grid are captured. In each frame, grid points on the calibration pattern are related to projected grid points on the image of camera plane through a homography H. In computer vision, homography is defined as a projective mapping from one plane to another. Thus, the mapping of points on a two-dimensional planar surface to the imager of one camera is an example of planar homography. The homographies provide enough constraints to extract camera specific parameters. Considering a calibration grid, in this case the “chessboard”, all reference points on the grid are coplanar, and without loss of generality the assumption that the model

plane is on $Z = 0$ of the world coordinate system is regarded. Thereby, if the columns of P are denoted as p_i the image of a point on the plane with $Z = 0$ is given by [6]:

$$\breve{m} = \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{bmatrix} p_1 & p_2 & p_3 & p_4 \end{bmatrix} \begin{pmatrix} X \\ Y \\ 0 \\ 1 \end{pmatrix} = \begin{bmatrix} p_1 & p_2 & p_4 \end{bmatrix} \begin{pmatrix} X \\ Y \\ 1 \end{pmatrix} \tag{3.14}$$

If it does not model the camera center explicitly, the next substitution is possible $P = [R \mid t]$ in Equation 3.14, and then, it is expressed as [6]:

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = K \begin{bmatrix} r_1 & r_2 & t \end{bmatrix} \begin{pmatrix} X \\ Y \\ 1 \end{pmatrix} \tag{3.15}$$

Where r_i denotes the i^{th} column of the rotation matrix R. The product $K\begin{bmatrix} r_1 & r_2 & t \end{bmatrix}$ can be expressed as a 3 x 3 homography matrix H defined up to a scalar factor. Therefore, if an image of the model plane (calibration pattern) is given, a homography can be estimated as follows. In the following, slightly changing the notation, M was used to denote a point on the model plane, but the coordinates are $M = [X, Y]^T$ since $Z = 0$ and $\breve{M} = [X, Y, 1]^T$. Equation 3.15 may be rewritten as [6]:

$$\breve{m} = H\breve{M} \qquad \text{where } H = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \tag{3.16}$$

The 3 x 3 matrix H is the homography between points located on the calibration pattern and the corresponding points on the image plane. The left and right side of Equation 3.16 is only equal up to arbitrary scale and provides two equations [6].

$$\begin{aligned} u(h_{31}X + h_{32}Y + h_{33}) &= h_{11}X + h_{12}Y + h_{13} \\ v(h_{31}X + h_{32}Y + h_{33}) &= h_{21}X + h_{22}Y + h_{23} \end{aligned} \tag{3.17}$$

Thus, the following linear transformation is obtained:

$$\begin{bmatrix} X & Y & 1 & 0 & 0 & 0 & -uX & -uY & -u \\ 0 & 0 & 0 & X & Y & 1 & -vX & -vX & -v \end{bmatrix} h = 0 \tag{3.18}$$

where p is 9D vector made of the entries of H and has 8 degrees of freedom [6]

$$h = (h_{11}, h_{12}, h_{13}, h_{21}, h_{22}, h_{23}, h_{31}, h_{32}, h_{33})^T$$

When n points correspondences $m \leftrightarrow M$ are given, n above equations exist, which can be collected in $2n$ x 9 matrix L. In fact, the equation $L.h = 0$ must be solved, where the solution is well known to be the right singular vector of L associated with the smallest singular vector. Because some of the elements of L are constant, some of them are in pixels, some of them are in world coordinates, and some of them are multiplications of both, the matrix L is poorly conditioned numerically.

OpenCV library provides a handy function, “cvFindHomography()”, which takes a list of correspondences and returns the homography matrix, which describes those correspondences. Four points as minimum are need for solving, but they can be even many more, if the quantity of the points is bigger it is beneficial, because invariably there is noise and other inconsistencies whose effect is tried to minimize. The correct entries for this function and also for certain applications of it, the OpenCV reference manual gives more details [1]. Concerning with the calibration algorithm proposed depicted in the flowchart corresponding with the next step after the corners have been found, their locations are stored. Also this depends of the number of samples.

As already mentioned OpenCV library recommends more than two views for calibrating using a 3 x 3 chessboard, but consideration for noise and numerical stability is typically what requires the collection of more images of a larger chessboard. In practice, for high-quality results, it is recommended at least ten views using 7 x 8 or even larger chessboard. An important point is if the chessboard is moved enough between images the set of views obtained is much [2]. For the proposed calibration algorithm the number of samples is equal to 50 different views and the chessboard used is width=7, and height=12.OpenCV library has the calibration algorithm already developed. It is based on theory already explained. The most important part of the whole process is the function called “cvCalibrateCamera2()”, its structure is shown in Figure 3-17.

```
void cvCalibrateCamera2(
  CvMat*    object_points,
  CvMat*    image_points,
  int*      point_counts,
  CvSize    image_size,
  CvMat*    intrinsic_matrix,
  CvMat*    distortion_coeffs,
  CvMat*    rotation_vectors    = NULL,
  CvMat*    translation_vectors = NULL,
  int       flags               = 0
);
```

Figure 3-17: Calibration function structure [2]

This function returns the camera calibration results, these are: *camera intrinsic matrix,* the *distortion coefficients,* the *rotation vectors,* and the *translation vectors.*

The first two of these constitute the intrinsic parameters of the camera, and the latter two are the extrinsic parameters corresponding with the positions of the objects, in this case the chessboard, and also their orientation. The distortion coefficients are the coefficients from the radial and tangential distortion equations. These coefficients are applied in Chapter 4, but they are obtained at the calibration is executed. The camera matrix is one of the most important results, because it allows transforming 3D coordinates to the 2D image coordinates. Also the camera matrix is used to do the reverse operation. OpenCV library contains a function for solving the problem related with the radial and tangential distortion, therefore also for this task the camera matrix are required, but to apply the function it is easy because both values camera matrix and distortion coefficients are known after calibration. Other function also executed in the calibration process is “cvCheckArr()”. The purpose of this function is to check that every array element in neither NaN nor ±∞. Also it is able to check if every element is greater than or equal to minVal and less than maxVal. The function returns nonzero if the check succeeded, and it returns zero otherwise. OpenCV reference manual gives more details [1]. The last function applied in the calibration process is called “cvProjectPoints2()”. This function has different applications. Principally it computes projections of 3D points to the image plane given intrinsic and extrinsic camera parameters. But in this case the function is used because it also computes back-projection error for with intrinsic and extrinsic parameters.

Lastly the camera calibration algorithm delivers a general report with the results from the calibration. Two different files also are created. In one of them the camera matrix is saved, and in the other one the distortion coefficients are saved. When the main program is executed and the initial conditions are setting, the camera matrix and the distortion coefficients vector are created using the values from the files. This is helpful because the User does not need to do a calibration every time that the program is closed and opened again. However, the calibration option is in case if the camera is changed for another one, or the user wants to do a new calibration, but the values given by the calibration algorithm proposed are very good.

The general report shows the details of the calibration results. It also shows particular information as date, even the time when the calibration was executed. Figure 3-18 shows the report delivered. In this case it is a report corresponding to the values given by the pictures saved in the file. They are 50 different pictures. The resolution of the image is also contained in the report and the chessboard size as well. The camera matrix, distortion coefficients and projection error are shown. Also it contains a list of the error values for each view or picture.

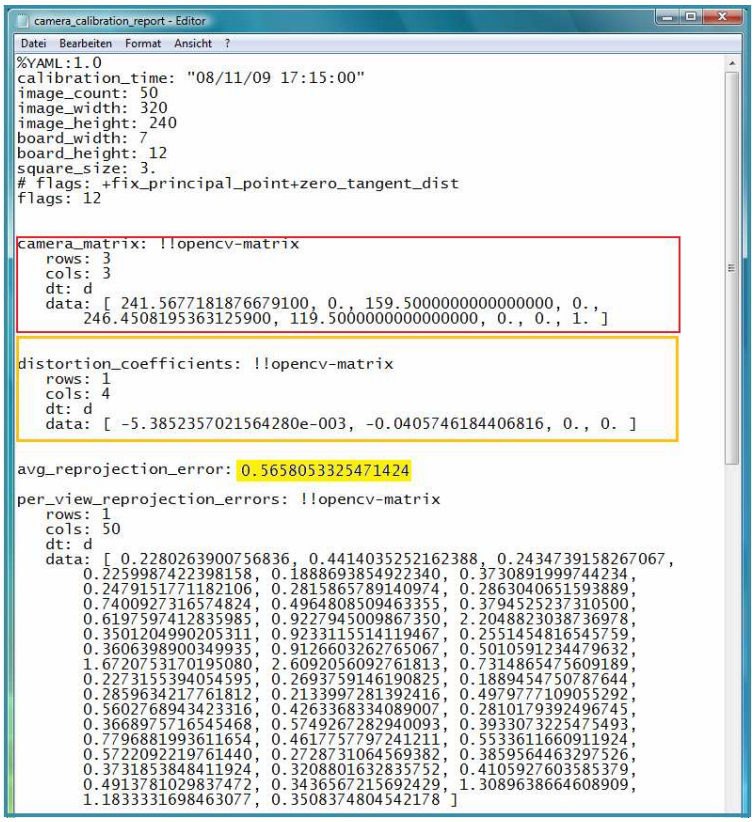


Figure 3-18: Camera Calibration Report delivered by Calibration Algorithm. Part 1

3.4 Object Detection Algorithm

The Vision Systems is the final integration of different subsystems. One subsystem commonly develops a certain task. In this case, the main task is the detection of objects in order to obtain information about them. The objects are already predefined, and all of them have the same characteristics in terms of size and shape.

To attain a good solution for this application firstly it is necessary to consider that the camera is fixed. Therefore the camera view is limited, and just the top of each object is viewed. There are many different options and functions contained in OpenCV library that can be used for creating an object detection algorithm. For the current objective as already mentioned, the objects characteristics permit to develop a method, which is able to be implemented for the objects and also for the tracking mark. The mark used for the tracking is defined as an object. The mark also has the same shape but smaller size. Thereby the algorithm proposed for object detection is applying an image processing, which is able to extract and analyze all the contours located in the image. The found contours with the sufficient characteristics in order to define them as ellipses are detected as valid objects. The algorithm proposed presented for object detection is regarding the previous description. Figure 3-19 shows the flowchart of the detection algorithm.

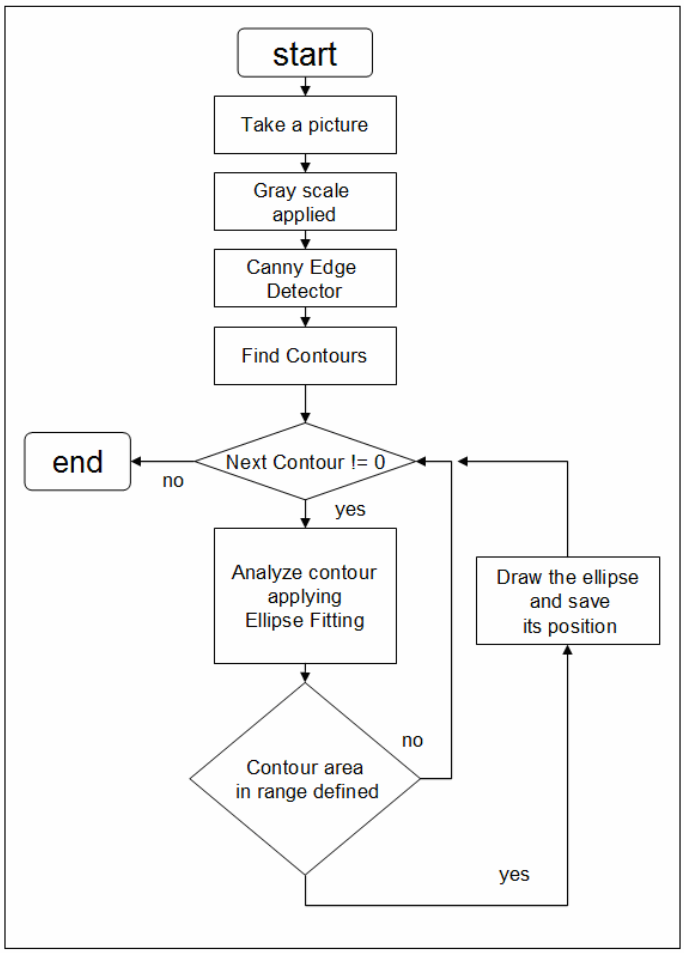


Figure 3-19: Flowchart for detecting objects

3.4.1 Image Processing

Image processing is a basic principle on a Vision System works. When a video is shown a movement can be watched in the screen. The movement is due to many pictures are taken at shorts time intervals. At those intervals the image is processed, and then it is shown. An image processing is required in order to do certain tasks, for instance if a particular object is searched, or there is a special interest for knowing information about the environment. It could also involve the actual orientation view, the distance between the camera view and the object plane, and perhaps the size of the objects wants to be known, and so forth. All this information can be extracted from the picture, thus applying different algorithms the interest data can be known.

OpenCV library contains different functions, only a few of the most important of them are explained below, they are related with the detection proposed. OpenCV library provides functions, which permit to process images, a few of the most important used for developing the detection algorithm are described here.

3.4.1.1 Image Functions

OpenCV library represents images in the format IplImage. That comes from Intel® Image Processing Library (IPL). IPL reference manual gives detailed information about the format [1]. The IplImage structure has many different fields, Figure 3-20 shows them. OpenCV reference manual gives detail information about the fields [1].

IplImage Structure Definition

```
typedef struct _IplImage {
    int nSize; /* size of iplImage struct */
    int ID; /* image header version */
    int nChannels;
    int alphaChannel;
    int depth; /* pixel depth in bits */
    char colorModel[4];
    char channelSeq[4];
    int dataOrder;
    int origin;
    int align; /* 4- or 8-byte align */
    int width;
    int height;
    struct _IplROI *roi; /* pointer to ROI if any */
    struct _IplImage *maskROI; /*pointer to mask ROI if any */
    void *imageId; /* use of the application */
   struct _IplTileInfo *tileInfo; /* contains information on tiling
*/
    int imageSize; /* useful size in bytes */
    char *imageData; /* pointer to aligned image */
    int widthStep; /* size of aligned line in bytes */
    int BorderMode[4]; /* the top, bottom, left,
    and right border mode */
    int BorderConst[4]; /* constants for the top, bottom,
                           left, and right border */
    char *imageDataOrigin; /* ptr to full, nonaligned image */
```

Figure 3-20: Image structure definition.[1]

All possible values of the field *depth* listed in *ipl.h* header file include:

IPL_DEPTH_8U - unsigned 8-bit integer value (unsigned *char*),
IPL_DEPTH_8S - signed 8-bit integer value (signed *char* or simply *char*),
IPL_DEPTH_16S - signed 16-bit integer value (short *int*),
IPL_DEPTH_32S - signed 32-bit integer value (*int*),
IPL_DEPTH_32F - 32-bit floating-point single-precision value (*float*) [1].

In the above list the corresponding types in C are placed in parentheses. One important parameter is *nChannels,* it means the number of color planes in the image. Grayscale images contain a single channel, while color images usually include three or four channels [1]. The parameter origin indicates, whether the top image row (*origin == IPL_ORIGIN_TL*) or bottom image row (*origin == IPL_ORIGIN_BL*) goes first in memory. Windows bitmaps are usually bottom-origin, while in most of other environments images are top-origin [1].

3.4.1.2 “Region of Interest” IplRIO Structure

At the process of an image, at certain time could be necessary to analyze a particular region, or perhaps to apply a process by selecting some rectangular part in the image, or both. The selected rectangle is called "Region of Interest" or ROI. The

structure IplImage contains the field RIO or this purpose. The structure IplROI contains parameters of selected ROI. In Figure 3-21 the structure of IplRIO is shown

IplROI Structure Definition

```
typedef struct _IplROI {
        int  coi;     /* channel of interest or COI */
        int  xOffset;
        int  yOffset;
        int  width;
        int  height;
    } IplROI;
```

Figure 3-21: IplRIO structure [1].

IplROI includes ROI origin and size as well. The field COI (“Channel of Interest”), if it is equal to 0, it means that all the image channels are selected. Otherwise it specifies an index of the selected image plane **[1]**.

3.4.1.3 cvCvtColor Function

An important function also contained in OpenCV library is “cvCvtColor()”, which is required for many applications, principally when an image processing is done. It is able to convert from one color space (number of channels) to another while expecting the data type to be the same. The exact conversion operation to be done is specified by the argument code, which has possible values. OpenCV book gives details about the possible values [2]. Figure 3-22 shows the function structure.

```
void cvCvtColor(
    const CvArr* src,
    CvArr*       dst,
    int          code
);
```

Figure 3-22 cvCvtColor function structure

The algorithm proposed for detecting objects, the step after the picture is taken a gray scale is applied. It is needed because of the Threshold function and Canny Edge Detection. For example, Figure 3-23 shows a picture with valid objects, and all the steps from the algorithm proposed are shown in order to explain the whole procedure for detecting.

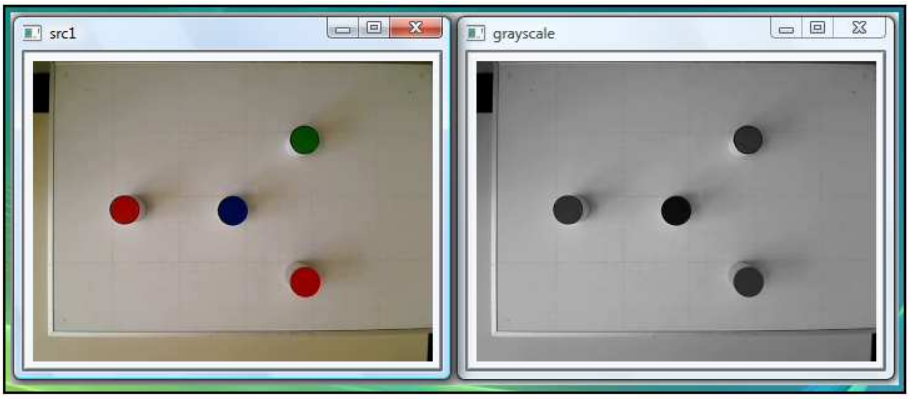


Figure 3-23: Image before and after the gray scale is applied

3.4.1.4 IplImage Limitations

OpenCV has several limitations in support of IplImage

- Each function supports only a few certain depths and/or number of channels.
  For example, image statistics functions support only single-channel or three-channel images of the depth *IPL_DEPTH_8U*, *IPL_DEPTH_8S* or *IPL_DEPTH_32F*.
  The exact information about supported image formats is usually contained in the description of parameters. Manual reference gives more details [1].
- The fields *colorModel*, *channelSeq*, *BorderMode*, and *BorderConst* are ignored.
- The field *align* is ignored and *widthStep* is simply used instead of recalculating it using the fields *width* and *align*.
- The fields *mask ROI* and *tileInfo* must be zero.
- COI support is very limited. Now only image statistics functions accept non-zero COI values.
- ROIs of all the input/output images have to match exactly one another.

In the above list some limitations are described, OpenCV manual reference gives more details [1]. OpenCV supports most of the commonly used image formats that can be supported by *IplImage*, it is helpful for testing [1].

3.4.2 Contour Processing

As already mentioned all the objects have the same shape and the mark as well, one of the easiest way for detecting them it is to know all the data contained in the picture. If the contours are extracted it is possible to recognize the valid objects, and also the mark using for the tracking with the same detection procedure. Therefore a

short introduce about how to get the contour from an image, and what is exactly considered as a contour, both are explained in the next paragraphs.

Contour detection in real images is a fundamental problem in many computer vision tasks. Contours are distinguished from edges which are variations in intensity level in a gray level image whereas contours are salient coarse edges that belong to objects and region boundaries in the image. A contour is a list of points that represent a curve in an image, its sense is not important. This representation can be different depending on the circumstance at hand. There are many ways to represent a curve.

Contours are represented in OpenCV by sequences, in which every entry in the sequence encodes information about the location of the next point on the curve.
The type of structure used by OpenCV is "*CvSeq*" sequence, which is a sequence of points. OpenCV reference manual gives detail information [2]. Two methods are used for OpenCV library for representing contours. The first method is called the Freeman method or the chain code. For any pixel all its neighbors with numbers from 0 to 7 can be enumerated as is shown in Figure 3-24 [1].

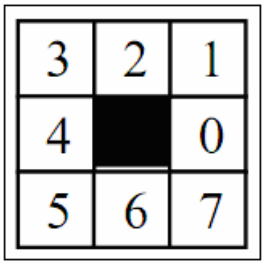


Figure 3-24: Contour Representation in Freeman method [1].

The 0-neighbor denotes the pixel on the right side. As a sequence of 8-connected points, the border can be stored as the coordinates of the initial point, followed by codes, from 0 to 7, that specify the location of the next point relative to the current one. This is shown in Figure 3-25, its contour representation is shown in Figure 3-26 [1].

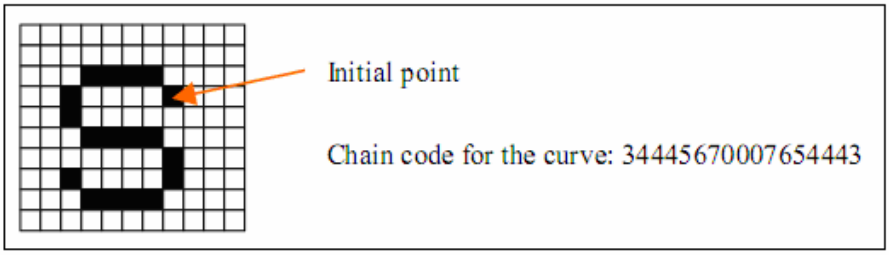


Figure 3-25: Freeman coding of connected components [1].

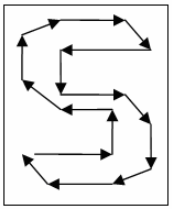


Figure 3-26: Contour representation [1].

The chain code is a compact representation of digital curves and an output format of the contour retrieving algorithms.

The second method is called Polygonal representation, which the curve is coded as a sequence of points, vertices of a polyline. This alternative is often a better choice for manipulating and analyzing contours over the chain codes. However, this representation is rather hard to get directly without much redundancy. Instead, algorithms that approximate the chain codes with polylines could be used [1].

The function “cvFindContours()” computes contours from binary images. It can take images created by “cvCanny()”, which have edge pixels in them, or images created by functions like “cvThreshold()”, in which the edges are implicit as boundaries between positive and negative regions [1]. “cvCanny()” and “cvThreshold()” are the options that can be applied after the moment when image in gray scale is obtained. The objective is to reduce the amount of the data contained in the picture. When Threshold is used, it is possible to apply the threshold values to the original picture, it means without gray scale applied. The gray scale is necessary because of “cvFindContour()” function.

3.4.2.1 Threshold

Threshold functions are used mainly for two purposes:

- Masking out some pixels that do not belong to a certain range, for example, to extract blobs of certain brightness or color from the image
- Converting grayscale image to bi-level or black-and-white image.

Usually, the resultant image is used as a mask or as a source for extracting higher-level topological information, contours, skeletons, lines [1].

Generally, threshold is a determined function $t(x, y)$ on the image: [1]

$$t(x,y) = \begin{cases} A(p(x,y)), f(x,y,p(x,y)) = true \\ B(p(x,y)), f(x,y,p(x,y)) = false \end{cases} \quad (3.19)$$

The predicate function $f(x, y, p(x, y))$ is typically represented as $g(x, y) < p(x, y) < h(x, y)$, where g and h is some functions of pixel value. Commonly they are simply constants [1].

There are two basic types of threshold operations. The first type uses a predicate function, independent from location, $g(x, y)$ and $h(x, y)$, which are constants over the image. The second type of the functions chooses $g(x, y)$ and $h(x, y)$, depending on the pixel neighborhood, in order to extract regions of varying brightness and contrast [1]. The functions, implement both these approaches. They support single-channel images with depth *IPL_DEPTH_8U*, *IPL_DEPTH_8S* or *IPL_DEPTH_32F* and can work in-place [1].

Each threshold type corresponds to a particular comparison operation between the i^{th} source pixel (source image src$i$) and the threshold, it is denoted in Table 3-1 by T. Depending on the relationship between the source pixel and the threshold, the destination pixel (destination image dst$i$) may be set to 0, the src$i$, or the maximum value, it is denoted in the table by M [2].

Threshold type	Operation
CV_THRESH_BINARY	$dst_i = (src_i > T)\,?\,M : 0$
CV_THRESH_BINARY_INV	$dst_i = (src_i > T)\,?\,0 : M$
CV_THRESH_TRUNC	$dst_i = (src_i > T)\,?\,M : src_i$
CV_THRESH_TOZERO_INV	$dst_i = (src_i > T)\,?\,0 : src_i$
CV_THRESH_TOZERO	$dst_i = (src_i > T)\,?\,src_i : 0$

Table 3-1: Threshold type supported by the function from OpenCV library [2].

Figure 3-27 depicts the effect or result produced by each type of threshold. The graphs also are helpful for taking a decision of which is the better option for a particular application.

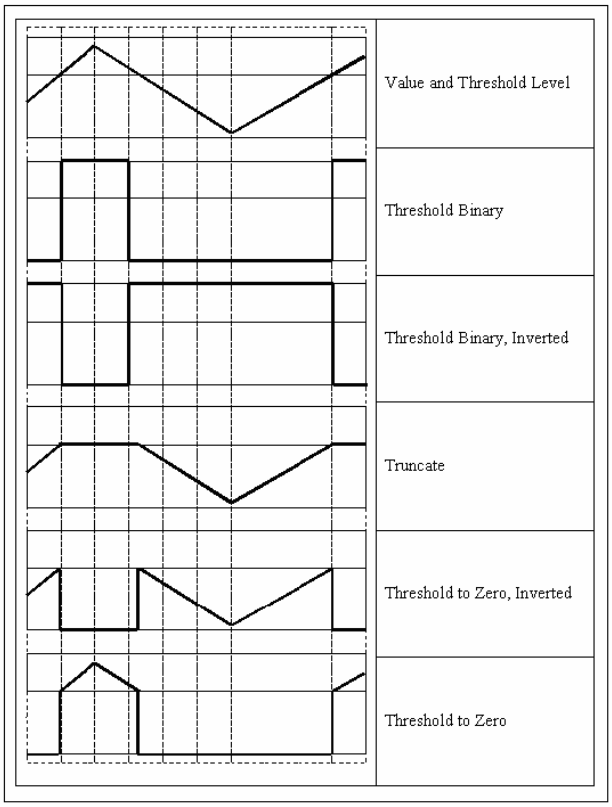


Figure 3-27: Meanings of Threshold types [2].

Related with the detection algorithm proposed, both situations are tested. This is using the original picture and also with a gray scale applied left and right side respectively in Figure 3-28. The objective of this is to analyze what is the different between the obtained results.

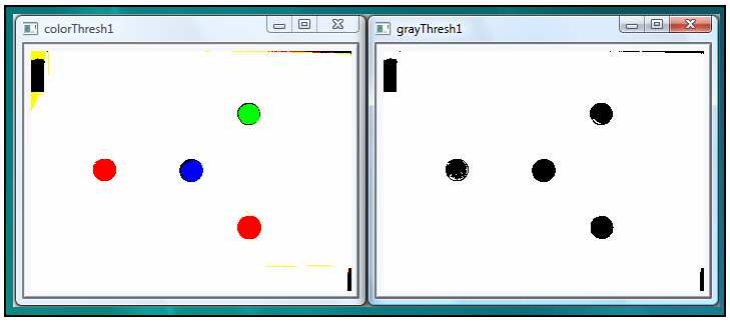


Figure 3-28: Threshold function applied

Both results permit to appreciate that, the amount of the data contained in the picture is drastically reduced. Only the result shown in the right side can be used for finding contours. The contours are found using the functions that OpenCV offers. The image obtained after the threshold process is used directly. Every contour given previously by the threshold is found and drawn. It is shown in Figure 3-29. There are two different colors in the image due to some the contours are regarded as internal or external.

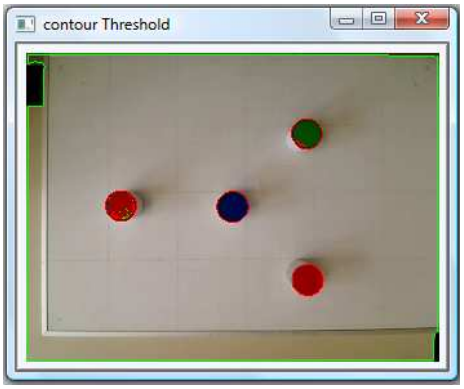


Figure 3-29: Found contours using Threshold result

3.4.2.2 Canny Edge Detection

Another option in order to process the image and extract the contours is Edge detector. The edge detection process serves to simplify the analysis of images, which reduce drastically the amount of data to be processed, while at the same time preserving useful structural information about object boundaries. The Canny algorithm uses an optimal edge detector based on a set of criteria, which include finding the most edges by minimizing the error rate. It is by marking edges as closely as possible to the actual edges to maximize localization and marking edges only once when a single edge exists for minimal response [11].
The basic idea is to detect at the zero-crossings, this from the second directional derivative of the smoothed image, in the direction of the gradient, where the gradient magnitude of the smoothed image being greater than some threshold depending on image statistics [11].

It seeks out zero-crossings of

$$\frac{\partial^2 (G * I)}{\partial n^2} = \frac{\partial \left(\left[\frac{\partial G}{\partial n} \right] * I \right)}{\partial n} \qquad (3.20)$$

Where n is the direction of the gradient of the smoothed image [11].

Canny zero-crossings corresponds to the first directional-derivatives, maxima and minima in the direction of the gradient, maxima in magnitude reasonable choice for locating edges [11]. The filter of Canny derives by optimizing a certain performance index that favors true positive, true negative and accurate localization of detected edges. Analysis is restricted to linear shift invariant filter that detect unblurred 1D continuous step [1]. The OpenCV function takes grayscale image on input, and

returns bi-level image where non-zero pixels mark detected edges. Below the 4-stage algorithm is described [1].

Stage1. Image Smoothing

The image data is smoothed by a Gaussian function, which is with the width specified by the user parameter.

Stage2. Differentiation

The smoothed image, retrieved at Stage1, it is differentiated with respect to the direction x and y. (blanks)

From the computed gradient values x and y, the magnitude and the angle of the gradient can be calculated using the hypotenuse and arctangent functions.

Stage3 Non-Maximum Suppression

After the gradient has been calculated at each point of the image, the edges can be located at the points of local maximum gradient magnitude. It is done via suppression of non-maximums, the points whose gradient magnitudes are not local maximums. However, in this case the non-maximum perpendiculars to edge direction, rather than those in the edge direction, have to be suppressed, since the edge strength is expected to continue along an extended contour. The algorithm starts off by reducing the angle of gradient to one of the four sectors shown in Figure 3-30. The algorithm passes the 3x3 neighborhood across the magnitude array. At each point the center element of the neighborhood is compared with its two neighbors along line of the gradient given by the sector value. If the central value is non-maximum, that is, not greater than the neighbors, it is suppressed.

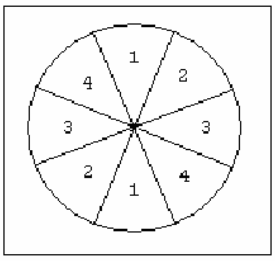


Figure 3-30: Gradient sectors [1]

Stage4. Edge Thresholding

The Canny operator uses the so-called “hysteresis” threshold. The functions for applying threshold use a single threshold limit, which means that, if the edge values fluctuate above and below this value, the line appears broken. This phenomenon is commonly referred to as “streaking”. Hysteresis counters streaking by setting an upper and lower edge value limit. Considering a line segment, if the value lies below

threshold it is immediately rejected. Points which lie between the two limits are accepted if they are connected to pixels which exhibit strong response.
The likelihood of streaking is reduced drastically since the line segment points must fluctuate above the upper limit and below the lower limit for streaking to occur. J. Canny recommends the ratio of high to low limit to be in the range of two or three to one, based on predicted signal-to-noise ratios [1].
Figure 3-31 shows the result when Canny Edge Detection is applied, in comparison with the threshold, the result is much better because of the principle of Canny algorithm which uses two different thresholds. It is helpful at the moment when the contours are searched, because it is possible to see and to analyze every contour located in the picture.

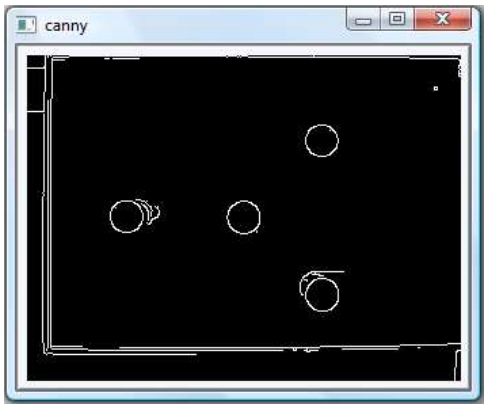


Figure 3-31: Canny Edge Detection result

With the contours found the “cvFindContours()” function can be applied, and also is suitable to drawn every contour located in the image, in order to see what was regarded as a contour. The result in Figure 3-32 shows every contour found in the image. There are two different colors in the image due to some the contours are regarded as internal or external.

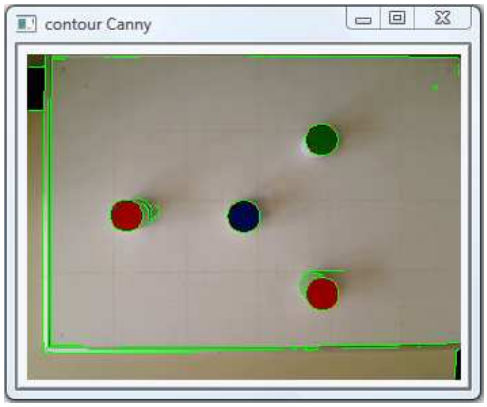


Figure 3-32: Found contours using Canny result

The quantity of the contours found in both cases, Threshold and Canny it is different, Canny algorithm allows to do more detail the contour analysis. Therefore the contours located in the image are found much easier than Threshold analysis. It is shown in the previous section, which only it is considering the “face” of the objects.
However, the two functions can be applied. The algorithm proposed is based on the results obtained from Canny function are better. Until this step every contour has been found, the next step is about the selection of the interest contours.

3.4.3 Geometry Ellipse Fitting

Almost the whole detection algorithm is completed. The selection of the valid contour is required. Considering the geometry of the objects, ellipse fitting is a good solution for the algorithm proposed. The majority of the problems in computer vision are related in one form or another to the problem of estimating parameters from noisy data. A parameter estimation problem is usually formulated as an optimization one.
Fitting of primitive models is one of the best options, principally for detecting objects when their geometry is the same. The results obtained applying this approximation they are in terms of simplification of the data, thereby a benefit of higher level processing is gotten. One of the most commonly used models is the ellipse, which, being a perspective projection of the circle is of great importance for many industrial applications [12].

The representation of general conic by the second order polynomial is as follows

$$F(\vec{a},\vec{x}) = \vec{a}^{T},\vec{x} = ax^2 + bxy + cy^2 + dx + ey + f = 0 \qquad (3.21)$$

with the vector denoted as

$$\vec{a} = [a,b,c,d,e,f]^T \text{ and } \vec{x} = [x^2, xy, y^2, x, y, 1]^T \text{ [12].} \qquad (3.22)$$

$F(\vec{a},\vec{x})$ is called the “algebraic distance between point (x_0, y_0) and conic $F(a,x)$”.
Minimizing the sum of square algebraic distance $\sum_{i=1}^{n} F(\vec{x_0})^2$ may approach the fitting of conic [12]. In order to achieve ellipse-specific fitting polynomial coefficients must be constrained.

For ellipse they must satisfy

$$b^2 - 4ac < 0 \qquad (3.23)$$

Moreover, the equality constraint $4ac - b^2 = 1$can imposed in order to incorporate coefficients scaling into constraint.

This constraint may be written as a matrix

$$\vec{a}^{T} C \vec{a} = 1. \quad (3.24)$$

Finally, the problem could be formulated as minimizing $\left\| D\vec{a} \right\|^2$ with constraint $\vec{a}^{T} C \vec{a} = 1$

where D is the $n * 6$ matrix $\left[\vec{x}_1, \vec{x}_2, ..., \vec{x}_n \right]^T$ [12].

Introducing the Lagrange multiplier results in the system $2D^T D\vec{a} - 2\lambda C\vec{a} = 0$, this can be re-written as follows:

$$\vec{a}^{T} C \vec{a} = 1$$
$$S\vec{a} = 2\lambda C\vec{a}$$
$$\vec{a}^{T} C \vec{a} = 1,$$

After the system is solved, ellipse center and axis can be extracted [12].

Conic fitting is used because: [12]

- It is one of the simplest problems in computer vision on one hand
- It is, on the other hand, a relatively difficult problem because of it is nonlinear nature.

The objects are circles, and in terms of size all of them have the same, regarding this characteristics, a good option for detecting them is determine the quantity of circles in the image. Thereby it is possible to know how many circles contain the image, after that, certain conditions must be applied, and decide if the circles correspond to valid objects, for instance the area and perimeter could be checked.

OpenCV library has a function called “cvFitEllipse()”, which was already described. This function returns the ellipse, which is the best approximation to the contour. This means, that not all points in the contour will be enclosed in the ellipse returned [2], as is shown in Figure 3-33. Thus, applying this function is possible to achieve an analysis for every contour found, and decide if the current contour is an ellipse or not. The functions called “cvContourArea()” and “cvArcLength()” are helpful, because they allow to know the area and the perimeter of the contour. Therefore a range of area is previously defined and the perimeter as well, if the current contour has all the characteristics described, it can be considered as a valid object.

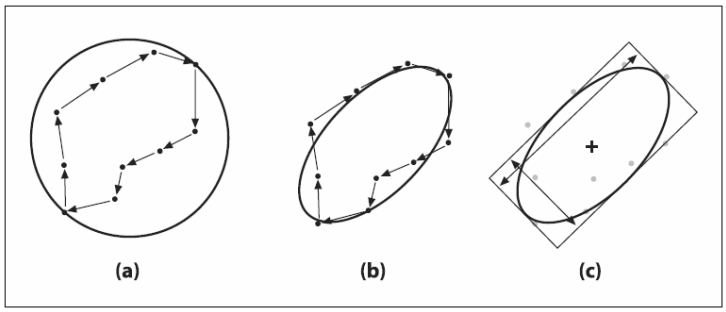


Figure 3-33: Ten point contours with the minimal enclosing circle super imposed (a), and with the best fitting ellipsoid (b); a box (c) is used by OpenCV to represent that ellipsoid [2].

The function “cvFitEllipse()” needs a minimum quantity of points, it must be equal to six, when the points are not enough the next contour is taken, and the current one is discarded. The function returns a type variable “cvBox2D32f”, which contain the necessary data for building an ellipse these data can be appreciated in Figure 3-34. Using those data is possible to define the new type of variables for extracting the center of the ellipse, and also the width and height respectively. The variables created are “CvPoint” and “CvSize”, both variables support integer values, and “cvBox2D32f” contain float values, when the values are copied a “cvRound()” function is needed. An important point is that, the value contained for width and height in the “cvBox2D32f” corresponds to the total amount for both values.The function for drawing the ellipse just need the half of those values, therefore a division is needed in order to only assignment the half of the total amount. To draw the ellipses for every object is because of the user interface, it must show in the screen where the objects are.

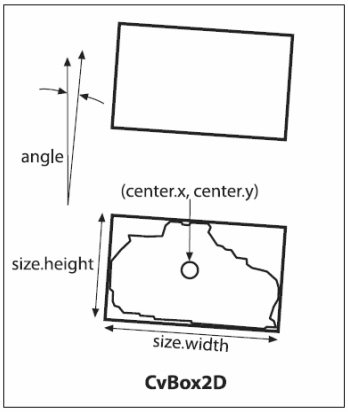


Figure 3-34: CvBox2D can handle rectangles of any inclination [2].

Figure 3-35 depicts the final structure required for defining a contour as an ellipse, and the variables previously mentioned are used for drawing the ellipse using “cvEllipse()” function. Hence these data are extracted, and also the orientation of the

ellipse, the angle also is necessary for drawing, but it can be obtained from the original variable, the structure field is called “box->angle”.

The value of the area of all objects depends of the total distance between the camera view and the object plane, and then, if the value of the area is in the range, the current contour is a valid object. When this occurs the center position and the dimensions (size) must be storage.

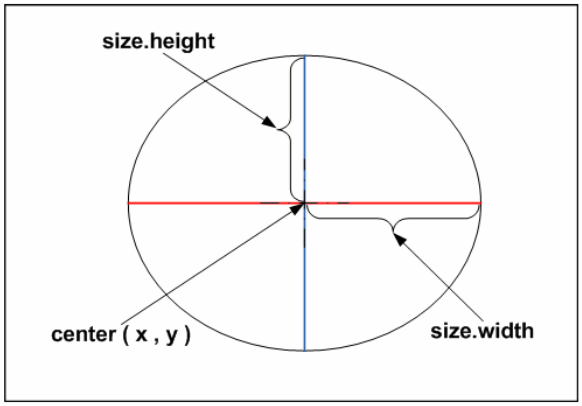


Figure 3-35: Ellipse structure defined by OpenCV functions

The final step of the algorithm is to define a region around the ellipse. The variable returned by the “cvFitEllipse()” function is not the complete position, it is because one point is missing. Therefore is suitable to define it because of the tracking system. Firstly two points are created from the values given by the “”cvFitEllipse()” function. These points can be used for drawing a rectangle around the object, but in this case are calculated because a third point is needed.

The values are gotten from the variables: center, size and box from the ellipse as follows:

Pt1.x is obtained from center.x position minus the half of the total width of the ellipse. This value is taken from “box->size.width”.

Pt1.y is obtained from center.y position plus the half of the total height of the ellipse. This value is taken from “box->size.height”.

Pt2.x is the result of the addition the center.x position and the “box->size.height” variable.

Pt2.y is the result of the subtraction the center.y position minus the “box->size.width” variable.

The previous operations allow knowing the points located as it is shown in Figure 3-36.

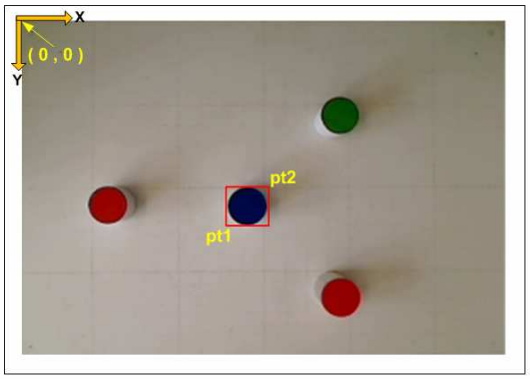


Figure 3-36: Rectangle drawn taken as reference pt1 and pt2

OpenCV Library has a type of variables called “CvRect”, it contains not only fields as x and y positions, also the width and height can be known. A clear example of this type of structure is depicted in Figure 3-37.

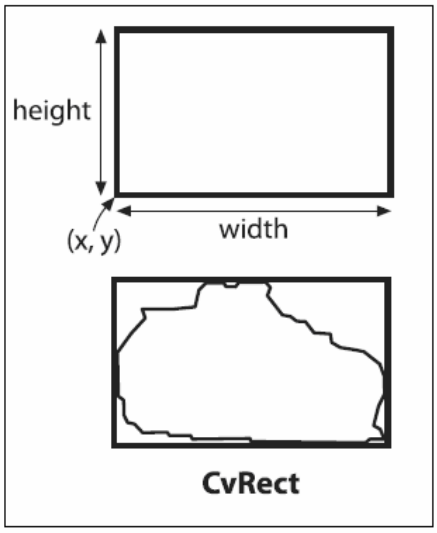


Figure 3-37: CvRect can represent upright rectangles [2]

For the application the point (x,y) in the Figure 3-37 must be in the top left corner, this is because of the tracking functions. The height and width values are taken directly from the result returned by the “cvFitEllipse()” function.

The variable is called “coordinates”. The name is because the four new values are the coordinates for each object.

coordinates.x is equal to Pt1.x, and coordinates.y is equal to Pt2.y, they are the point previously calculated.

coordinates.width is equal to “box->size.width” and coordinates.height is equal to “box->size.height”.

The final result is shown in the Figure 3-38. The coordinates of the object are defined. The same procedure is done for every object found. It is very important to consider that determine the coordinates is necessary because the accuracy for definition of the positions. Thereby the conditions for the tracking are much better if the area occupied for each object is known.

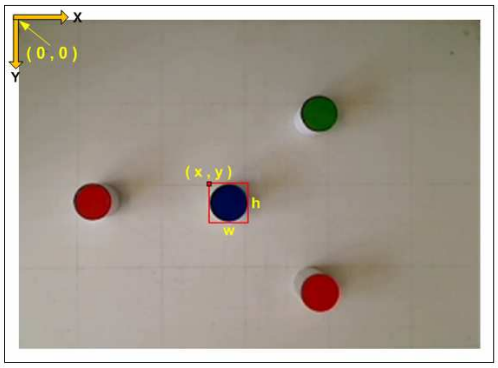


Figure 3-38: Calculation of the object coordinates

3.5 Tracking Algorithm

As opposed to individual still images, there are particular situations, where often there is a particular object or objects, which must be followed through the visual field. The main objective is continuously updates and estimates the position of that object. Given the functions for tracking system, and considering the algorithm for detecting objects, the developed method consists in to make a final algorithm, which must be able to achieve an optical tracking. The tracking algorithm proposed is depicted in Figure 3-39, which contain also the object detection as a part.

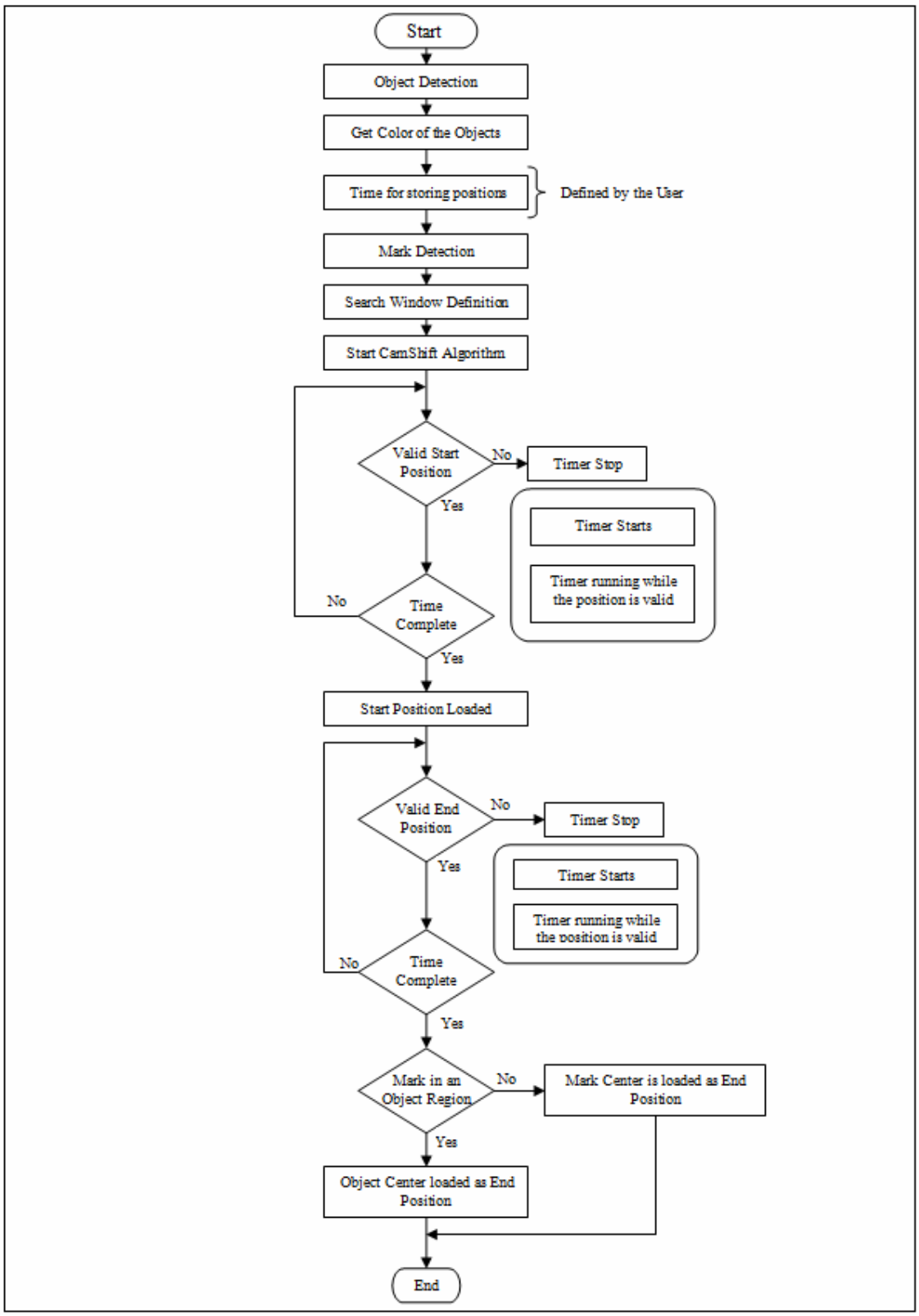


Figure 3-39: Flowchart of the tracking algorithm

For detecting mark is the same procedure implemented for detecting objects. The mark consists in a smaller circle in terms of area in comparison with the objects. When the mark is found the tracking is able to start, otherwise only the detection

keeps executing. The most important part of the tracking is done by a function contained in OpenCV library called “CamShift()”. This function needs certain previous information, which is given by another functions already developed in OpenCV library called “Histograms”. Therefore firstly is suitable a short introduction about the theory behind of these functions.

3.5.1 Histograms

When the detection is realized, the positions of the objects are known. In the case that more information about the objects is required, for instance the color of each object, it is possible to get applying the histograms. For the method proposed also the object color is an interest data, because the program must be able to make a list in order to know how many objects are in total, and also to know the quantity of the certain color. Therefore, the histograms not only are used for the tracking, also for getting the color of each color, in order to have more information about what is exactly a histogram, and how can they work, the next paragraphs are a briefly explanation.

Histogram is considered a discrete approximation of stochastic variable probability distribution. The representation of a Histogram in a programming language it is a variable, which can be both a scalar value and a vector. Histograms are widely used in image processing and computer vision [1].

For example, one-dimensional histograms can be used for [1]:

- Gray image enhancement
- Determining optimal threshold levels
- Selecting color objects via hue histograms back projection, and other operations.

Two-dimensional histograms can be used, for example, for [1]:

- Analyzing and segmenting color images, normalized to brightness (e.g. red-green or hue-saturation images)
- Analyzing and segmenting motion fields (*x-y* or magnitude-angle histograms),
- Analyzing shapes or textures.

Multi dimensional histograms can be used for [1]:

- Content based retrieval
- Bayesian-based object recognition

For designing a tracking system and also for getting the color of the objects, and doing emphasis about the Histogram application as the color distribution, it is possible to develop a short algorithm and solve both tasks. The main idea for the

tracking is the distribution of probabilities representing one current hypothesis about a location of the mark.

Histograms of edges, colors, corners, and so forth, they form a general feature type that it is passed to classifiers for object detection. An application also, it is possible to make sequences of color, or edge histograms are used to identify whether videos have been copied on the web. These are only a few of examples and applications of the Histogram, which are one of the classic and common tools of computer vision [13]. Histograms are simply collected counts of the underlying data organized into a set of predefined bins. They can be populated by counts of features computed from the data, such as gradient magnitudes and directions, color, or just about any other characteristic. They are used to obtain a statistical picture of the underlying distribution of data [13].

OpenCV has a data type for representing histograms. The histogram data structure is capable of representing histograms in one or many dimensions, and it contains all the data necessary to track bins of both uniform and nonuniform sizes. The type of variable is called “CvHistogram()”. OpenCV reference manual gives more details about the histogram functions [1].

When the interest color regions were detected from the incoming video, the edge gradient directions were computed as well around these interest regions, and these directions were collected into orientation bins within a histogram [2].

For creating a probability distribution image of the desired color in the video scene is based on the Hue Saturation Value (HSV) color system, that corresponds to projecting standard Red, Green, Blue (RGB) color space along its principle diagonal from white to black, which is depicted in Figure 3-40. This results in the hexcone in Figure 3-41. Descending the V axis in Figure 3-41 gives a smaller hexcone corresponding to smaller RGB subcubes shown in Figure 3-40 [14]. HSV space separates out hue (color) from saturation, that means, how concentrated the color is, and also it separates from brightness [14].

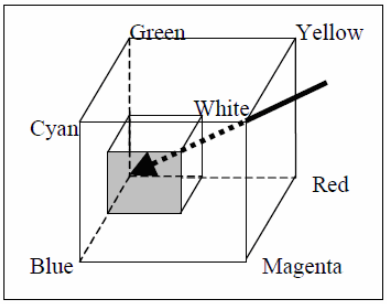


Figure 3-40: RGB color cube [14].

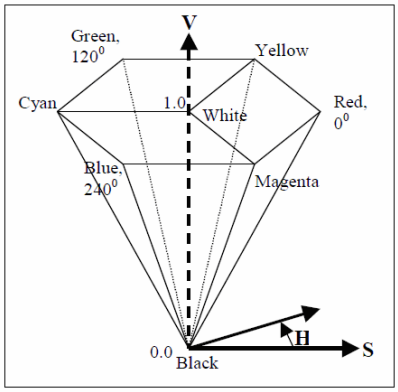


Figure 3-41: HSV color system [14]

One of the most important functions contained in OpenCV library is called “cvCalcBackProject()”, which essentially produces a color probability image from the input color image. The process back-projection uses the HSV data in conjunction with a color histogram [15]. OpenCV provides data structures for histograms and associated functions. The reference manual gives more details [1]. For a current group of pixels taken from the same position from all input single-channel images, the function “cvCalcBakcProject()” puts the value of the histogram bin to the destination image. The coordinates of the bin are determined by the values of the current group of pixels. In terms of statistics, the value of each output image pixel represents a probability of the current group of pixels belonging to an object. Thereby the local feature distribution histogram of the object is given and used [1]. To achieve the creation of a histogram, it must be initialized by sampling a representative area of one frame, it can be done manually. For example, to find a red object in the picture the procedure is as follows [2]:

1. Calculate a hue histogram for the red object assuming the image contains only this object. The histogram is likely to have a strong maximum, corresponding to red color.

2. Calculate back projection using the histogram, and get the picture where bright pixel corresponds to typical colors, in this case red color, in the searched object.

3. Find connected components in the resulting picture and choose the right component using some additional criteria, for example, the largest connected component.

The steps mentioned above are located in the initialization of the tracking algorithm, because the color object tracker must be defined, after that the function “cvCamShift()” works, and the rest of the tracking is calculated by this function. Firstly, before of the tracking algorithm explanation, the color of the objects is

obtained. The method used for this task it is applied immediately after the localization process. This is related with the last step of the detection algorithm. The histograms are used for analyzing the object region and get the color distribution. The position of the centers and the region of the objects are saved in one array respectively. The method consisted of only use the regions, with the coordinates of those regions it is possible to create and to apply a mask of those interest regions. Therefore, an image “RIO” for each object must be created.

In Figure 3-42 shows the detected objects with their regions drawn respectively. Figure 3-42 also permits to appreciate the size of each region. The analysis is done one by one object. The order depends of the number of each contour, for instance, if the contour corresponding to green object was defined as the first one, the mask is created first for that object, and the color distribution is calculated just for the current region. After this, the second region is taken and the procedure is the same for every object found.

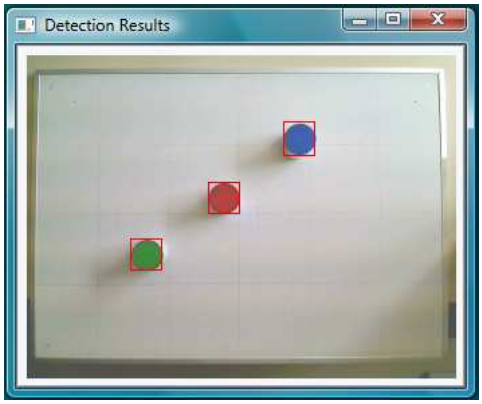


Figure 3-42: Region of the detected objects

The result obtained from a histogram is depicted in Figure 3-43, which permits to appreciate the intensity of the colors of the current region. In this case the result corresponding to one blue object.

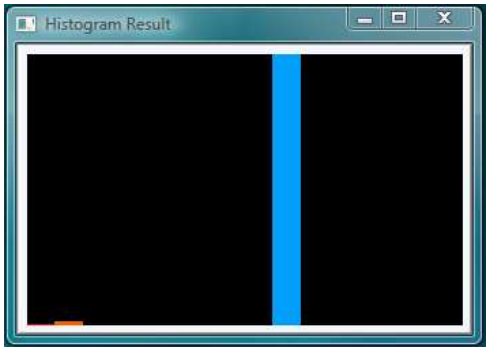


Figure 3-43: Histogram result for one blue object

For determining the object color, the method involves some OpenCV functions. Therefore the following steps give a short explanation about the procedure. OpenCV reference manual gives more details about the functions [1].

- Firstly a histogram must be created using the function called “cvCreateHist()”.
- Two masks are created in order to define the interest region of the image.
- “cvCalcHist()” using the masks with the same dimensions the calculation of the histogram is done.
- “cvGetMinMAxHistValue()” as its name indicates, this functions gives the maximum and minimum value of the histogram.
- “cvGetReal1D()” for getting a specific element of a single-channel array.
- Lastly the color is gotten and returned in a type variable “CvScalar”, which contains four arguments, and those arguments represent the color distribution of the current region.

The method proposed is able to define only three different colors, these colors are blue, red and green. This is due to the color range of OpenCV as already explained it, it is based on the Hue Saturation Value (HSV) color system, that corresponds to projecting standard Red, Green, Blue (RGB) color space along its principle diagonal from white to black. Hence the returned “CvScalar” variable is easy to analyze. The first three arguments are related with the color. For example {0, 127, 255, 0}, that means, the number 255 in the second position corresponding to green color, the Table 3-2 shows the possible values.

0	---	---	255	Blue color
0	---	255	---	Green color
0	255	---	---	Red color

Table 3-2: Positions of the color values in “CvScalar” variable

When the evaluation of the values is realized, the color of each object is defined and saved in its corresponding place in the array, and then the information about the objects is known and complete, which is shown in Figure 3-44. Therefore the program is able to start to execute the tracking algorithm.

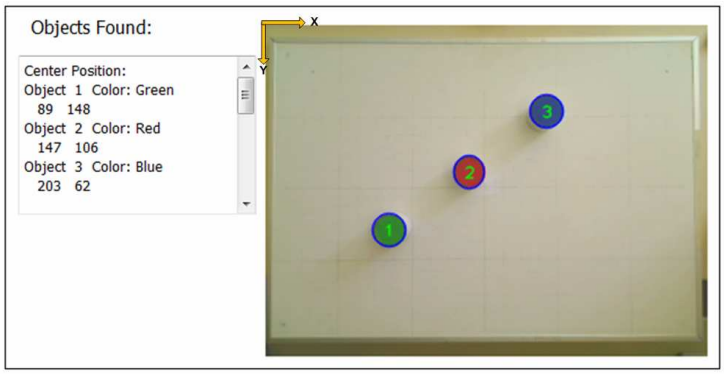


Figure 3-44: Final result of the detection algorithm. The positions are in pixel units, camera coordinates.

The tracking starts at the moment when the mark is found. According to the detection algorithm, the procedure detection mark is the same as the procedure detection object, which has been already explained.Figure 3-45 illustrates the Canny Edge Detection realized for finding the mark, at that precise moment just the mark is searched, the detection algorithm search a smaller ellipse.

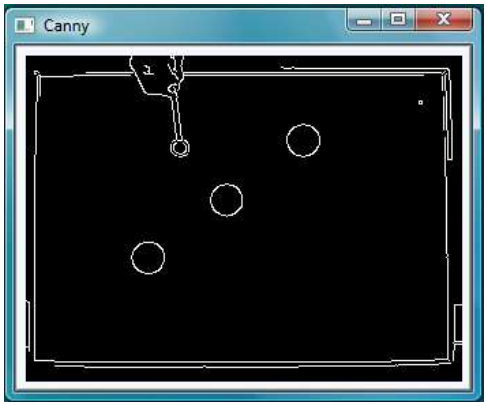


Figure 3-45: Canny Edge Detection for finding the mark

The process is the same as the preceding procedure used for getting the color object, it means, the mark region is defined. It is depicted in Figure 3-46 (left side). The distribution color is calculated for the mark region. In this case is a smaller region, and the histogram result is illustrated in Figure 3-46 (right side).

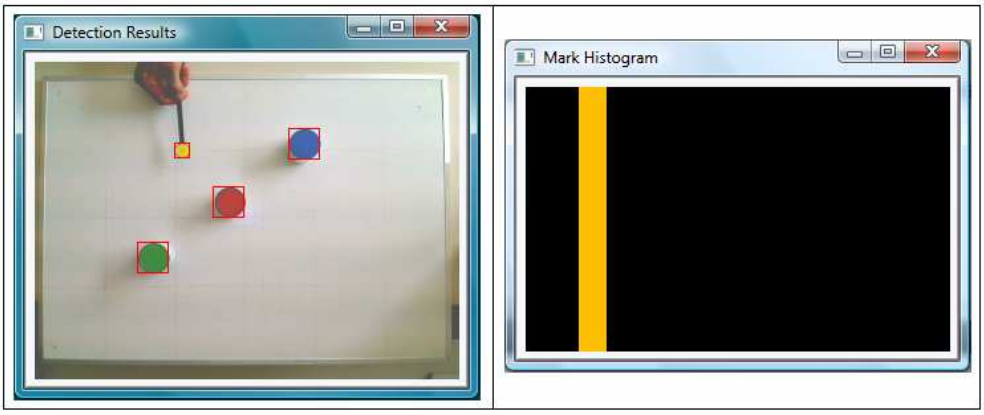


Figure 3-46: Definition of the mark region, and Histogram result of the mark

The mark color is different in comparison with the objects because of the color distribution result. The objects only must be blue, green or red color. Firstly it is because of the detection method of the color, it can just detect these types of colors, and also it is because of the CamShift algorithm, which is explained in the following section.Figure 3-47 shows the moment when the tracking starts. The mark has been found and the histogram has been defined, therefore the yellow color is going to be tracked.

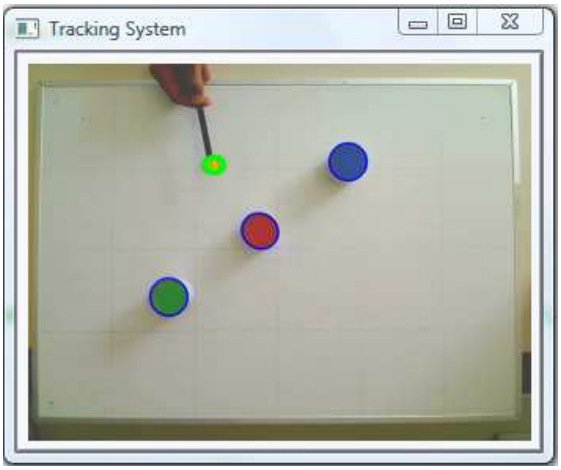


Figure 3-47: Tracking has been started

3.5.2 CamShift Algorithm in OpenCV

The main idea of a Tracking System is to know the actual position of certain object all the time, while the tracking algorithm is kept running. OpenCV library contain a function which allows applying the algorithm for developing a tracking of objects whose size may change during a video sequence. The function is called “CamShift()”. The main function is based in a robust nonparametric technique for climbing density gradients to find the mode of probability distributions called the mean shift algorithm [15].

The mean shift algorithm operates on probability distributions. To track colored objects in video frame sequences, the color image data has to be represented as a probability distribution, using color histograms to accomplish this [14]. Color distributions derive from video image sequences change over time, therefore the mean shift algorithm has to be modified adapt dynamically to the probability distribution it is tracking. The new algorithm that meets all these requirements is called CAMSHIFT [14]. Figure 3-48 depicts the CamShift algorithm. For each video frame, the raw image is converted to a color probability distribution image via a color histogram model of the color being tracked. The center and size of the color object are found via the CAMSHIFT algorithm operating on the color probability image, the gray box shown in Figure 3-48, it depicts the mean shift algorithm. The current size and location of the tracked object are reported and used to set the size and location of the search window in the next video image. The process is then repeated for continuous tracking. CamShift operates on a 2D color probability distribution image [14].

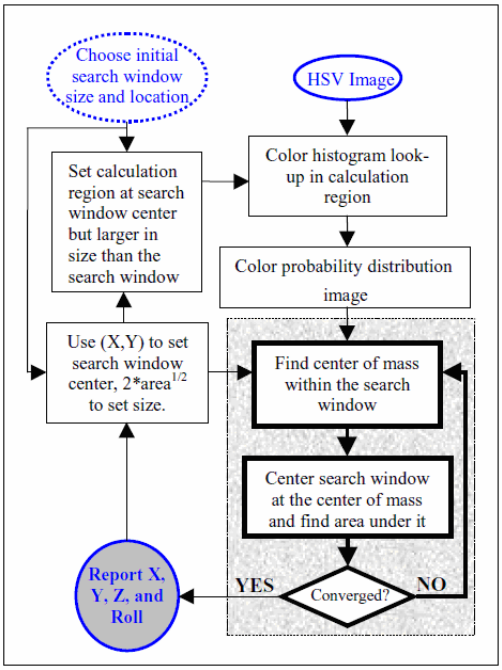


Figure 3-48: Block Diagram of CamShift Algorithm [14]

The Mean Shift part of the algorithm corresponding to gray area in Figure 3-48 it is as follows:

1.-Choose the search windows size.
2.-Choose the initial location of the search window.
3.-Compute the mean location in the search window.
4.-Center the search window at the mean location computed in the step 3.

5.-Repeat Step 3 and 4 until the search window center converges, i.e., until it has moved for a distance less than the present threshold.

The important characteristic of this type of scheme, it introduces a feed-back loop, which the result of the detection it is used as input to the next detection process. The version of CAMSHIFT applying these concepts to tracking of a single target in a video stream it is called “Coupled CamShift” [14].

3.5.2.1 Mass Center Calculation for 2D Probability Distribution

The mean-shift algorithm is briefly described in this section. It is related to the discipline of kernel density estimation where by “kernel” it is possible to refer to a function, which has mostly local focus (e.g., a Gaussian distribution) [2].

With enough appropriately weighted and sized kernels located at enough points, it can express a distribution of data entirely in terms of those kernels. Mean-shift diverges from kernel density estimation in that it searches only to estimate the gradient, in other works, the direction of change of the data distribution. When this change is 0, that means, it is at the stable peak of distribution [2]. For discrete 2D image probability distribution, the mean location, that means the center within the search window, Steps 3 and 4 above, it is founds as follows. These equations are by considering a rectangular kernel [2]:

Find the zeroth moment

$$M_{00} = \sum_x \sum_y I(x, y). \tag{3.25}$$

Find the first moment for x and y

$$M_{10} = \sum_x \sum_y xI(x, y). \tag{3.26}$$

$$M_{01} = \sum_x \sum_y yI(x, y) \tag{3.27}$$

Mean search window location, the centroid, then it is found as

$$x_c = \frac{M_{10}}{M_{00}} \tag{3.28}$$

$$y_c = \frac{M_{01}}{M_{00}} \tag{3.29}$$

Where $I(x, y)$ is the pixel (probability) value in the position (x, y) in the image, and x and y range over the search window.

Therefore the mean-shift algorithm may be used for tracking because it is the base of CamShift algorithm. Firstly feature distribution to represent an object is defined, it

could be color and texture, and after that mean-shift window is started over the feature distribution generated by the object, and finally compute the chosen feature distribution over the next video frame [2]. Starting from the current window location, the mean-shift algorithm is able to find the new peak or mode of the feature distribution, which is expected it is centered over the object, which produced the color and texture in the first place. Thereby, the mean-shift window tracks the movement of the object frame by frame. Figure 3-49 illustrates example of a two-dimensional distribution of data and an initial rectangular window. The arrows indicate the process of convergence on a local mode, or peak, in the distribution. It is possible to appreciate as it is expected that, this peak finder is statistically robust in the sense that points outside the mean-shift window do not affect convergence, such as the algorithm is not “distracted” by far-away points [2].

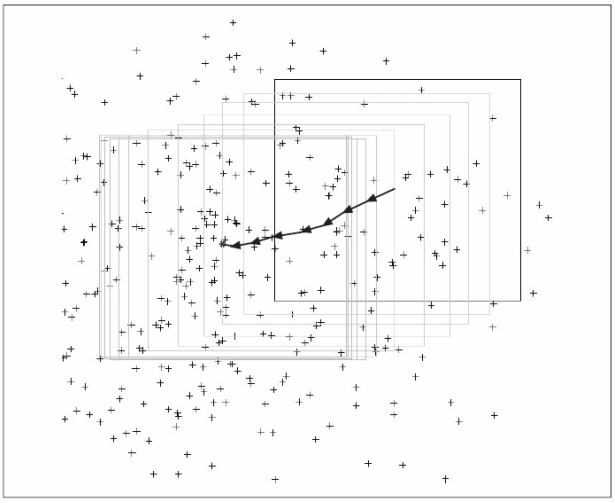


Figure 3-49: Mean-shift algorithm in action: an initial window is placed over a two-dimensional array of data points, and the center is successively updated over the mode, or local peak, of its data distribution until convergence [2]

CamShift algorithm is distinguished from Mean Shift algorithm, which is designed for static distributions whereas CamShift is designed for dynamically changing distributions. The main difference is CamShift adjusts itself the size search window. When an object in video sequences is being tracked and the object moves, its size and location of the probability distribution change in time. To consider the main characteristic of the CamShift algorithm, which consist of to adjust the search window size in the course of the time. The Initial window size can be set at any reasonable value. For discrete distributions, this means digital data, the minimum window length or width are three [14]. Instead of a set, or externally adapted window size, CamShift relies on the zeroth moment information. It extracted as part of the internal workings of the algorithm, to continuously adapt its window size within or over each window frame. The zero moment can be defined as the distribution “area” found under the search window [14].

The objective of the following example it is to explain how the CamShift works, and also the example permit to see one of the most important advantages of this algorithm. In this case, the example consists in a particular application, which is about a face tracking, but also it is using a color distribution. Therefore, it is the process that the tracking algorithm proposed is based. The example shown in Figure 3-50, which begins the search process at the top left step by step down the left then right columns until convergence at bottom right. The red graph is a 1D cross-section of an actual sub-sampled flesh color probability distribution of a face image, and a nearby hand. The yellow color is the CamShift search window, and purple is the mean shift point. The ordinate is the distribution value, and the abscissa is the horizontal spatial position within the original image. The window is initialized at size three and converges to cover the tracked face but not the hand in six iterations. In this sub-sampled image, the maximum distribution pixel value is 206. Therefore the width of the search window is set to $2 * M_0 / 206$ [14].

The typical behavior of CamShift it can be appreciated, because firstly it finds the center of the nearest connected distribution region, in this particular case the face, but it ignores another type of color, it is not distracted, for instances the hand.

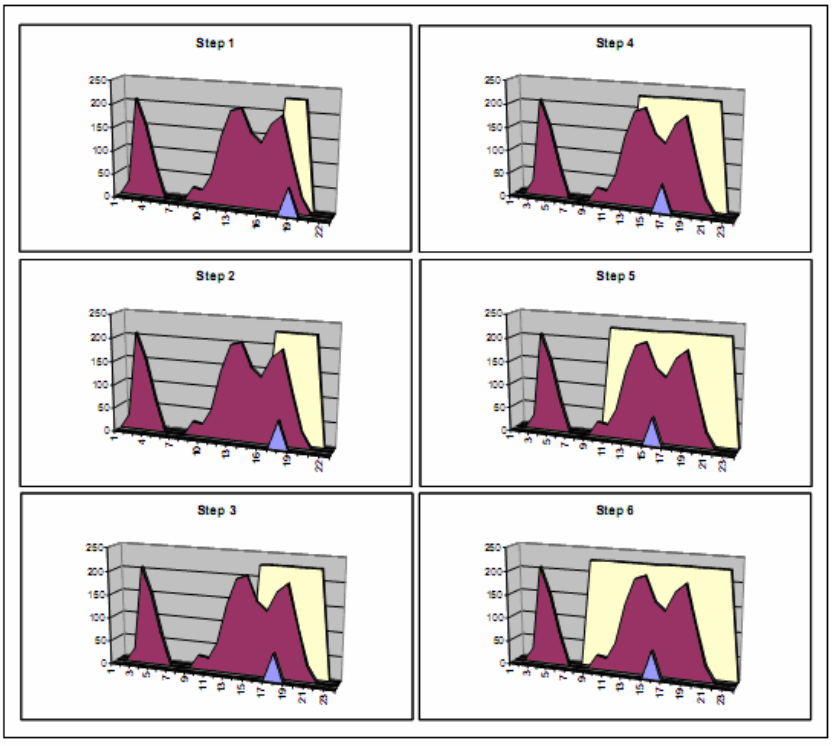


Figure 3-50: Cross Section of Flesh Hue Distribution [14]

Figure 3-51 shows frame to frame tracking. The red color probability distribution has shifted left and changed form. At the graph in left side, the search window starts at its

previous location from the bottom right in Figure 3-50. In one iteration it converges to the new face center [14].

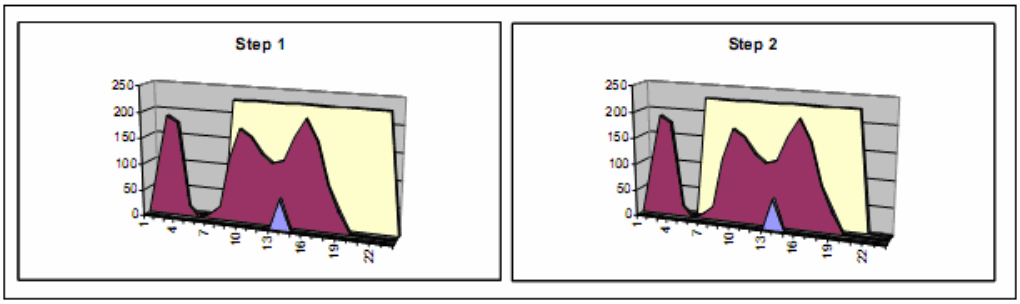


Figure 3-51: Example of CamShift tracking starting from the converged search location in Figure 3-50 bottom right [14]

Calculate the Coupled CamShift Algorithm [14]

1. Set the calculation region of the probability distribution to the whole image.
2. Choose the initial location of the 2D mean shift search window.
3. Calculate the color probability distribution in the 2D region centered at the search window location in an RIO slightly larger than the mean shift window size.
4. Run mean shift algorithm to find the search window center. Store the zeroth moment (area size) and center location.
5. For the next video frame, center the search window at the mean location stored in Step 4 and set the window size to a function of the zeroth found there. Go to Step 3.

The method proposed for developing an optical tracking system it is applying the CamShift algorithm, where the feature used it is a color distribution given by a histogram. These functions have been already explained. Therefore considering all the advantages that CamShift algorithm offers, the definition of the tracking system is as follows.

The optical tracking system must be able to assist human in handling of objects, and pick and place operations. Therefore two positions must be defined. The positions are related with the center position of each object given by the detection algorithm.

According with the detection algorithm, not only the positions of the center of the objects are known, also the regions occupied for each object are known, thereby the both positions that the tracking must define are going to be storage automatically.

The precise moment for storing the positions it can be known because the actual position of the mark is always known, because the mark is tracked, and then it is possible to verify when the mark is inside of certain object region. This is the main

reason that the mark is defined as smaller than the objects it is in order to do that analysis.

As is shown in Figure 3-52, yellow color has been defined as the feature which must be tracked. This principle permits to see one of the advantages of the CamShift algorithm. If the object is red, green or blue color it does not care, just the yellow color is tracked. Thus it is possible to achieve the selection of certain object through the procedure which requires the actual location of the mark for assuring if an object has been selected or not.

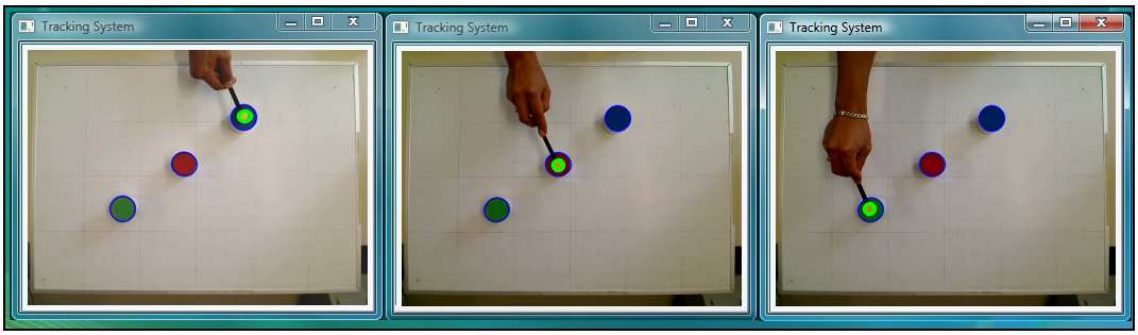


Figure 3-52: Tracking Mark in region of the different objects

Figure 3-53 below shown, it is used for visualizing the possible background distractors for the mark, it is clear visible that because of the color of the objects is different in comparison with the color mark there are no distractors, thereby the tracking works very well.

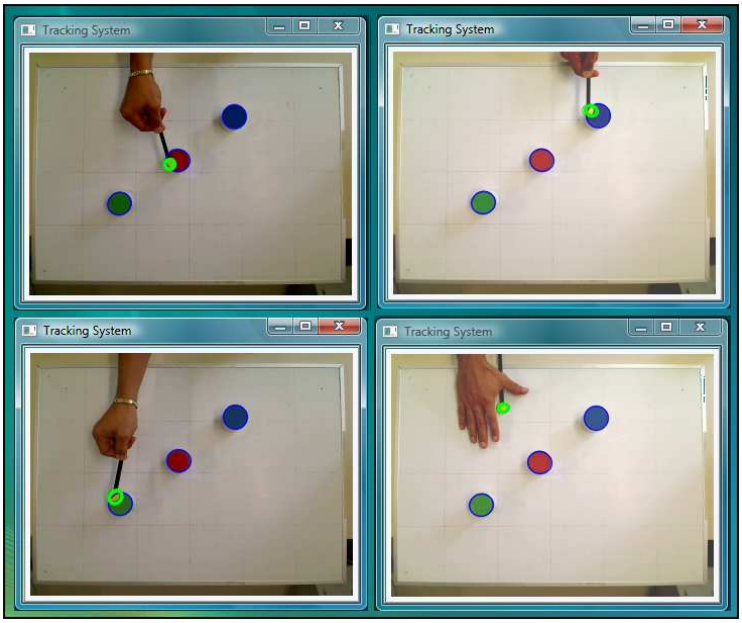


Figure 3-53: Tracking a yellow color with background distractor colors.

In contracts with the previous tests, if an object is the same color as the mark, or vice versa, there is a problem because of the distribution color and also because of the CamShift algorithm. This case is clear visible when the method used for resizing the search window is explained in subsection 3.5.2.3 Window Size and Placement.

3.5.2.2 Calculation of 2D Orientation

The 2D orientation of the probability distribution it is also easy to obtain. Using the second moments in the course of CamShift operation, where (x, y) range over the search window, and $I(x, y)$ is the pixel (probability) value at (x, y) [1].

Second moments are [1]

$$M_{20} = \sum_x \sum_y x^2 I(x, y) \tag{3.30}$$

$$M_{02} = \sum_x \sum_y x^2 I(x, y) \tag{3.31}$$

Then the object orientation, major axis, it is [1]

$$\theta = \frac{\arctan\left(\frac{2\left(\frac{M_{11}}{M_{00}} - x_c y_c\right)}{\left(\frac{M_{20}}{M_{00}} - x_c^2\right) - \left(\frac{M_{02}}{M_{00}} - y_c^2\right)}\right)}{2} \tag{3.32}$$

The first two eingenvalues, major length and width, of the probability distribution "blob" , they found by CamShift, it may be calculated in closed form as follows [1]. Let

$$a = \frac{M_{20}}{M_{00}} - x_c^2 \tag{3.33}$$

$$b = 2\left(\frac{M_{11}}{M_{00}} - x_c y_c\right) \tag{3.34}$$

And

$$c = \frac{M_{02}}{M_{00}} - y_c^2 \tag{3.35}$$

Then length l and width w from the distribution of the center are [1]

$$l = \sqrt{\frac{(a+b) + \sqrt{b^2 + (a-c)^2}}{2}} \tag{3.36}$$

$$w = \sqrt{\frac{(a+b) - \sqrt{b^2 + (a-c)^2}}{2}} \,. \qquad (3.37)$$

Applying the previous equations the orientation of the tracked object is possible to calculate it. Figure 3-54 shows the case when the CamShift algorithm has calculated the correct positioning of the tracked mark, which has been rotated. Left side depicts a normal position and right side depicts a rotated position in Figure 3-54.

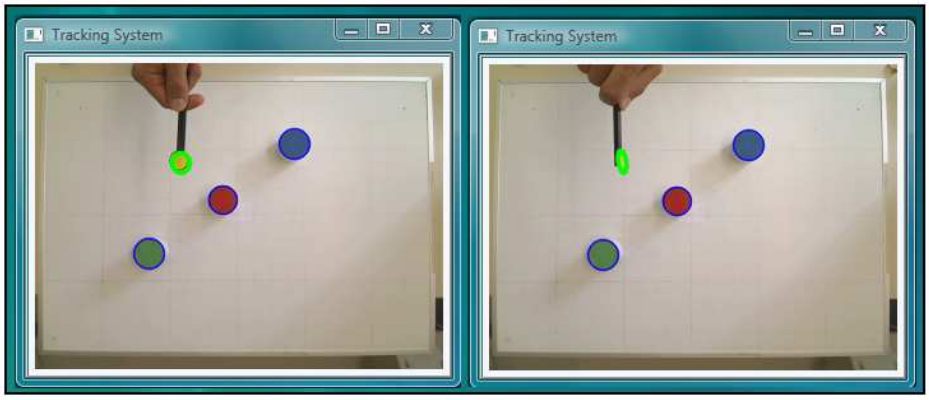


Figure 3-54: Tracking a mark which has been rotated

3.5.2.3 Window Size and Placement

Because the position of the tracked object changes continuously, and also its size, according with the CamShift algorithm the actual location of the mark is updated all the time, thereby certain conditions must be defined in order to achieve this task.

CamShift is an algorithm that climbs the gradient of a distribution, and the minimum search window size must be greater than one in order to detect a gradient. Also, It is in order to center the window, it should be of odd size [14]. Thus for discrete distributions, the minimum window size is set at three [14].

Therefore CamShift adapts its search window size, which is rounded up to the current or next greatest odd number. At start up, the color probability is calculated from the whole scene through the zeroth moment for setting the window size and also the center.

For adapting Window Size the first consideration is to translate the zeroth moment information into units, which make sense for setting window size.

Thus, regards to Figure 3-50 located in section 3.5.2.1, the maximum distribution value per discrete cell is 206. Thereby the zeroth moment is divided by 206 to convert

the calculated area under the search window to units of number of cells. The goal is to track the whole color object. Therefore an expansive window is needed [14].

Thus, the result is multiply by two, and then the window grows to encompass the connected distribution area. After this, the next greatest odd search window size is rounded, and then the center window is defined.

For 2D color probability distributions where the maximum pixel value is 255, the window size s is equal to

$$s = 2 * \sqrt{\frac{M_{00}}{256}} \tag{3.38}$$

It is divided by 256 for the same reason stated above, but to convert the resulting 2D region to a 1D length, the square root is needed [14].

Concerning to the mark used for the optical tracking proposed, the Figure 3-55 shows the behavior and the response of the CamShift algorithm at the moment when the mark is closer to the camera lens. Thereby the size of the mark increase and the quantity of the color distribution increase as well. Therefore the search window must be resized according with the calculus given by “cvCamShif()” function. Also the back-projection process is involved. Even when the mark is rotated as is shown in Figure 3-55 in right side.

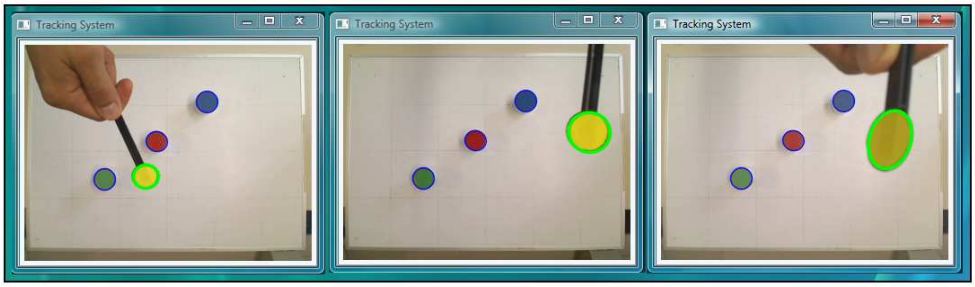


Figure 3-55: CamShift algorithm resizing the search window

In the following example illustrated in Figure 3-56, a mark has been defined with red color as a feature, and it is tracked. This is a special situation due to the red color also is a feature of one object already found. The CamShift behavior can be appreciate in Figure 3-57. The problem is not with different colors, the algorithm is able to differentiate the mark when it is closer to blue or green color as already shown. The main problem is with the red color due to the CamShift algorithm is updating the location of the mark. The back-projection process also is executed at new iteration. Therefore if the quantity of the color covers a major area firstly the

back-projection is calculated and after the CamShift algorithm as well in order to know the actual location of the tracked object, in this case the mark location.

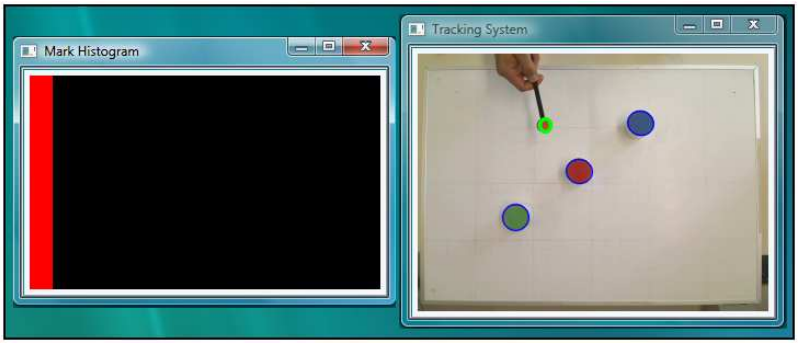


Figure 3-56: Definition of the feature mark as a red color

Figure 3-57 permits to see the response of the CamShift algorithm, it is updating and resizing the location of the mark, even though for the application of the optical tracking proposed is a wrong behavior.

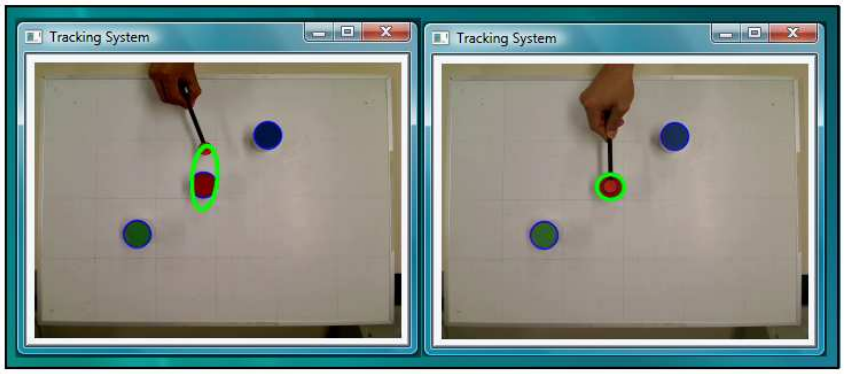


Figure 3-57: Search window resized

Regarding the previous results, this is the main reason that the color of the objects and the color of mark using for the tracking should be different each other.

4 Integration and Experiments

This Chapter presents the integration of the algorithms. Also it is presented the main results of each one, and the relationship between them. The algorithms have been already explained in Chapter 3.The integration begins with camera calibration. Conditions, restrictions and results are presented related with the calibration stage, and its influence in the system response. If the calibration is correct the system response also is going to be correct.When the problem of distortion is solved the response of the detection is analyzed, the expected behavior of the detection without lens distortion must be linear.

4.1 Camera Calibration

The calibration consists of to apply the algorithm proposed already described in Chapter 3. The window has two options for doing this task. The user is able to choose between them, one possibility it is to apply directly the calibration algorithm by selecting the option “Pictures From File” corresponding to type of calibration. This type of calibration uses the pictures already saved in a file. The quantity of the pictures is 50. The internal procedure is the same in both cases, the main difference is that, if the user chooses the option “Take Pictures” as is shown in Figure 4-1, the user is able to take the pictures directly in that moment. Figure 4-1 also shows the moment when the calibration has started, the program is waiting for the calibration patter or “chessboard”. If the corners are not found it is not possible to take the picture. It is possible until the corners have been found, as it is shown in bottom right corner in the same Figure 4-1.The number of samples, the size of the chessboard and the output file name must be defined before starting the calibration. However, the fields have a minimum and maximum default value.

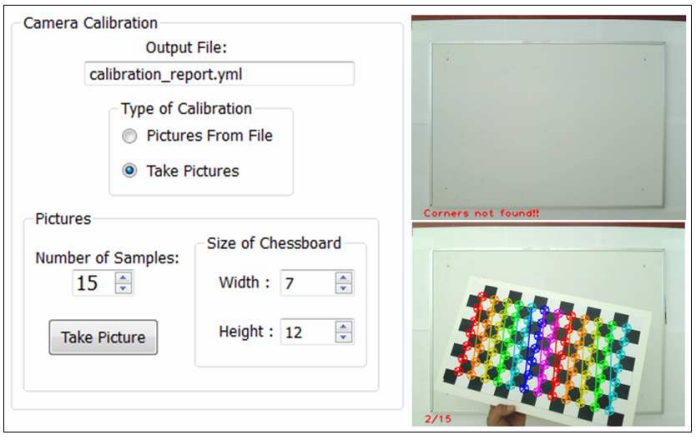


Figure 4-1: Camera Calibration done by the user

Two possible situations could be presented at calibration. One of them, it is when the corners are not found it is easy to know because the corners are not drawn. Instead a red circles appear, and a message as well. This situation depends on both, when the chessboard size is not correct or also, it is possible to happen because of chessboard orientation. If the corners are drawn, it not always means that it is correct. There is a problem when the chessboard orientation is almost the correct for the camera view, or when the distance between the camera and the chessboard is so large such as the function “cvFindChessboardCorners()” cannot define the exact position of the corners. Thereby the corners are not drawn correctly. Hence the calibration result is going to be totally wrong. Both situations are illustrated in Figure 4-2, left and right side respectively.

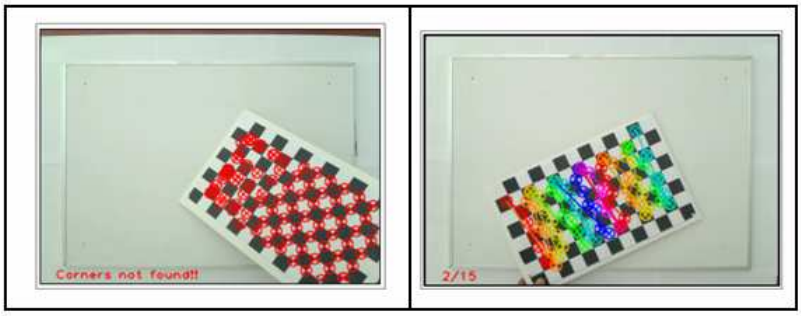


Figure 4-2: Special situations at calibration

4.1.1 Lens Distortion

The main objective of the calibration algorithm is to know the camera matrix, and to correct for distortion effects through distortion coefficients. The assumption that a linear model is an accurate model of the imaging process is different from the world point, due to image point and optical center are collinear, and world lines are imaged as lines and so forth.

Because of the manufacturing process of the lens, in practice, no lens is perfect, hence it is much easier to make a spherical lens than to make more mathematical model parabolic lens. The most important deviation is generally a radial distortion. Even through also tangential distortion is present.
In general the difference is that radial distortions arise as a result of the shape of lens, whereas tangential distortions arise from the assembly process of the camera as a whole. However, in practice the distortion error becomes more significant as the focal length of the lens decreases [2].

Principally in the applications such as computer vision, it requires distortion compensation for the accurate location.

The correction must be carried out in the right place in the projection process. Lens distortion takes place during the initial projection of the world onto the image plane. The resulting calibration matrix reflects a choice of affine coordinates in the image, translating physical locations in the image plane to pixel coordinates.

The radial distortion is 0 at the optical center of the imager. If one object is located at the optical center, there is no distortion, but if the object is moved toward the periphery the distortion increases. This distortion effect is depicted in Figure 4-3, which permit to appreciate the difference between the view with and without distortion.

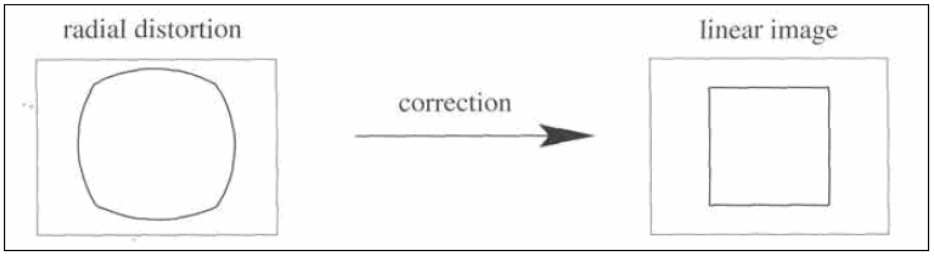


Figure 4-3: The image of a square with significant radial distortion is corrected. It would have been obtained under a perfect linear lens [19]

In practice, this distortion is small and can be characterized by the first few terms of a Taylor series expansion around $r = 0$ [17]. For cheap web cameras, generally are used the first two such terms. The first of them is conventionally called k_1 and the second k_2. Concerning just for highly distorted cameras such as fish-eye lenses a third radial distortion term k_3 is used [2].

Let (u, v) be the ideal (non-observable distortion-free) pixel image coordinates, and $(\breve{u}, \breve{v})$ the corresponding real observe image coordinates. The ideal points are the projection of the model points according to the pinhole model, which is already described in Chapter 3. Similar, (x, y) and $(\breve{x}, \breve{y})$ are the ideal (distortion-free) and real (distorted) normalized image coordinates.

$$\breve{x} = x + x\left[k_1(x^2 + y^2) + k_2\left(x^2 + y^2\right)^2\right]$$
$$\breve{y} = y + y\left[k_1\left(x^2 + y^2\right) + k_2\left(x^2 + y^2\right)\right],$$

As already mentioned k_1 and k_2 are the coefficients of the radial distortion. The center of the distortion is the same principal point. From $\breve{u} = u_0 + \alpha\breve{x} + \gamma\breve{y}$ and $\breve{v} = v_0 + \beta\breve{y}$ and assuming $\gamma = 0$, and then

$$\breve{u} = u + (u - u_0)\left[k_1\left(x^2 + y^2\right) + k_2\left(x^2 + y^2\right)^2\right] \quad (4.1)$$

$$\breve{v} = v + (v - v_0)\left[k_1\left(x^2 + y^2\right) + k_2\left(x^2 + y^2\right)^2\right] \quad (4.2)$$

Estimating Radial Distortion by Alternation: As the radial distortion is expected to be small, the others five intrinsic parameters would be to estimate by simply ignoring distortion. The strategy could be to estimate k_1 and k_2 after having estimated the other parameters, which give the ideal pixel coordinates (u, v). Then from Equation 4.1 and Equation 4.2 two equations for each image are obtained:

$$\begin{bmatrix} (u - u_0)\left(x^2 + y^2\right)^2 & (u - u_0)\left(x^2 + y^2\right)^2 \\ (v - v_0)\left(x^2 + y^2\right)^2 & (v - v_0)\left(x^2 + y^2\right)^2 \end{bmatrix} \begin{bmatrix} k_1 \\ k_2 \end{bmatrix} = \begin{bmatrix} \breve{u} - u \\ \breve{v} - v \end{bmatrix}$$

Given m points in n images, it is possible to stack all equations together to obtain in total $2mn$ equations, or in matrix form as $Dk = d$, where $k = [k_1, k_2]^T$. The linear least-squares solution is given by:

$$k = \left(D^T D\right)^{-1} D^T d \quad (4.3)$$

OpenCV library provides an undistortion algorithm that takes the camera matrix and the distortion coefficients from “cvCalibrateCamera2()” and produces a corrected image. The access to that algorithm is through the function called “cvUndistort2()”.

When the calibration is executed, which is explained in Chapter 3, it is simple to apply this function that OpenCV library offers, its structure is shown in Figure 4-4.

```
void cvUndistort2(
  const CvArr*   src,
  CvArr*         dst,
  const cvMat*   intrinsic_matrix,
  const cvMat*   distortion_coeffs
);
```

Figure 4-4: Structure of the “cvUndistor2()”.[2]

The procedure consist of the actual image, which is from the capturing video, it is load in temporal variable. Thereby the temporal variable becomes in the input for the function (src), and the main image is the output of the function (dst). The intrinsic matrix and distortion coefficients are taken and used directly.

Figure 4-5 shows the moment when the calibration has finished, and then it is possible to apply the distortion function in order to test if the calibration is correct or not. A message appears which says that the camera is already calibrated but the option undistortion in off mode, it is corresponding with the left side. Right side shows the image without distortion, the difference between the figures is clear, and the

distortion of the camera is solved drastically, in other words it is considered as a good calibration.

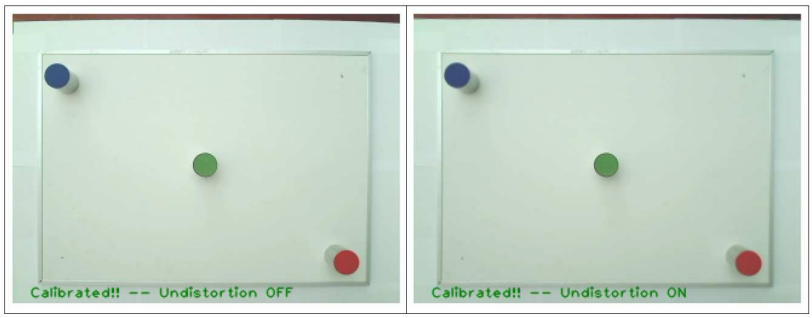


Figure 4-5: Camera calibration done

The calibration result is much easier to appreciate at the object detection. Both situations are presented. Figure 4-6 shows the influence of the distortion. The main problem is reflected in the objects 1 and 3, which are located in the corners of the image. The values of their position are expressed in world coordinates, which corresponding to millimeters, (it is explained in the next section). Doing a comparison between both situations, it is possible to see a considerable difference between them. Therefore really the camera calibration proposed is able to solve the distortion problem. It is helpful because of the accuracy of the robot.

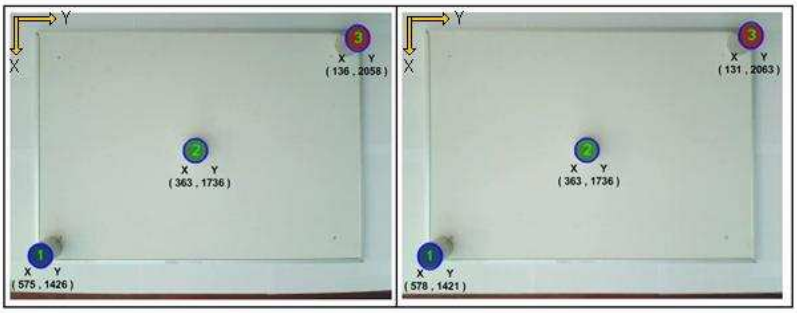


Figure 4-6: Object detection. Undistortion Off and Undistortion On respectively

At undistortion is applied, the response of the detection is much better. The test in order to know the detection response it was done according with the values of the different positions. All the positions had a common reference point. An object located in the image center that means, in the coordinate (160,120), it was considered as a point reference. Moving the objects towards of the axes in steps with the same size equal to 120mm, (they were considered 120mm concerning to the minimum distance which the robot is able to take an object), the difference between steps it kept almost the same value. The variation was ±0.151180 pixel, which in millimeters corresponds to 0.359808mm in X_{robot} axis and 0.365970mm in Y_{robot} axis. The values for X axis and Y axis are different because of non-uniform scaling, as explained in Chapter 3. These values are in range for the robot. Therefore, the assumption that the response

of the object detection without distortion it solved by calibration algorithm proposed it is linear.

4.1.2 World Coordinates and Camera Coordinates

The detection algorithm delivers the position of the centers of each ellipse found, but the value for each position it is expressed in pixel units. The resolution of the image is 320 x 240 pixels. This resolution is because of the OpenCV library. The total area of the camera view, it is related with this values.

These positions must be sent to the robot system, but the problem is for example, the coordinate expressed in pixel units (186,136), it does not make sense for the robot, therefore a coordinate transformation is needed.

As already mentioned in Chapter 3, because of the Homography the assumption that the model plane is on $Z = 0$, and also regarding that there is no distortion, one solution for transforming from camera coordinates to world coordinates it is obtained a scale factor.

The camera is fixed, therefore the area of the camera view it is defined by the resolution of the image. This area in comparison with the robot area it is very small. Hence the method consist of to know and to define a movement area for the robot, which must be delimited by the camera view.
Firstly the range of the movement area is it defined by the white board, which has dimensions equal to 515mm x 710mm. The camera is located to 695mm in relation with the white board. The coordinates of the robot must be expressed in millimeters.
Five objects are needed for calculating the scale factors. Two different scale factors are expected, because of the non-uniform scaling, it is explained in Chapter 3. A main reference point is needed, which can be located at the image center. The image center is considered as the coordinate (160,120). The other four objects are located in the corners of the white board, top and bottom, left and right respectively. This is in order to know the total dimension of movement area for the robot.

Because of the orientation of the camera, the axes of the camera and the axes of the robot they are in different position each other. This means, X axis of the camera it is equivalent to the Y axis of the robot, and Y axis of the camera it is to equivalent to X axis of the robot. The sense of both it is in the same way. A definition of two different planes it is necessary. One plane is defined as (X_{cam}, Y_{cam}) corresponding to the camera coordinates, and the other one is defined as (X_{robot}, Y_{robot}) corresponding to the world coordinates or robot coordinates. The general idea about the method it is depicted in Figure 4-7.

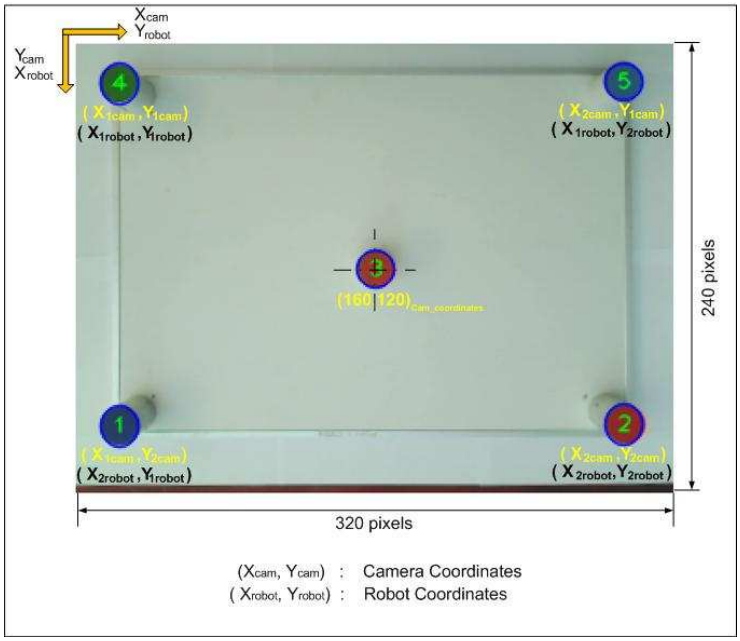


Figure 4-7: Calculation of scale factor. Conversion from camera coordinates to world coordinates

After definition of the coordinate planes and also the five positions of the objects, the scale factors can be calculated as follows:

The five positions are known using the object detection. Therefore the total distances of the axes in camera coordinates they can be calculated as the differences between them. Four differences can be calculated, but it is expected that two of them are equal. Hence only two differences are obtained.

$$x_{2cam} - x_{1cam} = \Delta x_{cam} \ ,$$
$$y_{2cam} + y_{1cam} = \Delta y_{cam} \qquad (4.4)$$

The procedure with the coordinates of the robot it is the same, but the equivalent positions according with the objects they must defined manually with the program of the robot. At the moment when the five positions of the objects in world coordinates are known, the total distances can be calculated, but now, they are expressed in world coordinates.

$$x_{2robot} - x_{1robot} = \Delta x_{robot} \ ,$$
$$y_{2robot} + y_{1robot} = \Delta y_{robot} \qquad (4.5)$$

Using the results given by the Equation 4.4 and Equation 4.5 the scale factors are defined as:

$$scale\_factor\_x = \frac{\Delta x_{robot}}{\Delta y_{cam}} \ ,$$

$$scale\_factor\_y = \frac{\Delta y_{robot}}{\Delta x_{cam}} \qquad (4.6)$$

The next step is to regard the reference points. These are the center of the image, and its equivalent value in robot coordinates. When a new position is transforms the complete procedure is realized.

For example the center of the image (160,120), its equivalent value in robot coordinates it is equal to (375, 1747). Where the scale factors, which corresponding to Equation 4.6, they are the next:

$$scale\_factor\_x = 2.380073 mm / pixels$$
$$scale\_factor\_y = 2.42076 mm / pixels$$

and then, the new position is the coordinate (242,150) in camera coordinates, the transformation procedure is as follows:
The absolute value of the distance according with the image center is calculated.

$$X_{cam} = |242 - 160| = 82\, pixels$$
$$Y_{cam} = |150 - 120| = 30\, pixels$$

Those values are multiplied by the scale factor respectively

$$X_{robot} = 2.380073 * 82 = 195.1659 mm$$
$$Y_{robot} = 2.42076 * 30 = 72.6228 mm$$

The values must be integers, hence they are rounded. And the, because the position is located after the reference point in both axes, the results obtained in the previous step are added to the reference point, but in world coordinates, such as:

$$X_{robot} = 375 + 195 = 570$$
$$Y_{robot} = 1747 + 73 = 1820$$

Therefore the coordinate (142,150) in camera coordinates its equivalent is the coordinate (570, 1820) in world coordinates. This value can be sent to the robot, and now it knows where that position is.

The important consideration is that, if the distance between the camera and object plane change, the scale factors must be calculate again. Also if the camera keeps the same distance but with different orientation of the view, the image center change, therefore, the positions and the relation pixels/millimeters change as well.

4.2 Object Detection

The experiments realized concerning with the detection algorithm proposed they were achieved without lens distortion. The analysis of the positions is the most interesting part because of the accuracy of the system. The values of the positions are expressed in world coordinates (the conversion of the coordinates it is explained in the preceding section), in order to see much clear the behavior of the detection and its influence directly in the values of the coordinates calculated.

Firstly, the GUI after detection it shows a list with all the valid object found. The information given about the objects is its concerning with the center position, X and Y axis respectively and the number of the corresponding object. But that information could be unclear for the User because perhaps there is more than one red object. The numbers of the objects are assigned according with the order of the contours found. Hence for identifying the objects, the GUI has a button which permits to draw the corresponding number to the objects. It is illustrated in Figure 4-8.

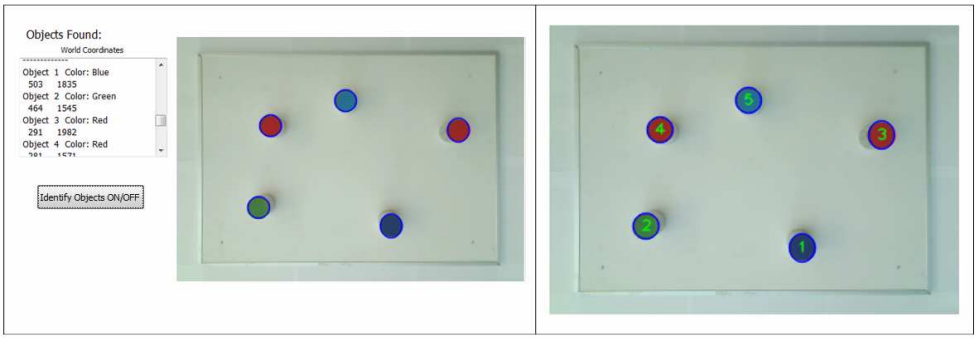


Figure 4-8: Objects detected and identified

So far all the examples shown, which are related with the object detection they have been presented only in a single level. But the scenario when one object is located over another object, it has not been analyzed yet. Figure 4-9 shows the algorithm behavior when the scenario mentioned occurs.

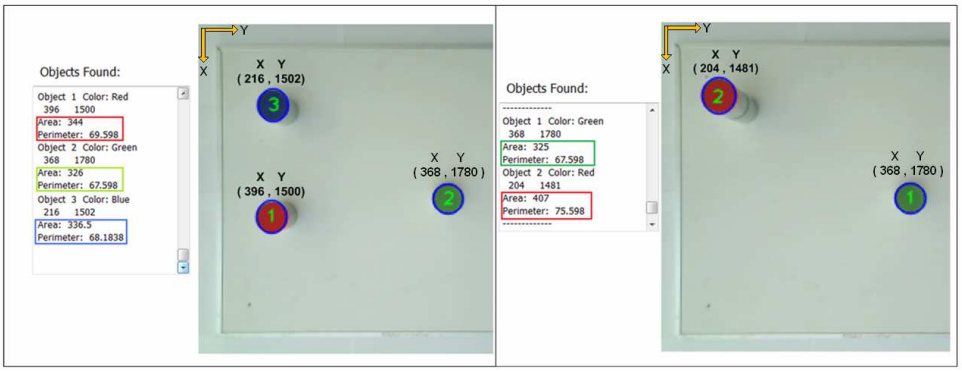


Figure 4-9: Value of the area depends of the distance, and also the position changes because of camera view

The definition of a valid object it is based on the area value of the corresponding contour. Hence if the physical distance of an object is closer to the camera, thereby the area of the object change and the perimeter as well.

But not only those values are affected, also the position of the center object is involves, due to the perspective of the camera view. It is also possible to appreciate in Figure 4-9 in the left side, object 3 and object 1 they are alignment each other in Y axis. At detection after object 1 is moved and it is located over object 3, the new position is shifted in both axes in comparison with the previous one. The new area value for the object 2 is 407 area units, when in the previous detection it was the object number 1 and its area was 326 area units. There is a difference higher that 50, therefore, it is drastically changed.

Figure 4-10 shown below, it permits to see the scenario when a third object is moved and after that it is located over the object 2, which is already in a second level. Therefore it is an especial situation and two problems could be occurs.

One of them is about the area range previously defined directly in the program code (the area range has been modified for this experiment), due to this, it is possible that the actual area value of the object it could be out range and then the object is not detected.

The second problem is related with the camera view, principally when this situation occurs in a corner of it. For example, if another object is moved and after it is located over object 1, it is shown in Figure 4-10 right side. The fourth object is going to be located out side of the camera view, because of its perspective, and then, the object is not detected. Also it is easy to appreciate that the position of the object 1 (right side Figure 4-10) it is shifted drastically in both axes in comparison with object two (left side Figure 4-10). Therefore is suitable not do this, especially in the corners.

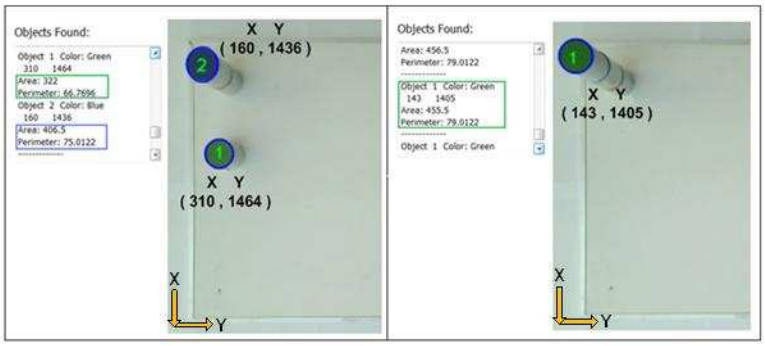


Figure 4-10: Three objects, the perspective change drastically because of the camera view

The mayor problem is, regarding that the position is shifted, and concerning with the conversion of the coordinates explained in the previous section, the robot is not able to take an object if the object is located over another object because of the factors. One solution for this problem be implemented, it consist of to find a relation between the values of the area and perimeter per level, thereby defining different ranges according with those values it is possible to do. The transformation of the coordinates must be also for different levels, because the physical distance change and the factors as well.

The previous examples show especial situations, the objects can be detected under these conditions. If the range defined for the area value is larger in order to detect the objects in different levels a problem can be occurs. This problem could be a “false detection”.
False detection means, the algorithm could consider any ellipse contains in the image, while the current ellipse has its area in range, it can be consider as an object. This is because of the geometry ellipse fitting. It is explained in Chapter 3. Hence the area value must be delimited for a range of the area. Figure 4-11 shows the situation one a false detection is done.

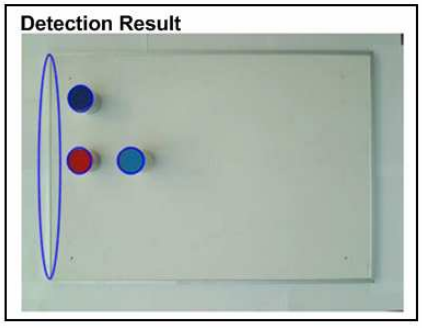


Figure 4-11: False detection due to the range of the area

In this case, when this problem occurred a contour processing for the current image was analyzed (see Figure 4-12), this was in order to see much clear the reason of why the false object appeared.
It is clearly visible that the contour which was considered as false object it is not exactly an ellipse. But for the ellipse fitting algorithm at certain moment, that contour can look like an ellipse.

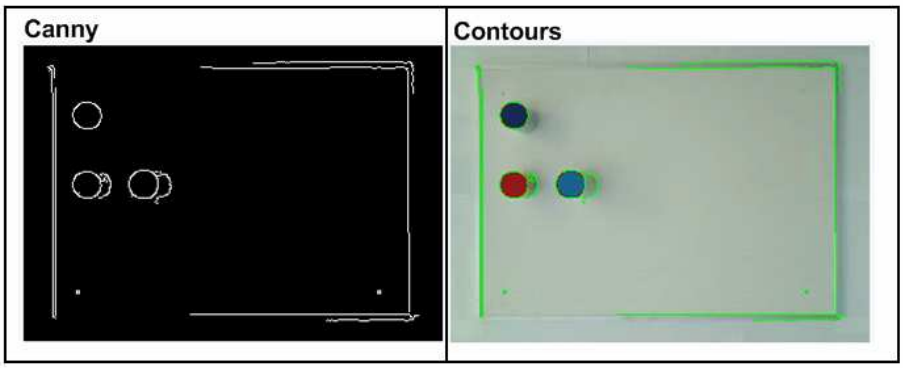


Figure 4-12: Contour processing of the false detection

Therefore, this problem is caused for another factor. It not only depends of the range of the area value. Also the intensity of the light is an influence for detecting objects, and also sometimes it produces false detections. When the area is delimited by a range, if one false object appears because of the light, it does not care because its area is out range respecting with the objects. The light is an influence for this method, principally when it is a natural light because it produces shadows. Hence the natural light must be controlled in order to detect the objects using the algorithm proposed. Figure 4-13 shows the impact of the light in the detection results. It is not possible to detect the objects correctly. Also this is a clear example about the area, because the Canny Edge Detection permits to see that the objects look like a bigger ellipses, therefore the ellipse fitting algorithm can be consider one object and its shadow as an ellipse, and then, if the total area of both together is in range, it is consider as a valid object.

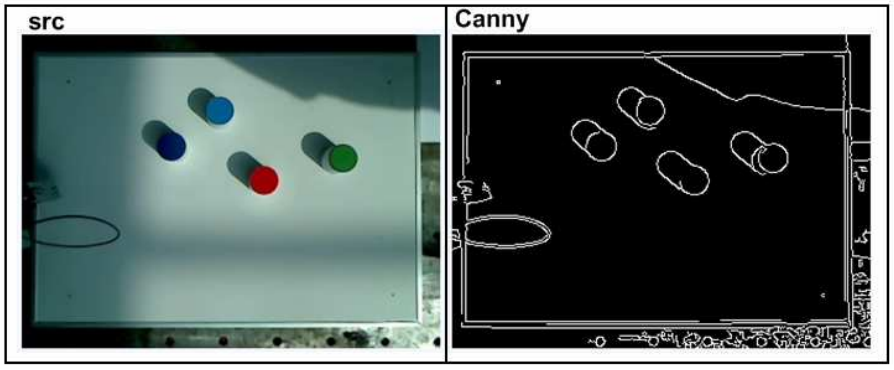


Figure 4-13: Shadows produced by the natural light

Figure 4-14 depicts the difference of the Canny Edge Detection results when two objects are located one in a region with high intensity of natural light (left side), and other object in a region with the natural light controlled (right side). It is clearly visible the high impact of the natural light in the results of the object detection algorithm proposed.

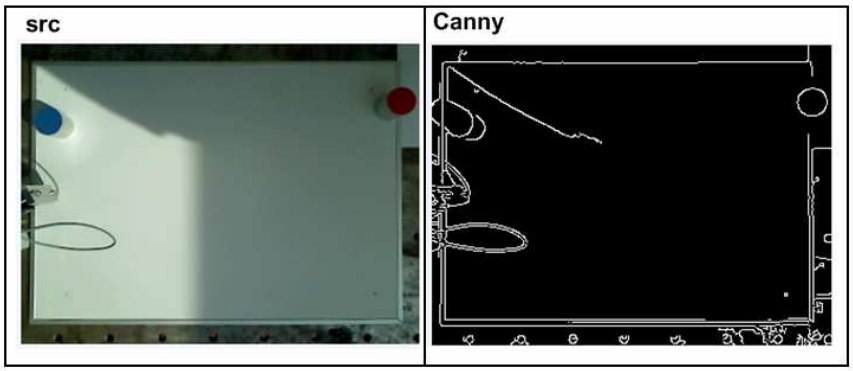


Figure 4-14: Difference of result detection according with the natural light

Thus, if the intensity of the natural light is controlled the results given by Canny Edge Detection they are much better, thereby the objects are correctly detected. Figure 4-15 illustrates when the contour processing is done with the quantity of the natural light controlled.

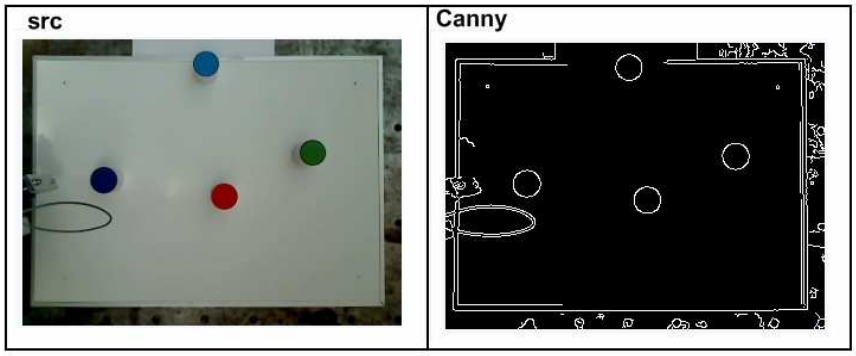


Figure 4-15: Intensity of the natural light controlled

4.3 Tracking requirements and constrains

Certain conditions, constrains and requirements principally for defining start and end position they are defined. Start position means, the object that it is going to be moved. End position means, the position where the object selected it is going to be moved.

The main requirement is about the correct location of the mark at the moment when a position is defined. The mark must be totally inside of the region of the object. Otherwise the position is defined as invalid position. Figure 4-16 illustrates how the mark must be located if one object is defined as start or end position.

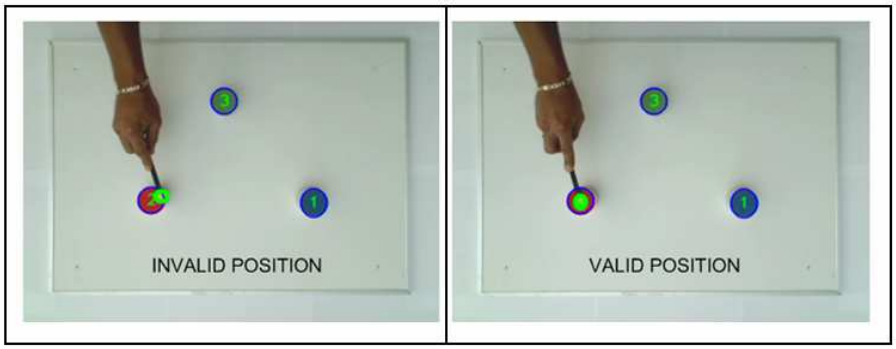


Figure 4-16: Requirement for defining position

Regarding the previous requirement, now the conditions and constrains of the tracking are presented.

In order to start to execute the tracking the mark must be searching. It is important to make allowance for not always at the first time when the mark is searching it is found. Sometimes the mark is found until the fifth time. However, at the mark is found, the tracking system does not have problems for tracking the mark.

When the position is valid it is loaded automatically, this depends of a certain time. For example, if the program is waiting for a start position and the mark is in valid position more that 4 seconds, the valid position is loaded because limit time has been completed. The user is able to define the limit time for this task.

Figure 4-17 depicts when the mark is searched, in that precise moment the object detection is stopped, this is in order to keep all the position of the objects.

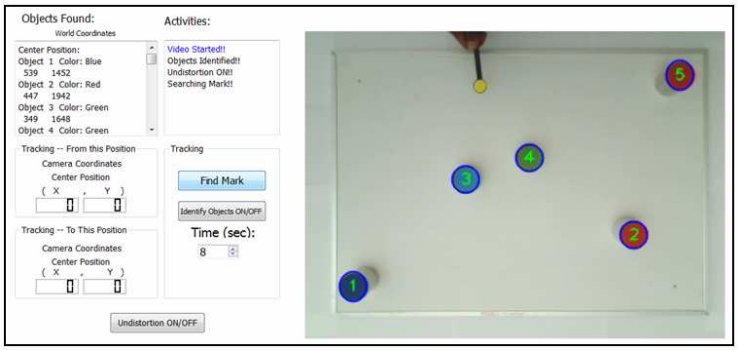


Figure 4-17: Searching Mark

At tracking start, the actual position of the mark is known and also it is shown in the GUI as it is illustrated in Figure 4-18, in which the red rectangle area shows the position of the mark, which is refreshed until the start position is defined. This is concerning with the start position.

Figure 4-18 also illustrates one of the conditions for defining start position. The position of the mark is defined as invalid because there is no object. Therefore one condition is about at least one object must be selected. Otherwise it does not make sense because the objective of the tracking is to manipulate the position of the objects.

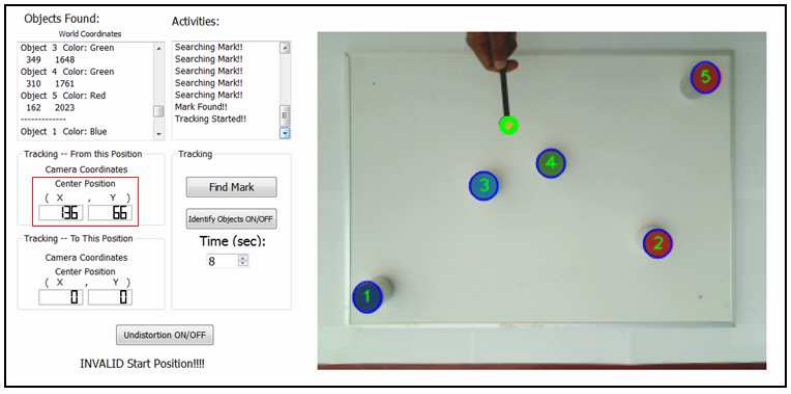


Figure 4-18: Mark found, and tracking has started. Mark in invalid position

There is other condition for start position, which is related with the minimum distance because of the robot. Thus a physical distance between the objects is defined, it is equal to 120mm from one object center to another object center. This condition is depicted in Figure 4-19, where object 3 and object 4 cannot be selected as start position because the distance between them is not the minimal required. The distance is checked every time because the actual position of the mark is known, it is checked until start position is defined.

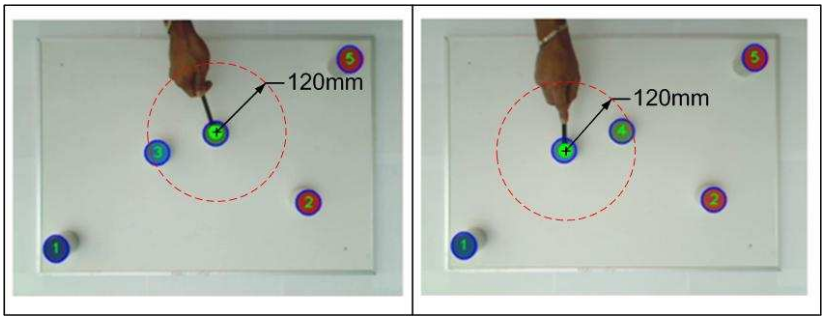


Figure 4-19: Invalid Start position of the mark because of the distance between the objects 3 and 4. Distance is measure from center mark to center object

Figure 4-20 shows the moment when the mark is still in a valid position, and then a loading bar appears. It shows the time left for loading the position, this depends of how many seconds the user has defined.

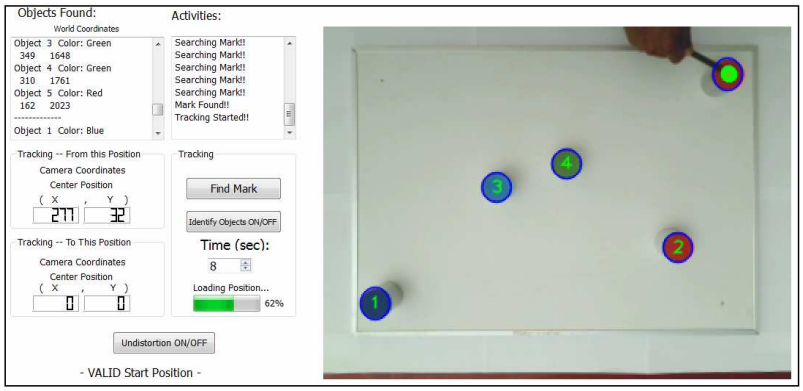


Figure 4-20: Mark is still in valid position, loading position

When the time is complete, the position loaded is equal to the center position of the center. The position of the mark just is used in order to know where the mark is, and with this to define and to decide if the mark is a valid position or not.

Automatically after load start position, the program is waiting for the second position or end position. For the second positions there are three conditions. The first is according of the second or end position cannot be the same object selected for first position. It is shown in Figure 4-21

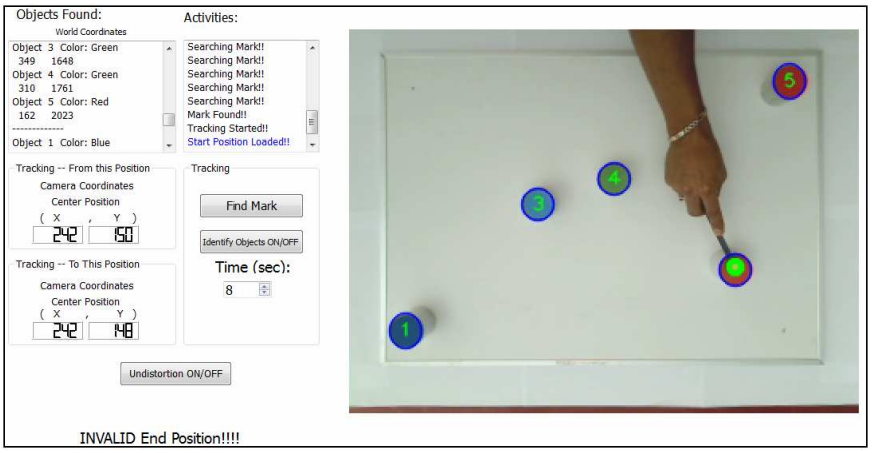


Figure 4-21: Start position loaded, now the program waiting for end position. End position cannot be the same as start position

The second and third condition is about the minimum distance, as is depicted in Figure 4-22. The difference between conditions of start position and end position it is that, for an end position is not required an object, it just need the minimum distance. Left side in Figure 4-21, the object has not problems with the distance related with

the other objects, and then the object 1 can be defined as end position, but in this case the center of the object is loaded, the mark just is a reference. In contrast in right side, the mark is not in an object, it does not matter because the center of the mark can be defined as end position, and just the minimal distance is required.

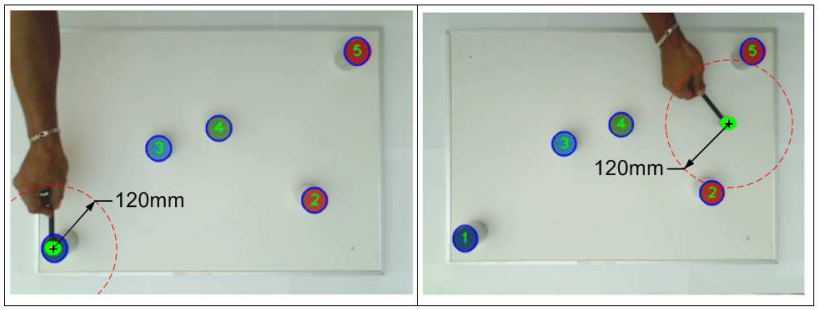


Figure 4-22: Valid end positions. Conditions required

About constrains of the tracking, the minimal distance in this case also depends of the characteristics of the system described in section 4.1.2 concerning with the physical distance between the camera and object plane. This is because if the distance change also the relation millimeters – pixels change. If the distance is changed, the minimal distance between objects must be recalculated with the new relation of the pixels and millimeters.At tracking running, the mark must not be moved very fast because the dimensions of the mark are small, therefore tracking at certain moment can lose the mark. When this occurs a small green point appears in the screen as it is illustrated in Figure 4-23 left side. If the mark is closer to this small point, the mark is found again and the tracking continues (see Figure 4-23 right side). In general this is not a problem, but is much better is the mark is moved slower.

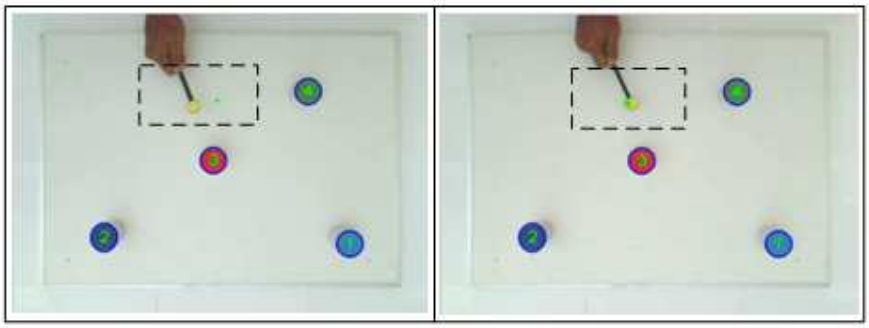


Figure 4-23: Mark lost because its movement was very fast. It can be found again

4.4 Instructing the Robot System

For the robot system a server for network communication was already available. The specific format of the commands needed was already defined. The system robot request the positions given by the detection and tracking. In the case of the objects, the colors also are requested. The connection to the network in order to sent the data

to the robot system it is through Internet Protocol Suite TCP/IP. It is named from two important protocols in it, which are the Transmission Control Protocol (TCP) and the Internet Protocol (IP). TCP is especially well suited for continuous transmission of data. Qt creator contains QTcpServer class, which provides a TCP-based server. This class makes it possible to accept incoming TCP connections. Therefore a host address must be defined. In this case is it is possible to connect to any host in the network through IPv4 any-address. It is equivalent to QHostAdress(“0.0.0.0”) in Qt creator. When a client connects to the server, the connection is established. It is possible to do in Qt creator through QTcpSocket, which is a convenience subclass of QAbstractSocket that allows establishing a TCP connection and transfer streams of data.

The object detection is to carry out in time intervals of 1sec. This is in order to refresh the information about the positions. At request data from a client, in this case the robot, the connection is established, afterwards the data are sent. The specific format for data from detection is shown in.

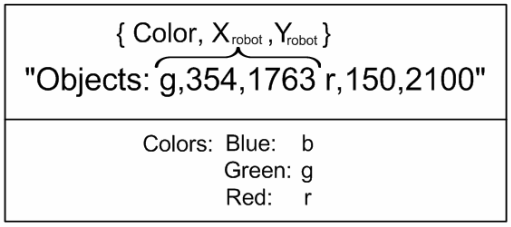


Figure 4-24: Format of detection data sent to the robot

When there are no valid objects, if the data is request from a client, only the word “Objects:” is sent. For the positions given by the tracking is almost the same format, the main difference is that not always the positions are known, only when the tracking is executed. At the moment when the tracking finished, the two positions are known and they can be sent. Therefore if the data from tracking is requested only the word “Positions:” is sent. Otherwise, the format is shown in Figure 4-25.

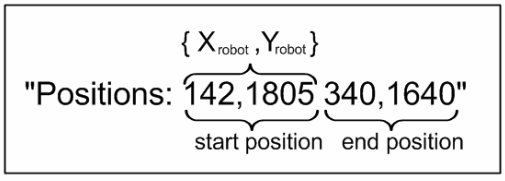


Figure 4-25: Format of positions of the tracking sent

5 Discussion of Results

In principle, the project is divided in three main parts. Each part has an algorithm designed for a specific task. The integration of them is the whole project presented. The method proposed for detecting object it is also used for detecting the mark used for tracking algorithm. In particular and especial case the camera calibration is very important in order to accurate the system response. When the calibration is executed, the parameters of it are extracted and saved in different files. Applying the parameters from the files the lens distortion is solved. Therefore, the calibration only can be executed once. Especially when the computer used for executing this application has limitations in terms of memory RAM and slow processor. In that case if the calibration is executed, the process for calculating the parameters of calibration takes several minutes. Hence using the values from the files this is not a problem. Even the User is able to execute the calibration in a powerful computer and obtain the calibration parameters. The tracking system can be executed in another computer with less technical properties using the calibration parameters from the files.

In this Chapter the discussion of the results of two methods is presented. According with the calibration method, its results are much clear when they are applied to detection method. Hence it is not dialed in a particular section. The tracking algorithm depends of the detection algorithm. After the mark is detected the tracking algorithm works itself. Thus if the method for detection has a better response, the results of tracking are going to be much better in order to find much faster the mark. The tracking algorithm offers by OpenCV library it is considered one of the best methods used for tracking, it only need to be initialized and then the mark is tracked successfully.

5.1 Object Detection Method

The method proposed for detecting objects it works based on the geometry of the objects, where the principal common characteristic is the size in terms of area.

The algorithm implemented is simple. The contour processing is a good option when the objects are the same in terms of geometry. OpenCV library provides functions for doing these tasks and they are powerful. Two possible options could be implemented for contour processing. They are threshold and Canny edge detection. The best option of them for this application was Canny edge detection which is based in a general method called feature synthesis. Regarding the theory behind of Canny algorithm which is explained in Chapter 3. Canny is distinguished from threshold function which is useful to remove noise from the image, it is only applying a single threshold whereas Canny permits to define two different values of threshold, an upper and lower. Hence the results from “cvThreshold()” function were not sufficient

for a good detection. For example when a single threshold is applied in an image, and at certain moment there is an edge in the image. Such that the response of the operator has mean value equal to threshold value defined in the function. There is going to be some fluctuation of the output amplitude due to noise, even if the noise is very slight. It is expected to be above threshold only about half the time. This leads to a broken edge contour [11]. Hence it is very difficult to set a single threshold value which must be able to give a good response. In this case perhaps it is possible to define a value for detecting contour. Because the value of the noise presented in the scene is not so much. Only the objects are present. But OpenCV library provides the "cvCanny()" function. Therefore it is a better option because uses adaptive thresholding with hysteresis to eliminate streaking of edge contours. Considering that the objective is to use the same algorithm for detecting the objects as well as the mark used for tracking.

For setting the correct values was not complicate because there is no much noise. The ratio recommended of high:low threshold is 2:1 and 3:1, for this proposed the values used are 3:1, that means 150:50. The problem presented in this stage of the detection algorithm was related with the intensity of the natural light. As already mentioned in chapter 4, when the detection is tested in an environment which has a strong illumination of natural light, the Canny algorithm does not work as it was expected.

This major problem was unsolved, different values were used for threshold in Canny algorithm but it was not possible to find the correct values. In order to show the results for this especial situation of the detection, four different values were implemented (see Figure 5-1 and Figure 5-2). Therefore Canny algorithm was a good option for finding contours, but regarding one important condition, the intensity of the light, in particular the natural light must be controlled.

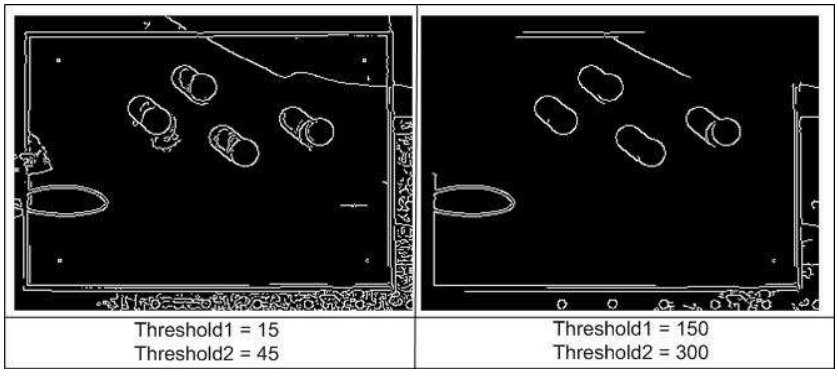


Figure 5-1: Result given by "cvCanny()", ratio 3:1, 3:1

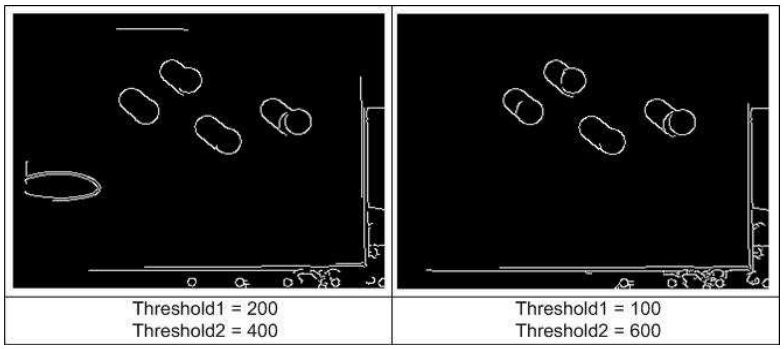


Figure 5-2: Results given by “cvCanny()”, ratio 2:1, 6:1

The ellipse fitting also was a good option for this task. The objects are not ellipses, but because of camera view and the different perspectives, the best approximation is an ellipse. Ellipse fitting doest not know if the ellipse found corresponds to a valid object or not. That was one of the most significant problems at detection. The definition of valid objects is according with the area value, but if the physical distance between the camera and the object plane change, the area change as well, if the value area is out range, the object is not detected. Only under this condition an object cannot be detected. This also involves the response of the system. The robot is not able to take an object over another object because the coordinates are shifted. This also is shown in Chapter 4. Concerning with the calibration algorithm, the detection method is able to detect the objects with lens distortion. After calibration the accuracy of the system in much better, it changes drastically. Due to the undistortion, the response of the detection algorithm, regarding the error value calculated is considered as a linear response.

Therefore the method proposed presents some limitations and especial conditions as well. Even though, regarding all of them the objects are detected. In general the method for detecting, which is based on the geometry of the objects is good in terms of the requirements predefined of the whole project.

5.2 Tracking Method

The method proposed is based on CamShift algorithm from OpenCV library, which is based on color distribution. Considering that the tracking is the most important part of the project, it is important remark these characteristics.

In the tracking algorithm proposed and shown in Chapter 3 contains the technique employed for getting the color of each object. This is because when the tracking was developed, the functions of the histograms provided by OpenCV are using for calculating the color distribution that CamShift algorithm needs. Therefore histograms were a very good option for getting the color of each object. The restriction of this method is that it is able to detect only the colors blue, red and green. However, the

objects used for this application are red, green and blue color. According with the requirements of the application the technique used for getting the color is good.

Due to the CamShift algorithm provides for OpenCV is a powerful method, which has different important characteristics: it is able to scale its search window to object size naturally handling perspective-induced motion irregularities. Regards with the search windows, the problem presented in the experimentation stage related with Chapter 4 it is that the search window must be defined. This definition is done when the mark is found. Therefore the search window depends directly of the detection method. Hence the mark has the same geometry characteristic as the objects. But sometimes the mark in not found at the first time, it is necessary to tray to find the mark several times. This is due to the detection. The dimensions of the mark are smaller than the objects, therefore related with the detection algorithm it is suitable to use different values of the threshold in order to detect the mark faster.

According with the color model used by CamShift algorithm is able to eliminate much of the noise. Thereby CamShift tends to ignore the remaining outliers. Also because of the color distribution CamShift is able to ignore objects outside its search window or possible distractors such as nearby objects or even the hand of the User. They do not affect the tracking of CamShift. Hence it is possible to achieve the selection of the objects. The results obtained at the moment when an object is selected as start position or end position they were successful, the color of the objects do not affect the tracking. The characteristics can be considered as advantages of the CamShift algorithm. The tracking is able to work at any level of physical distance related with the camera, but the main problem is the detection algorithm, because it is not able to work at any level. Also the useful advantage which was possible to see and to analyze it is about the modes of freedom of CamShift. It is able to detect four modes of degrees of freedom. These are X, Y Z and roll. The roll is a powerful advantage of the algorithm because was possible to see what occurred when the orientation of the mark was not totally in the horizontal plane in relation with the camera.

When the color of the mark was defined, in the first time it was defined with green color, but at the moment when the mark was closer to a green object the response of the tracking was resized the search window because if the distribution color increased. Hence the color of the mark had to be changed to a total different color in comparison with the color ob the objects. Thus it was possible to understand the behavior of the CamShift algorithm, and also one restriction of the algorithm. CamShift is on based in a unique feature that is the color, therefore errors in color such as colored lighting, dim illumination, and high intensity of light. These errors can cause errors in tracking. Regarding the limitations of CamShift algorithm the results were the expected after final definition of the mark.

6 Conclusions and Future Prospects

In this thesis the integration of OpenCV library in Qt creator was presented. It could be shown that OpenCV library is a good option for image processing. Even though, it was demonstrated that OpenCV library does not have high quality in terms of resolution for capturing video. It was an influence at the moment when the detection area was defined. It was also a problem because it was difficult to see the events occurred at object detection. Qt creator was very helpful for this problem because it was possible to present a solution doing a zoom to the captured video. Qt designer was also a good option for this thesis. It permitted to design a complete graphical user interface for this project.

Related with camera calibration algorithm, the main problem was about the lens distortion. It was very necessary to solved because of the detection. The distortion problem was eliminated using the functions provides for OpenCV library. The number of samples used for calibration were enough in order to accurate the camera for this application. It could be shown that is not necessary recalibrated the camera if the physical distance between the camera and the object plane is changed. Hence the parameters of the calibration are saved in files and they can be used for the next time when the program is executed. This was very helpful when the program was executed in a computer with technical limitations, because the calibration takes several minutes for calculating the parameters. The most interesting unsolved problem related with camera calibration is the transformation from camera coordinates to world coordinates. It could be a possible extension of the calibration. When the scale factors are calculated for one level for the objects the system works very well, but if the distance is changed the scale factors changed as well. Therefore the calibration algorithm can be modified in order to obtain the scalar factors related with the size of the pixels.

The object detection method was based on geometry of the objects, where ellipse fitting was demonstrated as a good approximation for this task. Contour processing was used thinking about of the mark used for the tracking. It was demonstrated that through contour processing it was possible to detect just the mark when it was required. It could be shown for finding contours the difference between to apply one single value of threshold or two different values. Thereby the edge detection was much better, and the contour processing as well. Canny algorithm also could be shown as a practical good option because of the threshold hysteresis. The objects and the mark could be detected successfully. The disadvantage presented for the method was related with the intensity of the natural light. It was possible to see the behavior of the Canny edge detection when in the scene there was a strong intensity of the natural light. The detection did not work because of the shadows. A solution

was to control the intensity of the light. The major problem with detection is when one object is located over another one. In this case also the calibration was involves. A compensation for detection is needed. To obtain a relation between the shifted produced in the X and Y axes, it could be possible to know the new location of the object.

The tracking system method proposed is based on one of the best methods for tracking. CamShift algorithm provides from OpenCV library was implemented. It could be demonstrated all the advantages that this algorithm offers. It is based on color distribution as feature. It was possible to see what occurred when another color was closer of the mark at tracking. It was shown that it did not affect to the tracking. Another advantage was about the resize of the search window. In general the optical tracking could be developed. The problem presented concerning with the tracking, it was about the detection of the mark. The method used for detecting the mark was the same as well as for objects. But sometimes for detecting the mark in the scene it took several times, therefore id the detection of the mark, or even better, if the detection algorithm is restructured in order to found faster the mark, the optical tracking is going to work much better. Another problem is for starting the tracking. The user must press a button. In order to find the mark automatically when it is in the scene could be solved through to implement a detection of features, for instances the convexity defects. The hand could be detected at the moment when it is scene, after that the mark could be searched. In general the complete system presented is able to achieve handling of objects and pick & place operations with optical tracking.

7 References

[1] Open Source Computer Vision Library, “Manual Reference”, Copyright © Intel Corporation, 1999 – 2001, United States of America.

[2] G. Bradski, A. Kaebler, “Learning OpenCV Computer Vison with OpenCV library”, O’Reilly, First Edition, 2008.

[3] R. Y. Tsai, “A versatile camera calibration technique for high accuracy 3D machine vision metrology using off -the-shelf TV cameras and lenses,” IEEE Journal of Robotics and Automation 3 (1987): 323–344.

[4] K. M. Dawson-Home and D. Vernon, “Simple Pinhole Camera Calibration”, Department of Computer Science, Trinity College, Dublin, Ireland.

[5] Z. Zhang, “Flexible camera calibration by viewing a plane from unknown orientations”, Proceedings of the 7th International Conference on Computer Vision (pp. 666–673), Corfu, September 1999.

[6] R. Hartley and A. Zisserman, “Multiple View Geometry in Computer Vision, Cambridge”, Second Edition, UK: Cambridge University Press, 2006.

[7] P. F. Sturm and S. J. Maybank, “On plane-based camera calibration: A general algorithm, singularities, applications,” IEEE Conference on Computer Vision and Pattern Recognition, 1999.

[8] J. Heikkila and O. Silven, “A four-step camera calibration procedure with implicit image correction,” Proceedings of the 1997 Conference on Computer Vision and Pattern Recognition (p. 1106), 1997.

[9] Z. Chum, T. D. Long, and Z. Feng, “A High-Precision Calibration Method for Distorted Camera”, Robotics Laboratory, Sbenyang Institute of Automation, Proceedings 01 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems. September 28 - October 2, 2004, Sendai, Japan.

[10] F. Remondino, C. Fraser, “Digital Camera Calibration Method: Considerations and Comparisions”, ISPRS Commision V Symposium “Image Engineering and Vision Metrology”, IAPRS Volume XXXVI, Part 5, Dresden 25-27 September 2006.

[11] J. Canny, “A Computational Approach of Edge Detection”, IEEE Trans Pattern Analysis and Machine Intelligence, 8(6): 679-698, Nov 1986.

[12] Zhengyou Zhang. “Parameter Estimation Techniques: A Tutorial with Application to Conic Fitting”, Image and Vision Computing Journal, 1996.

[13] W. T. Freeman and M. Roth, “Orientation histograms for hand gesture recognition,” International Workshop on Automatic Face and Gesture Recognition (pp. 296–301), June 1995.

[14] G. Bradski, “Computer vision face tracking for use in a perceptual user interface”, Intel Technology Journal Q2 (1998): 705–740.

[15] A. R. J. François, “CAMSHIFT Tracker Design Experiments with Intel OpenCV and SAI”, Institute for Robotics and Intelligent Systems University of Southern California, July 2004.

[16] Y.Rubner, C. Tomasi and L. J. Guibas, “A Metric for Distributions with Applications to Image Databases”, Computer Science Department, Stanford, University Stanford, CA 94305, 1998.

[17] D. C. Brown, “Close-range camera calibration,” Photogrammetric Engineering 37 (1971): 855–866.

[18] J.G. Fryer, T.A. Clarke, and J. Chen, “Lens distortion for simple C-Mount lenses”, International Archives of Photogrammetry and Remote Sensing, 30(5):97-101.

[19] R. Hartley and A. Zisserman, “Multiple View Geometry in Computer Vision, Cambridge”, Second Edition, UK: Cambridge University Press, 2006.

[20] Bjarne Stroustrup, The C++ Programming Language, 13th ed. Addison-Wesley, 2006.

[21] T. Siepmann, “OO-Design and UML”, Computer Science, Lectures, Fachhochschule Aachen – Aachen University of Applied Sciencie, 2008.

[22] A.J. Blauch, P.D.Johnson, “Structure Designed using FlowChart”, C code implementation, rev. 2.0, Grand Valley State University, 2001.

[23] K. Dorfmüller-Ulhaas, “Optical tracking from user motion to 3D interaction”, Vienna University of Technology, Octuber, 2002.

Internet Resources

http://www.qtsoftware.com/products/developer-tools

http://www.qtsoftware.com/downloads

http://opencv.willowgarage.com/wiki/

http://opencv.willowgarage.com/documentation/

http://sourceforge.net/projects/opencvlibrary/files/opencv-win/

http://www.logitech.com/index.cfm/webcam_communications/webcams/devices/3056&cl=GB,EN

8 Appendix

Type of variables	
Name	Description
CvMemStorage	Memory storage is a low-level structure used to store dynamically growing data structures such as sequences, contours, graphs, subdivisions.
CvSeq	This structure is a base for all of OpenCV dynamic data structures. They are used to represent growable 1d arrays - vectors, stacks, queues, and deques.

CvPoint
CvPoint2D32f
	2D point with integer coordinates (usually zero-based). Its structure is:
	typedef struct CvPoint
	{
	int x;
	int y;
	}
	CvPoint;
	The difference between both points, it is that the second one its values are float type.

CvHistogram	typedef struct CvHistogram
	{
	int type;
	CvArr* bins;
	float thresh[CV_MAX_DIM][2]; // for uniform histograms
	float** thresh2; // for nonuniform histograms
	CvMatND mat; // embedded matrix header
	// for array histograms
	}
	CvHistogram;

CvConnectedComp	typedef struct CvConnectedComp { double area; /* area of the segmented component */ CvScalar value; /* average color of the connected component*/ CvRect rect; /* ROI of the segmented component */ CvSeq* contour; /* optional component boundary (the contour might have child contours corresponding to the holes) */ } CvConnectedComp;
CvMat	There is no “vector” construct in OpenCV. If one vector wants to be declared, it must be a matrix with one column. Example of two dimensional matrix: cvMat* cvCreateMat (int rows, int cols, int type) General structure: typedef struct CvMat { int type; int step;

	int* refcount; // for internal use only
	union {
	uchar* ptr;
	short* s;
	int* i;
	float* fl;
	double* db;
	} data;
	union {
	int rows;
	int height;
	};
	union {
	int cols;
	int width;
	};
	} CvMat;

Table 3: Type of variables from OpenCV Library [1][2]

Functions	
Name	Description
CvCapture* cvCreateCameraCapture(int index)	Initializes capturing a video from a camera. Parameter: index – Index of the camera to be used. If there is only one camera or it does not matter what camera is used -1 may be passed.
void cvCanny(const CvArr* image, CvArr* edges, double threshold1, double threshold2, int aperture_size=3)	It finds the edges on the input image image and marks them in the output image edges using the Canny algorithm.
ouble cvThreshold(const CvArr* src, CvArr* dst, double threshold, double max_value, int threshold_type)	It applies fixed-level thresholding to a single-channel array. The function is typically used to get a bi-level (binary) image out of a grayscale image or for removing a noise, i.e. filtering out pixels with too small or too large values.
int cvFindContours(CvArr* image, CvMemStorage* storage, CvSeq** first_contour, int header_size=sizeof(CvContour), int mode=CV_RETR_LIST, int method=CV_CHAIN_APPROX_SIMPLE, CvPoint offset=cvPoint(0, 0))	It retrieves contours from the binary image and returns the number of retrieved contours. The pointer first_contour is filled by the function.
void cvEllipse(CvArr* img, CvPoint center, CvSize	It draws a simple or thick elliptic arc or fills an ellipse sector. The arc is clipped by the

axes, double angle, double start_angle, double end_angle, CvScalar color, int thickness=1, int line_type=8, int shift=0)	ROI rectangle. All angles are given in degrees.
void cvReleaseCapture(CvCapture** capture) void cvReleaseImage(IplImage** image) void cvReleaseMemStorage(CvMemStorage** storage) void cvReleaseMat(CvMat** mat) void cvReleaseHist(CvHist** hist)	These functions are very important, especially after the corresponding variables are used, because sometimes if these are not applied an overflow in memory occurs.
CvHistogram* cvCreateHist(int dims, int* sizes, int type, float** ranges=NULL, int uniform=1)	It creates a histogram of the specified size and returns a pointer to the created histogram. If the array ranges is 0, the histogram bin ranges must be specified later via the function cvSetHistBinRanges().
void cvCalcBackProject(IplImage** image, CvArr* back_project, const CvHistogram* hist)	It calculates the back project of the histogram. For each tuple of pixels at the same position of all input single-channel images the function puts the value of the histogram bin, corresponding to the tuple in the destination image.
int cvCamShift(const CvArr* prob_image, CvRect window, CvTermCriteria criteria, CvConnectedComp* comp, CvBox2D* box=NULL) Parameters: * prob_image – Back projection of object histogram * window – Initial search window * criteria – Criteria applied to determine when the window search should be finished *comp – Resultant structure that contains the converged search window coordinates (comp->rect field) and the sum of all of the pixels inside the window (comp->area field) * box – Circumscribed box for the object. If not NULL,it contains object size and orientation	Its Finds the object center, size, and orientation. The function cvCamShift() implements the CAMSHIFT object tracking algrorithm. First, it finds an object center using cvMeanShift() and, after that, calculates the object size and orientation. The function returns number of iterations made within cvMeanShift(). The CvCamShiftTracker class declared in cv.hpp implements the color object tracker that uses the function.

Table 4: Functions from OpenCV Library [1] [2]

System Requirements Webcam
Windows® XP
• Pentium® P4 (or compatible) processor, 1.4 GHz (2.4 GHz recommended) • 128 MB RAM (256 MB recommended) • 200 MB available hard disk space • 16-bit color display adapter • Windows®-compatible sound card and speakers (full-duplex sound card recommended) • USB port • CD-ROM drive
Windows Vista™
• Pentium® P4 (or compatible) processor, 2.4 GHz (2.8 GHz recommended) • 512 MB RAM (1 GB recommended) • 200 MB available hard disk space • 16-bit color display adapter • Windows®-compatible sound card and speakers (full-duplex sound card recommended) • USB port • CD-ROM drive
Technical Specifications
• Carl Zeiss® lens • Autofocus system • Ultra-high resolution 2-megapixel sensor with RightLight™2 Technology • Color depth: 24-bit true color • Video capture: Up to 1600 x 1200 pixels (HD quality) (HD Video 960 x 720 pixels) • Frame rate: Up to 30 frames per second • Still image capture: 8 million pixels (with software enhancement) • Built-in microphone with RightSound™ Technology
Note: System recommendations apply to use of Logitech® RightLight™2, RightSound™, Video Effects™, or Fun Filters.

Table 5: System requirements of Webcam

SV3X SPECIFICATIONS		
Structure		Vertical jointed-arm type
Controlled Axes		6
Payload		3 kg (6.6 lbs.)
Vertical Reach		1,019 mm (40.1")
Horizontal Reach		677 mm (26.7")
Repeatability		±0.03 mm (±0.001")
Maximum Motion Range	S-Axis (Turning/sweep) L-Axis (Lower Arm) U-Axis (Upper Arm) R-Axis (Wrist Roll) B-Axis (Bend/Pitch/Yaw) T-Axis (Wrist Twist)	±170° +150°/-45° +190°/-70 ±180° ±135° ±350°
Maximum Speed	S-Axis L-Axis U-Axis R-Axis B-Axis T-Axis	210°/s 170°/s 225°/s 300°/s 300°/s 420°/s
Approximate Mass		30 kg (66.2 lbs.)
Brakes		All axes
Power Consumption		1 kVA
Allowable Moment	R-Axis B-Axis T-Axis	5.39 N • m 5.39 N • m 2.94 N • m
Allowance Moment of Inertia	R-Axis B-Axis T-Axis	0.1 kg • m2 0.1 kg • m2 0.03 kg • m2

Table 6: SV3X specifications

XCR 2201 Controller Specifications	
Structure	Free-standing, enclosed type
Dimensions (mm)	750 (w) x 860 (h) x 550 (d) (29.5" x 33.9" x 21.7")
Approximate Mass	70 kg (154.4 lbs.)
Cooling System	Indirect cooling
Ambient Temperature	During operation: 0°C (32°F) to +45°C (113°F) During transmit and storage: -10°C (14°F) to +60° C (140°F)
Relative Humidity	90% max. non-condensing
Primary Power Requirements	3-phase, 200/220 VAC (+10% to -15%) at 50/60 Hz
Grounding	Grounding resistance: ≤100 ohms Separate ground required
	Separate ground required
Digital I/O	Specialized signals (hardware): 12 inputs/3 outputs General signals (standard max): 40 inputs/40 outputs Expandable to 256 inputs/256 outputs
Position Feedback	By absolute encoder
Drive Units	Servo packs for AC servomotors
Accel/Decel	Software servo control
Program Memory	5,000 steps and 3,000 instructions

Pendant Dim. (mm)	200 (w) x 325 (h) x 77 (d) (7.9" x 12.8" x 3.0")
Pendant Buttons Provided	Teach Play, Remote, Servo On, Start, Hold, Emergency Stop, Edit Lock
Safety	Emergency Stop Pushbuttons, 3-position Deadman, Brake release switches Meets ANSI/RIA R15.06-1999 standard

Table 7: XCR 2201 Controller Specifications

List of Figures

List of Tables