School of Telecommunications Engineering

Master in Telematic and Computer Systems

MASTER THESIS

Visual Tracking of Multiple Objects in Cluttered Environments

Author: Luis Menéndez García
Tutor: Prof. Dr. José María Cañas Plaza

Course 2013/2014

Contents

List of Figures

List of Algorithms

Abstract

Generally speaking a tracking system can be defined as any kind of system able to effectively predicts its state over the time, given any initial data and having feedback about its real state from time to time. In this way there are a myriad of tracking systems, used to predict economical variables, the water reservoir levels, etc. Nowadays there is another interest area in those tracking systems able to effectively track visual objects from images. These systems are widely used in biomedical applications, video surveillance solutions, traffic control, sports, etc.

The aim of this master thesis consists in showing, with a practical approach, different techniques to effectively track multiple objects inside images in the presence of measurement origin uncertainty. The basic Kalman Filter (aka KF) is exposed to continue with an explanation to more powerful approaches that take into consideration cluttered environments, such as Probabilistic Data Association Filter (aka PDAF) and Join Probabilistic Data Association Filter (aka JPDAF). To better understand how these algorithms work, a C++ framework was developed. The framework implements a simulator where the KF, PDAF and JPDAF can be tested with simulated targets and clutter. Another module allows to use real images that can be processed with color filter techniques to isolate the objects to be tracked and also a background subtraction technique to better identify objects that are moving in the real world.

Chapter 1

Introduction

1.1 Computer Vision

Computer Vision is a vast field. A possible definition comprises the methods for acquiring, processing, analyzing, and understanding images from the real world in order to produce any kind of information or a decision. Therefore, Computer Vision is concerned with the theory behind artificial systems that extract information from images.

The image data can take many forms, such as video sequences, views from multiple cameras, or multi-dimensional data from a medical scanner. After the images are processed the resulting information can be used by visual representation systems or by the navigation system in a robotic device, for instance.

Generally speaking all the computer vision systems are composed by two basic elements: the image acquisition system and the image processing system. The former comprises the illumination, image getting and signal acquisition systems. The latter is built by the vision algorithms that transform the images and extract the information from the raw images.

- *Illumination System:* are composed by all the elements that light up the objects by any kind of electromagnetic radiation. Examples of these artifacts are the natural light radiation system such as the Sun, or the artificial light radiation systems such as the lamps or the lasers.

- *Image Getting System:* transforms the reflected radiation into electrical signals. The most widely used systems are the cameras for visible and invisible spectrum.

- *Signal Acquisition System:* The electrical signals coming from the cameras build the video signals. Its main objective is putting the video signal inside the computer data bus that will be used for the signal processing phase.

- *Processing System:* It is a computer or a specific device inside it that implements the algorithms needed to process the digital images and to elaborate the information required by the purpose of the computer vision system.

  The processor system can comprise the following phases:

  Pre-process: During this phase the image can be adapted to better extract the required information according to the algorithms and methods used. The main objective of this phase is to obtain a better quality of the incoming image, using techniques such as denoising, deconvolution, enhancements and others.

  Segmentation: In this phase the image is divided into interest areas. For instance differencing spherical from square objects, or selecting the lanes of the highway from the rest of the image. Different techniques can be used for this purpose: thresholds, discontinuities, regions grow, color filter, movement detection, etc.

  Classification: Once the image have been divided by regions of interest (ROIs), it is time to extract the specific features of each one. This can be done by morphological analysis, by texture or using color classification techniques.

- *Peripheral System:* Comprises the information receptor elements, including monitors, devices that use the information for taking decisions purposes, etc.

Nowadays, the applications of computer vision are growing very quickly, due the availability of inexpensive hardware able to successfully run vision algorithms and the popularization of artificial vision algorithms. For instance, we can find quality control applications for testing several manufactured products, parts identification, labeling, food quality control, etc. In robotics the typical applications we can find are welding automation and control, and those with regards of unmanned vehicles. In biosciences the artificial vision can help in image analysis (cancer, evolution of degenerative diseases, etc). In astronomy it can help to compose a more quality images, etc.

One of the most popular applications for artificial vision is the Optical Character Recognition, also known as OCR. The purpose of these systems are the identification of printed characters (in a business card, a vehicle plate or similar) to have a posterior process of that information.

A popular uses of this technology is the automatic number plate recognition (see figure 1.1). Despite it was invented in 1976 at the Police Scientific Development Branch in the UK, the popularization of this technology was during 1990s. The main applications are detecting stolen cars, detecting cars without insurance validity, traffic control and automatic garage doors.

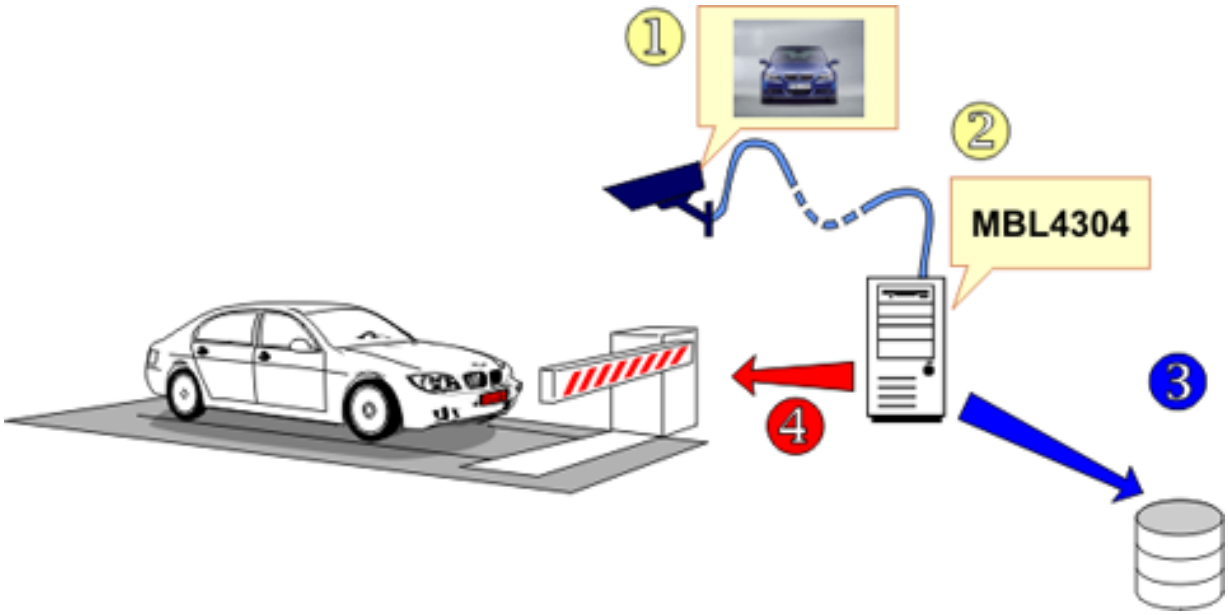


Figure 1.1: Automatic plate number recognition system

The biometrics systems are those able to automatically recognize individuals based on their biological and behavioral characteristics. These characteristics can include:

- *Ears Recognition:* the identification of an individual using the shape of the ear.
- *Eyes - Iris Recognition:* the use of the features found in the iris to identify an individual.
- *Eyes - Retina Recognition:* the use of patterns of veins in the back of the eye to accomplish recognition.
- *Face Recognition:* the analysis of facial features or patterns for the authentication or recognition of an individuals identity. Most face recognition systems either use eigenfaces or local feature analysis.
- *Fingerprint Recognition:* the use of the ridges and valleys found on the surface tips of a human finger to identify an individual.
- *Finger Geometry Recognition:* the use of 3D geometry of the finger to determine identity.
- *Gait Behavioural Biometric:* the use of an individuals walking style or gait to determine identity.
- *Hand Geometry Recognition:* the use of the geometric features of the hand such as the lengths of fingers and the width of the hand to identify an individual.
- *Vein Recognition:* is a type of biometrics that can be used to identify individuals based on the vein patterns in the human finger or palm, as it can seen in figure 1.2.

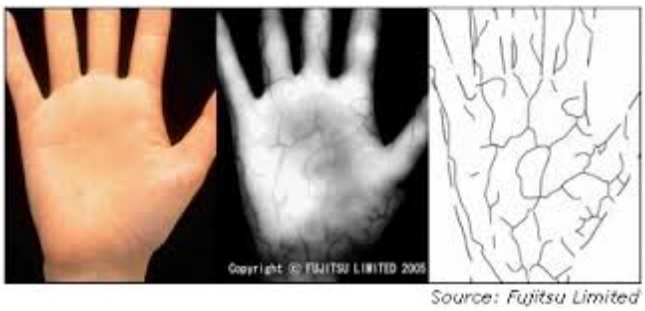


Figure 1.2: Vein recognition biometric system

Another fast growing area are the surveillance systems. We can find them in any critical infrastructure we already use: in airports, governmental buildings access, etc. These systems can be used for personal training purposes or for automatic identification of potential risks, as it can seen in figure 1.3

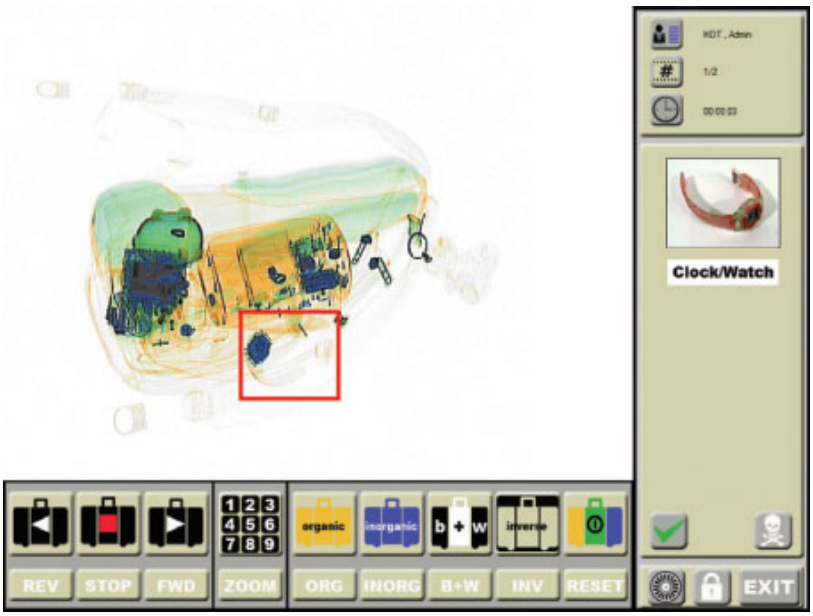


Figure 1.3: Baggage scanner

With regards of automatic threats detection we can find surveillance systems able to automatically detect and alert security forces in case a threat was discovered, as shows in figure 1.4, where the system has identified a group of persons fighting, enclosing them into a red square.

Figure 1.4: Automatic threat detection surveillance system

Figure 1.5: Haweye tennis tracking

Hawk-Eye is a computer system used officially in numerous sports such as cricket, tennis, Gaelic football, hurling and association football, to visually track the trajectory of the ball (see figure 1.5) and display a record of its statistically most likely path as a moving image. Hawk-Eye was developed in the United Kingdom by Dr Paul Hawkins. The system was originally implemented in 2001 for television purposes in cricket. The system works via six (sometimes seven) high-performance cameras, normally positioned on the underside of the stadium roof, which track the ball from different angles. The video from the six cameras is then triangulated and combined to create a three-dimensional representation of the trajectory of the ball. Hawk-Eye is not infallible and is accurate to within 3.6 millimeters but is generally trusted as an impartial second opinion in sports.

1.2 Visual Object Tracking

Video tracking is the process, inside computer vision, of locating a moving object (or multiple objects) over time using cameras. It has a variety of uses, some of which are: human-computer interaction, security and surveillance, video communication, augmented reality, traffic control, medical imaging and video editing.

The objective of video tracking is to associate target objects in consecutive video frames. The association can be especially difficult when the objects are moving fast relative to the frame rate. Another situation that increases the complexity of the problem is when the tracked object changes orientation over time. For these situations video tracking systems usually employs a motion model that describes how the image of the target might change for different possible motions of the object over time.

An algorithm that analyzes sequential video frames and follows the movement of targets between frames it is needed to perform video tracking. There are two major components

of a visual tracking system: first a target representation and localization algorithm, and second a filtering and data association technique.

Target representation and localization methods gives a variety of tools for identifying the moving object. Locating the target object successfully depends on the algorithm used.

The following are some common target detection techniques:

- *Segmentation based in threshold* transforms the image into gray scale to isolate the foreground objects (multi band threshold, semi band threshold, adaptive threshold, etc)
- *Segmentation based in border detection* searches for gray level discontinuity: isolate points, lines detection, image gradient, Laplace of Gaussian (LoG), Canny, borders union, Hough, etc.
- *Segmentation based on regions* builds the regions of interest directly: growing or union, dividing, split and merge, morphological segmentation, color based segmentation, active contours (snakes), move based techniques (background subtraction, image differences, ADP, etc.)
- *Visual feature matching:* the process of transforming different sets of data into other domains like: intensity-based vs feature-based, transformation models, spatial vs frequency domain methods, single vs multi-modality methods, automatic vs interactive methods, similarity measures for image registration. Examples of this are *Scale-invariant feature transform (SIFT)* for extracting the feature description of an object in order to match it within an objects description database, or *Speeded Up Robust Features (SUFR)* partly inspired by SIFT algorithm than claims to be more robust and quickly than SIFT.

Filtering and data association involves incorporating prior information about the scene or object, dealing with object dynamics, and evaluation of different hypotheses.

There are several ways to track objects using deterministic methods:

- *Blob tracking:* segmentation of objects (blob detection, block-based correlation or optical flow).
- *kernel-based tracking:* an iterative procedure based on the maximization of a similarity measure.

- *Contour tracking:* detection of object boundaries (active contours).

These methods allow the tracking of complex objects along with more complex object interaction like tracking objects moving behind occlusions. Common probabilistic filtering algorithms description follows:

- *Mean Shift* create a confidence map (probability density function) in the new image based on the color histogram of the object in the previous image, and use mean shift to find the peak of a confidence map near the object's old position. A few algorithms, such as ensemble tracking and CAMshift, expand on this idea.
- *Kalman filter:* an optimal recursive Bayesian filter for linear functions subjected to Gaussian noise.
- *Probabilistic Data Association Filter:* Kalman Filter derivative used for tracking a single object in cluttered environments.
- *Join Probabilistic Data Association Filter:* Probabilistic Data Association Filter used for tracking multiple objects supporting presence of noise.
- *Particle Filter:* useful for sampling the underlaying state-space distribution of non linear and non-Gaussian process.

There are multiple applications for video tracking systems, widely used both in civilian and military applications.

- *Traffic Control:* nowadays traffic control systems are widely distributed not only to classify and control the traffic in highways but also to check how the cars and pedestrians are distributed in little towns too. A possible use of these systems resides in counting and classifying the different types of vehicles in a high way, giving information about the traffic jams, the most occupied lanes and so on, as shown in figure 1.8. These systems can also help different policy departments to quickly locate stolen cars, or to issue traffic tickets when a certain vehicle has ridden over a restricted area.
- *Military applications:* another possible use, apart from civilian applications, consists in target location and tracking for defense systems. In the image 1.6 we can see how a fighter is tracked. In this particular case the defense system uses a visual tracking method that supports occlusions and false positives, using statistical filters.

Figure 1.6: Visual fighter tracking example implemented in this master thesis

1.3 Visual Tracking projects in URJC

The Robotics Group of University King Juan Carlos (URJC - Universidad Rey Juan Carlos) has developed a framework named JdeRobot for programing applications in robotics, computer vision, home automation and scenarios with sensors, actuators and intelligent software.

JdeRobot is open-source software, licensed as GPL and LGPL. It also uses third-party software like Gazebo simulator, OpenGL, GTK, Qt, Player, Stage, Gazebo, GSL, OpenCV, PCL, Eigen, Ogre, etc.

The following are an example or projects developed inside the JdeRobot framework.

- *UAV* [7]

  The Unmanned Aerial Vehicle (UAV) is a growing area in robotics research, because the terrific applications this kind of technology can provide. There is a wide scope for civil UAV applications, such as automatic and unattended delivery of goods like Amazon intends to do, news coverage offering better perspective for snapshots and recorded video, visual inspection in difficult areas (think in natural disasters for instance), surveillance missions in enclosed areas (parking, campus, etc), and so on.

  One of the developed projects using UAV and video tracking consist in programming a quadcopter to follows objects (see figure 1.7). The object is selected from the front side camera by converting the original image into HSV color space, smoothing the image with Gaussian Blur, applying a color filter, and finally extracting the blob

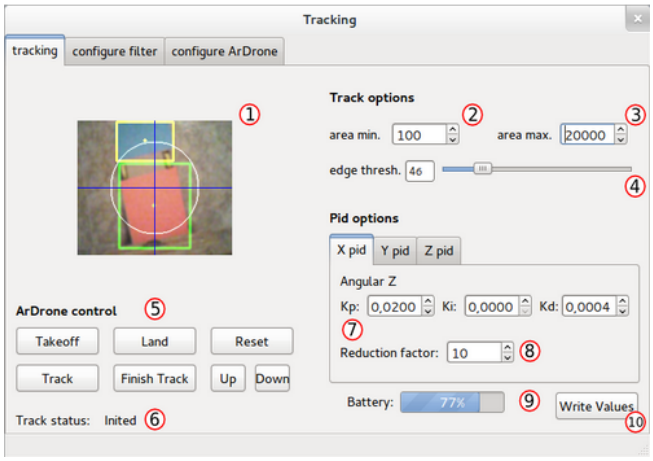


Figure 1.7: Airdrone interface for tracking objects developed in URJC

centroid and applying a low pass filter to avoid variation in centroid coordinates due to changes in light conditions or drone movements.

Once the object has been isolated from the background, the tracking module activates the horizontal and vertical tracker in order to follow the object using a Proportional-Integral-Derivative (PID) controller.

The Air Drone project is also able to estimate its altitude using video techniques, having a track of the reference objects visualized by its ventral camera and counting its variation in size on the perceived image.

- *Traffic Monitor* [5]

  This project deals with an application that is able to carry out precise estimation of vehicle speeds. Besides, it counts the number of cars going on the road. For this purpose, it was used a single camera, as the unique source of information. The images it provides are analyzed to give estimations of vehicle speeds.

  The CarSpeed application detects and tracks the vehicles on the road to provide the estimated speed of each one using an evolutionary algorithm. The cars are detected with a background subtraction technique that consists in calculating the difference between the actual image -with cars- and the highway background image -without cars.

  Once one car is detected the system computes all the possible candidates with different speeds associated with it, using a constant speed model. After that it computes the health function of each candidate in order to accept only those

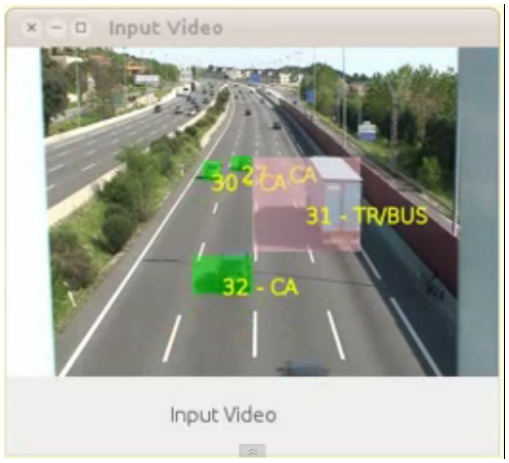


Figure 1.8: Traffic Monitor

candidates that better match with the detected object, showing the speed associated to that candidate (see figure 1.8).

- *ElderCare* is an application developed with the aim of assist and taking care of elder people in their own home. The application implements a 3D tracking of people in order to detect when an elder person has fallen down (see figure 1.9).

  The first approach was able to identify and track a unique person with a predefined color. The second version [8] implemented a multiple people tracking by automatic learning of each individual color. The third version of the project [9] implements a more robust technique to detect people by color and multi-modal evolutionary algorithm for tracking them.

  The system receives the images provided by four cameras located in each room corner, identifies people in movement with image subtraction techniques between frames, and learns the color of the resulting sections based on histogram values of each one.

  The 2D tracking uses a multi-modal evolutionary algorithm that weights the health of each individual candidate by multiple methods: density of the pixels in movements, histogram, matching of characteristics points (SIFT and SURF), and Kalman. The Kalman Filter was used to check the convergence between the position estimation of each individual candidate and the one provided by the Kalman Filter.

  The 3D tracking implements a multi-modal evolutionary algorithm too, weighting each individual health function with color and movement density inside the prism that surrounds the target, and the color and movement density in the neighbor of the prism surrounding the target.

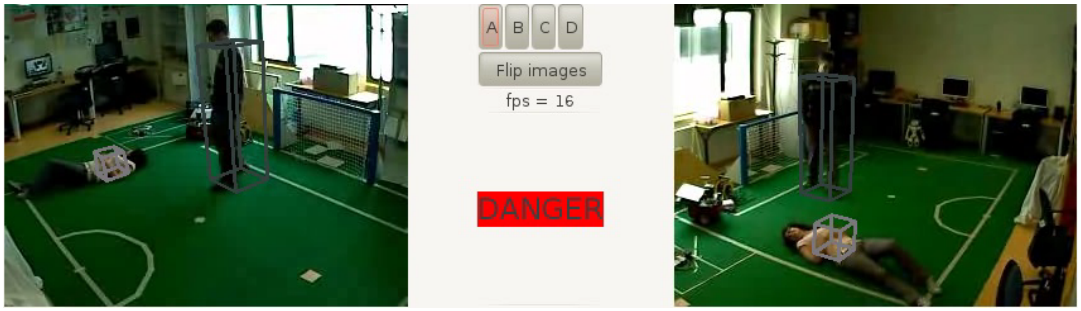


Figure 1.9: ElderCare

- *Algorithms Analysis for Visual Object Tracking* [11]

  That master thesis analyzes different approaches to track objects. Begins with classic methods based on object segmentation using background learning techniques between consecutive frames (see figure 1.10), getting the characteristics points of an image (those pixels that are invariant between frames), getting descriptors such as SIFT, SURF, ORB, BRIEF, FREAK, etc., and common matchers using brute force like FLANNE and Lucas Kanade.

  The second part of that master thesis consists in implementing the PixelTrack algorithm. In contrast to classic approaches, PixelTrack can be used with fixed or mobile cameras. The algorithm needs a first input about where the object to be tracked is, and dynamically updates its position using a Hought and segmentation technique over the seek area with continuous feedback between both.

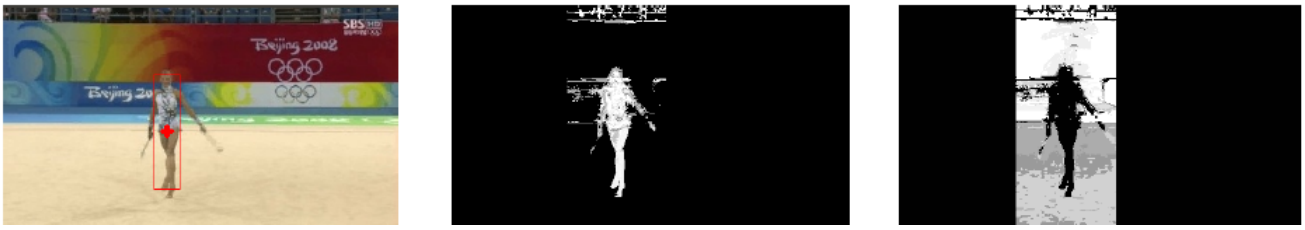


Figure 1.10: PixelTrack segmentation: a) user selected input b) foreground c) background

Chapter 2

Objectives

The global aim of this master thesis is the development of an algorithm for robust visual tracking of multiple objects using several probabilistic filters, such as Kalman Filter, Probabilistic Data Association Filter and Join Probabilistic Data Association Filter. The input of the different algorithms is a real image sequence.

In order to achieve the general goal, it has been divided in three subgoals:

1. *Development of an observation Module*: Because the general purpose of this master thesis is to effectively visual track objects in real images, to have a method to determine where the objects to be tracked are in the image it is needed. In this work different object extraction techniques will be developed, including the basic based in color filter and those based in background subtraction techniques. Because of there are different approaches to extract objects it is a great opportunity to test the most valuable method. For each observed object in scene, with independence of segmentation method used, the observation module will build a list of object coordinates to be processed by the available tracking filters in order to track them.

2. *Design and implementation of a tracking Module*: Once it is known where the objects to be tracked are on the image, we need a method to track them. The probabilistic methods to be developed are *Kalman Filter*, *Probabilistic Data Association Filter* and *Join Probabilistic Data Association Filter*. They will be able to work with single or multiple targets simultaneously, adapting such filters to work in multi track environment when necessary. The tracking system to be developed will be able to follow the object paths when occlusions or noise arise. Not all the filters suitable to track objects can deal with these situations and therefore it is required to find the best of them.

3. *Experimental Validation*: The final phase consists in evaluate the different object extraction techniques working in conjunction with the three probabilistic tracking filters implemented, in order to determine how they deal with noise (originated by the observation module) and occlusions in single and multiple visual object tracking scenarios. In this sense, the filters must be tested to verify how they work with the variation of its fundamental parameters in order to know their strengths and weakness.

2.1 Requirements

The general requirements to be considered when implementing the code follows:

1. *Object extraction.* The objects must be extracted from scene independently they are in movement or remain stable.

2. *Robustness against occlusions.* What happens if the tracked object suddenly disappear from the image? If we are tracking the ball in a soccer field, using cameras installed over the land in the boundaries of the field, for sure the ball will disappear occluded by the soccer players from time to time. In the case the object was not detected in the scene an occlusion happens. In this case the algorithm has to keep track of the object.

3. *Cross paths.* When tracking several objects simultaneously, their tracks usually cross. A desirable requirement is that the system should not miss the path of the tracked objects when cross paths happen.

4. *Robustness against sensor noise.* Depending on the used technique to isolate the object the presence of noise will raise. The noise must be understood as inaccuracies in object extraction techniques or due to false positives, and the tracking algorithm has to cope with it.

2.2 Methodology

The developed system have been built by using the spiral model with prototyping, due it allows to develop in an incremental way, growing in complexity and delivering functional prototypes. Using such methodology several functional prototypes have been released, with

the benefit to allow the reuse of each one in the subsequent modules. The spiral model also allows a quickly change adaptation, as required in this kind of projects.

The spiral model performs four basic activities in every cycle, as shown in the figure 2.1, each of one with a different objective:

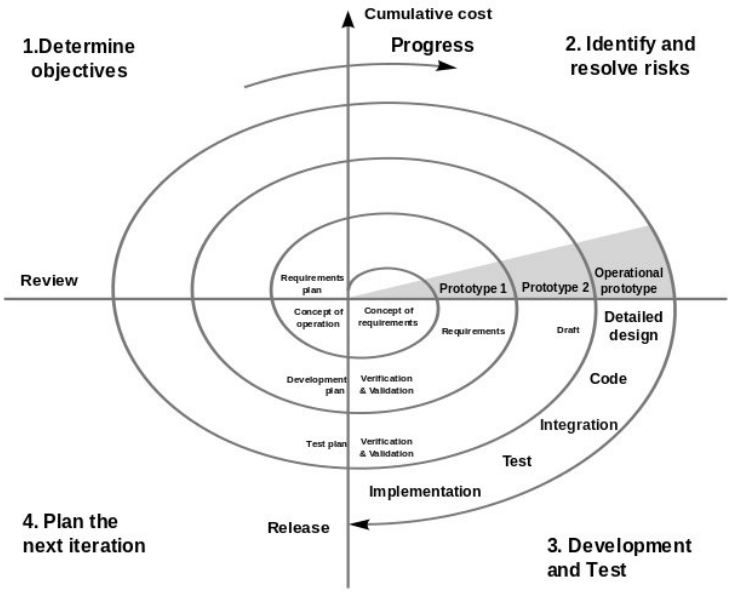


Figure 2.1: Spiral model based on prototypes

- Determine objectives: Consider the win conditions of all success-critical stakeholders for each iteration, having in mind the goal objectives. In each iteration the overall cost and complexity of the product will grow.
- Identify and resolve risks: Identify and evaluate alternative approaches for satisfying the win conditions established in the previous phase, having in mind risk must be minimized.
- Development and test: Identify and resolve risks that stem from the selected approach(es). Developing of the prototype following the best approach to reach the objectives of the iteration, and testing the overall implementation.
- Plan the next iteration: Obtain approval from all success-critical stakeholders, plus commitment to pursue the next cycle.

At the beginning of the master thesis regular meetings will be scheduled, to better align the progression of the developed work with the general objectives of the master thesis.

Furthermore the use of the project media wiki is a must to better check the features implemented. The project media wiki can be find at reference [10].

The code, papers, and generated documents will be updated to the central URJC repository using the Apache Subversion (SVN[1]).

2.3 Working plan

Using the above methodology several functional prototypes will be developed according to the following goal objectives for the master thesis:

1. *Understanding JdeRobot platform and related components* required to use in this thesis. Before anything else a first contact with `cameraserver` component will be needed, developing minor changes in its behavior to better understand how it works. It will be required to change the color spaces and little transformation over the images to be served in order to get enough knowledge of the component.

2. *Observation Module*

   (a) *Synthetic Object simulator.* As a help in developing and tunning the different tracking algorithms to be implemented, a synthetic object simulator will be developed. This first deliverable will be used to test the behavior of different filters before they will be used with real images.

   (b) *Basic Object Detection using color filtered images.* The object to be tracked will be extracted from the image using color filter techniques based on HSV image filtering. Once the object have been isolated from the image the centroid of the resulting image will be obtained and considered as the center of the object we will intend to track. Because the method to be used to get the object centroid will be based in selecting a range of colors from the image, a lot of false positives are supposed to be generated. Although this is not a problem when using PDAF or JPDAF algorithms it will be hard to deal with basic Kalman Filter, forcing us to have a fine tunning of the color filter process in order to avoid the inaccuracy of the resulting implementation.

[1] http://subversion.apache.org/

(c) *Object extraction from background.* As an improvement of the previous stage, an object extraction technique will be implemented using learning backgrround techniques. So the objects will be isolated from the scene by identifying the changes in the background they originates frame by frame. Applying this extraction technique will allow us to identify the contour of the object in movement, but in order to detect all the contour as an unique object, avoiding the detection of sub blobs, we will need to transform and dilate the resulting contour.

3. *Tracking Module*

(a) *Kalman Filter.* The first milestone in the project will be to deal with basic Kalman filter algorithm, and it will consists in implementing a first framework able to track an object in the plane using synthetic images. Afterwards it will be the implementation of Kalman Filter to support operations with real images, so after the object detection will happen, it will be needed to inject its coordinates into the measurement model and to observe if the prediction phase is working properly. This first phase will only support a single track.

i. Afterwards it will come the *multi track implementation for Kalman Filter*, using the simplest approach as possible, as an array of the single track filter implementation adding the control logic to create or delete tracks.

ii. *Error ellipse associated with the innovation covariance.* An important part of the implementation will be to deal with the error ellipsoid associated with the covariance of the predicted state, because the error ellipse represents how the uncertainty grows in the absent of measurements. It will be useful in multi kalman implementation, and also in further stages of this master thesis, when PDAF and JPDAF will be implemented, as a fundamental method to associate the detected measurements according to a certain probability in the current track.

(b) *Probabilistic Data Association Filter:* The second milestone will be implementing the complete algorithm for Probabilistic Data Association Protocol with a single object to track in the plain. To achieve this objective it will be needed an extension of the simulator module in order to support the first approach of PDAF implementation using synthetic images.

Afterwards it will be the time to apply the PDAF algorithm to be used with real images.

i. *Multi track implementation for Probabilistic Data Association Filter.* Despite the PDAF approach was originally designed to track a single track in noisly environments, a multi PDAF track will be implemented using the simplest approach as possible, as an array of the single PDAF implementation adding the control logic to create or delete tracks.
To achieve this objective, it will be implemented a method to associate an independent PDAF to each object, with the required logic to create or delete tracks.

(c) The final part will consist in implementing the *Join Probabilistic Data Association Protocol*, that will require multiple changes in the previous work due it will be needed to implement the semantics to support the multi track environment, because of the complexity to calculate the partial probabilities of each track having in mind the original algorithm requires a previously known number of tracks, and one of the requirements of the master thesis is the ability to create or delete tracks dynamically.

4. *Experiment Module.* As a general objective of the project, and to better refine the code and implemented algorithms, an experiment module will be developed, able to track virtual objects with basic Kalman Filer, PDAF and JPDAF. It will be used to verify each of the parts needed to reach the goals of this master thesis. It will be able to simulate random noise to observe how PDAF or JPDAF deals with it, random movements of the virtual objects to be tracked, abrupt movement of the particles to watch how occlusion takes part, crossing paths of several objects to see how the different tracking methods deals with, and so on.

   In addition, the experiments module will be use to have a fine tunning of the filters when tracking with real images will happen.

5. Following the spiral model exposed previously it will be required to continuously code, test and debug the implemented features, in order to obtain the final deliverable of the master thesis.

Chapter 3

Infrastructure

A description of the software and hardware elements used to develop this project are enumerated in this chapter.

The host system will be under the Linux Ubuntu 64bits platform, version 12.04.4 TLS. The hardware required to implement this project does not demand specific storage or processor capabilities. I used a conventional laptop provided with an i5 dual core vPRO HT Intel processor working at 1.9 Ghz and 8 Gigs of memory.

Instead of using all the available hardware to this project, I have used a virtual machine with 22 GiB of storage and 4 Gigs of RAM. The resources available were more than enough to develop and run the code, although when JPDAF with multiple track were used the overall performance was dropped.

3.1 Ice

The Internet Communications Engine [14], or Ice[1], is an object-oriented middleware platform that provides object-oriented remote procedure call, grid computing and publish/subscribe functionality developed by ZeroC and dual-licensed under the GNU General Public License and a proprietary license. It supports C++, Java, .NET-languages (such as C# or Visual Basic), Objective-C, Python, PHP and Ruby on most major operating systems such as Linux, Solaris, Windows and Mac OS X. A light variant of Ice runtime, called Ice-e, may run inside mobile phones. As its name indicates, the middleware may be used for applications without the need to use the HTTP protocol and is capable of traversing firewalls unlike other middleware.

[1] `http://www.zeroc.com`

ZeroC was founded in 2002 in Florida. Ice was influenced by the Common Object Request Broker Architecture (CORBA) in its design, and indeed was created by several influential CORBA developers, including Michi Henning. However, according to ZeroC, it was smaller and less complex than CORBA because it was designed by a small group of experienced developers, instead of suffering from design by committee.

Ice components include object-oriented remote-object-invocation, replication, grid-computing, failover, load-balancing, firewall-traversals and publish-subscribe services. To gain access to those services, applications are linked to a stub library or assembly, which is generated from a language-independent IDL-like syntax called slice.

- **IceStorm** is an object-oriented publish-and-subscribe framework that also supports federation and quality-of-service. Message content consist of objects of well defined classes.
- **IceGrid** is a suite of frameworks that provide object-oriented load balancing, failover, object-discovery and registry services.
- **IcePatch** facilitates the deployment of ICE-based software.
- **Glacier** is a proxy-based service to enable communication through firewalls, thus making ICE an internet communication engine.
- **IceBox** Icebox is a service-oriented architecture container of executable services implemented in .dll or .so libraries. This is a lighter alternative to building entire executable for every service.
- **Slice** Slice is a Zeroc-proprietary file format that programmers follow to edit computer-language independent declarations and definitions of classes, interfaces, structures and enumerations.

The program developed in this master thesis uses the ICE 3.4.2 version to receive images from JdeRobot cameraserver component.

3.2 JdeRobot

JdeRobot[2] is a software development suite for robotics, home-automation and computer vision applications. These domains include sensors (for instance, cameras), actuators, and intelligent software in between. It have been designed to help in programming such intelligent software. It is written in C++ language and provides a distributed component-based programming environment where the application program is made up of a collection of several concurrent asynchronous components. Each component may run in different computers and they are connected using ICE communication middleware. Components may be written in C++, python, Java... and all of them interoperate through explicit ICE interfaces.

JdeRobot simplifies the access to hardware devices from the control program. Getting sensor measurements is as simple as calling a local function, and ordering motor commands as easy as calling another local function. The platform attaches those calls to the remote invocation on the components connected to the sensor or the actuator devices. They can be connected to real sensors and actuators or simulated ones, both locally or remotely using the network. Those functions build the API for the Hardware Abstraction Layer. The robotic application get the sensor readings and order the actuator commands using it to unfold its behavior. Several driver components have been developed to support different physical sensors, actuators and simulators. The drivers are used as components installed at will depending on your configuration. They are included in the official release.

JdeRobot includes several robot programming tools and libraries. First, viewers and teleoperators for several robots, its sensors and motors. Second, a camara calibration component and a tunning tool for color filters. Third, VisualHFSM tool for programming robot behavior using hierarchical finite state machines. It includes many sample components using OpenCV, PCL, OpenGL, etc.. In addition, it also provides a library to develop fuzzy controllers, a library for projective geometry and some computer vision processing.

Each component may have its own independent Graphical User Interface or none at all. Currently, GTK and Qt libraries are supported, and several examples of OpenGL for 3D graphics with both libraries are included.

JdeRobot is open-source software, licensed as GPL and LGPL. It also uses third-party software like Gazebo simulator, OpenGL, GTK, Qt, Player, Stage, Gazebo, GSL, OpenCV, PCL, Eigen, Ogre.

[2] `http://www.jderobot.org`

This master thesis will use several modules of JdeRobot platform version 5.2.1: *cameraserver* in order to manage the camera images to be provided and *colorspaces* structure. Because of JdeRobot was written using C++ with Linux as operative system, the implemented code will be written in C++ language too, using the Ubuntu Linux distribution.

Due there is no more dependencies in the code than *cameraserver* client and types for manage color spaces (*colorspaces*), the code could be easily decoupled if needed. Because it will be used several facilities from JdeRobot platform, the resulting program should feel like any other module of such platform.

3.3 Glade interface

Glade[3] is a RAD (Rapid Application Development) tool to enable quick and easy development of user interfaces for the GTK+ toolkit and the GNOME desktop environment.

The user interfaces designed in Glade are saved as XML, and by using the GtkBuilder GTK+ object these can be loaded by applications dynamically as needed.

By using GtkBuilder, Glade XML files can be used in numerous programming languages including C, C++, C#, Vala, Java, Perl, Python,and others.

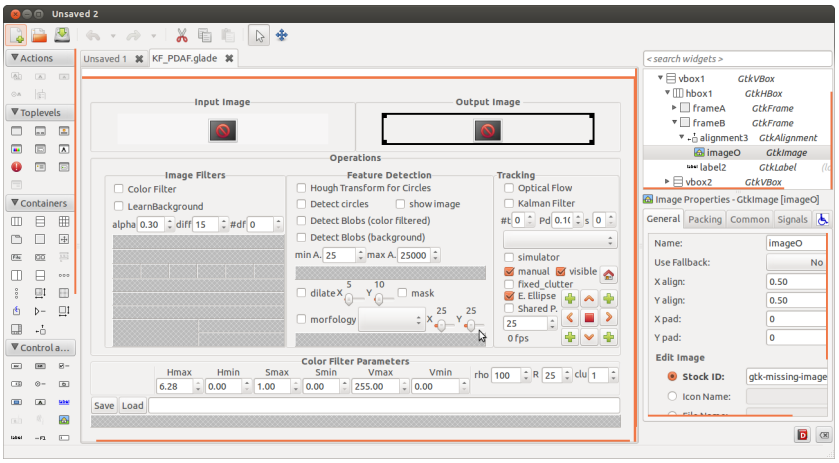


Figure 3.1: Glade, tool for rapid development of interfaces using GTK+

Glade is Free Software released under the GNU GPL License

[3] `https://glade.gnome.org/`

I have used Glade version 3.8.0 to build the Graphical User Interface of this master thesis. In figure 3.1 we can see the main framework developed in this master thesis.

3.4 OpenCV

OpenCV (Open Source Computer Vision Library) is a library of programming functions mainly aimed at real-time computer vision, developed originally by Intel to support advance CPU-intensive applications. Nowadays is supported by Willow Garage and Itseez. It is free for use under the open source BSD license. The library is cross-platform. It focuses mainly on real-time image processing. If the library finds Intel's Integrated Performance Primitives on the system, it will use these proprietary optimized routines to accelerate itself.

Figure 3.2: OpenCv facilities used for matrix calculations and basic image processing

Library documentation can be found through wikid pages[4].

In August 2012, support for OpenCV was taken over by a non-profit foundation, OpenCV.org, which maintains a developer[5] and user site[6].

In this master thesis I have used the OpenCV version 2.4.7. Nowadays the library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, etc.

Despite OpenCV library has a kalman filter implementation I didn't used it because it does not fit well with the purpose of this master thesis. Therefore, the three implemented

[4] `http://opencv.willowgarage.com/wiki/`
[5] OpenCV Developer Site: `http://code.opencv.org`
[6] OpenCV User Site: `http://opencv.org/`

track filters have been built from scratch, using the matrix calculation facilities OpenCV has in order to simplify the code required to implement the probabilistic filters.

In addition of matrix facilities it was used functions for basic image processing, such as image conversions between color spaces, contours finding, etc.

Chapter 4

Theoretical Basis of Probabilistic Filters

This chapter gives a theoretical description of the different state estimation techniques, considering the association algorithm used in each one. We assume the state will be the position of the object to be tracked. In real applications we are not able to make continuous measurements of the tracked object, so the problem falls into the estimation theory, as a statistical specialization used to predict an approximate value (state estimation) from real measurements (observations).

When considering object tracking we need first to develop a mathematical model that represents the movement of the object to be tracked, having in mind that there is no exact method to perfectly represent its movements, because the impossibility to determine all the factors that influence the object in action. In this way the mathematical model and the dynamics of the system to be modeled are a source of uncertainty.

Besides the modeling of the system dynamics, we need to consider the observation model too, that typically consists in getting the object position using any kind of sensor, such a camera. There is not a sensor that provides perfect and complete data about the object to be observed, and so, this is another source of uncertainty. If we want to build a system able to effectively resolve the problem of a visual object tracking, we need to deal with uncertainty, even more if we are using limited systems dynamics model and inaccurate sensors.

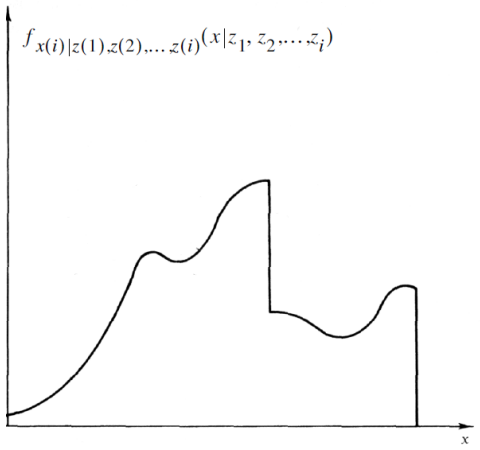


Figure 4.1: Conditional Probability Density

Consider the figure 4.1 that represents the conditional probability density of the variable x at time $x(i)$, whose values are conditioned to the measurements represented by $z(i) = z_i$ at time instant i. For example, $x(i)$ can be the position of an object and $z(j)$ can be the vector that describes the position measurements using a couple of cameras. The function indicates the probability of $x(i)$ considering all the available measurements taken up through time instant i. The shape of the graph represents the amount of certainty in the knowledge of the value of x. The most certainty of x values will be plotted as a narrow peak, whereas the less certainty of x values will be represented with a gradual shape, indicating the probability weight is spread over a wider rang of x values.

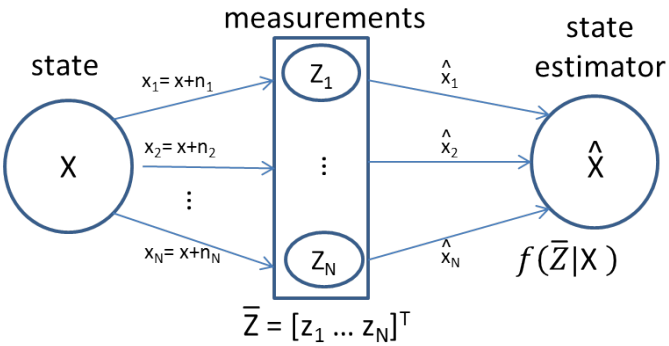


Figure 4.2: Estimator Process

In the figure 4.2 the state we want to estimate (represented by X) is not directly observable and is considered to be aleatory. $\bar{Z} = [z_1, \cdots, z_N]^T$ is the measurement vector, $f(\bar{Z}|X)$ is the transformation function (with certain *pdf*) depending of state and $\hat{X}$ is the state estimator (also with certain *pdf*).

The estimator can be considered as a "measurement instrument" of the values the state will have, depending of actual state and the available observations, with an estimation error (with certain pdf) defined by:

$$\epsilon = X - \hat{X} \tag{4.1}$$

Generally speaking probabilistic estimators represent the state of the system using probability distributions. Probabilistic filters are based on Bayes filter algorithm whose objectives are the estimation of values of a state vector X given an observation vector Z, where both vectors are considered to be random. The state vector is governed by the systems dynamic model with some random noise due the unmodeled features. The observation vector also has some random noise associated with the measurement process, and is assumed to have a known prior density function $p(x)$. This prior probability distribution contains all the knowledge about the state vector before the inclusion of any measurement to the system.

A discrete dynamic process can be defined as a process where the current system state depends on one or more prior states. If the current state depends only on the previous one we have a first-order Markov process, so a discrete Markov dynamic process can be characterized as:

$$x_n = f(x_{n-1}, u_n, \eta_{n-1}) = f(x_{n-1}, u_n) + v \tag{4.2}$$

where x_n is the state of the system at time t_n, f is the transition function that moves the state x between time t_{n-1} to time t_n, u_n is a known control input that drives the system dynamics and η is the white noise that represents those parts of the transition functions that were not modeled.

Because we need to estimate the unobservable state vector x_n based on all the available measurements represented by vector $Z_{1:n} \triangleq \{z_1, z_2, \cdots, z_n\}$ we can assume a relation between the observation vector z_n at time t_n and the state vector x_n at time t_n.

$$z_n = h(x_n, \eta_n) = h(x_n) + w \tag{4.3}$$

Where h_n is the observation function that connects the observations with the state vector. Equations 4.2 and 4.3 define a system model and an observation model that allow us to obtain the estimation of state vector x_n, that can be expressed using Bayesian approach like the estimation of conditional posterior density $p(x_n|z_{1:n})$ taking into consideration all the available observations $z_{1:n}$ as follows:

$$p(x_n|z_{1:n}) = \frac{p(z_{1:n}|x_n)p(x_n)}{p(z_{1:n})} \tag{4.4}$$

The posterior density $p(x_n|z_{1:n})$ contains all the knowledge about the state vector after taking into account values of the measurements vector. Applying reductions and Bayes law yields the following progression:

$$p(x_n|z_{1:n}) = \frac{p(z_n|x_n)p(x_n|z_{1:n-1})}{p(z_n|z_{1:n-1})} \tag{4.5}$$

Considering $p(z_n|z_{1:n-1}, x_n) \rightarrow p(z_n|x_n)$ because the observations at time t_n does not depend on the observations at time $t_{1:n-1}$ and applying the Chapman-Kolmogorov equation gives us the relationship between the prior density and the posterior density.

$$p(x_n|z_{1:n-1}) = \int p(x_n|x_{n-1})p(x_{n-1}|z_{1:n-1})\mathrm{d}x_{n-1} \tag{4.6}$$

4.1 Kalman Filter basics

The Kalman Filter theory was published by Rudolf Emil Kálmán in 1960, describing a revolutionary method to solve the discrete-data linear problem in an optimal way, a mathematical algorithm that is widely used in signal processing, control and guidance systems. For his work Kálmán was awarded with the National Medal of Science on October 7, 2009.

Rudolf E. Kálmán published his work in 1960 with a relative success until he exposed his ideas to the NASA Ames Research Center, who quickly adopted the new technique during the Apollo program, and furthermore, in the NASA Space Shuttle. Later it was used in Navy submarines, unmanned aerospace vehicles and weapons.

Nowadays Kalman filter and derivatives are being used widely in different areas of knowledge, from economics (in order to obtain successful predictions about the expected inflation) to navigation systems (in order to have an accurate estimation about the position and speed of a mobile vehicle in absence of measurements). Another area of application for Kalman based filters is that relative to target tracking systems, both in military and civilian applications such as wildlife tracking, traffic surveillance both in metropolitan and highway areas in order to develop efficient and intelligence transport systems and air traffic controls.

Kalman filter is an optimal recursive probabilistic estimator. It is optimal because the mean, mode, median and all the optimal estimates coincides, for those systems than can be represented with a linear model, whose system and measurement noise are white (the noise is not correlated in time) and Gaussian. It is recursive because it does not require to retain and reprocess all the previous data every time a new measurement is considered.

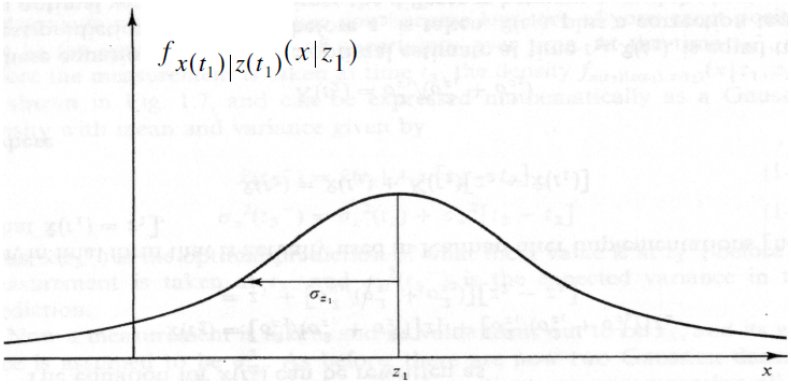


Figure 4.3: Conditional density of position based on measurement z_1

In the figure 4.3 the x parameter represents the position probability, the z_1 is the position we have measured with standard deviation (uncertainty) σ_{z_1}. This way the conditional probability of $x(t_1)$ represents the position at time t_1 considering the measurement z_1. A significant percentage of the probability certainty is contained within the band of σ_{z_1} in both sides of the mean. Under this circumstances the best estimate of the position is $\hat{x}(t_1) = z_1$ with error variance $\sigma_x^2(t_1) = \sigma_{z_1}^2$.

In the figure 4.4 we can observe how a second measurement z_2 is available with lower σ_{z_2}. Because this measurement is considered to be more precise than the first one the graph shows a narrower peak due to the smaller variance this second measurement has.

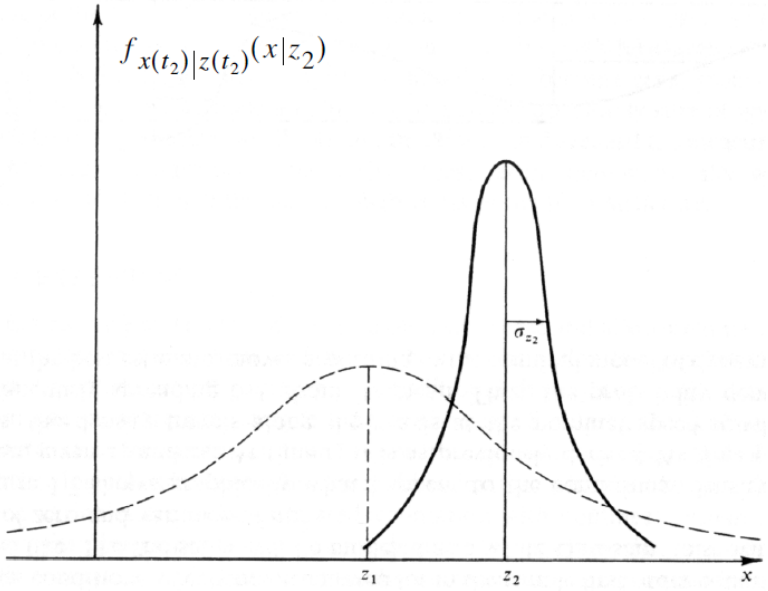


Figure 4.4: Conditional density of position based on measurement z_2

The combination of both measurements (state estimation for $x(t_2)$), with their respective inaccuracies, results in a Gaussian with mean μ and variance σ^2.

$$\mu = z_1 \frac{\sigma_{z_2}^2}{\sigma_{z_1}^2 + \sigma_{z_2}^2} + z_2 \frac{\sigma_{z_1}^2}{\sigma_{z_1}^2 + \sigma_{z_2}^2} \tag{4.7}$$

$$\frac{1}{\sigma^2} = \frac{1}{\sigma_{z_1}^2} + \frac{1}{\sigma_{z_2}^2} \tag{4.8}$$

This can be seen in figure 4.5. The uncertainty in the estimation position for $x(t_2)$ has been decreased because the new variance σ is less than either σ_{z1} and σ_{z2}.

In the case the standard deviations for both measurements were equal, what means the observation was made using the sames sensors in the same conditions, the equation 4.7 shows the optimal estimate of position will be the average of both measurements. In case σ_{z_2} were larger than σ_{z_1} indicating the uncertainty of z_2 measurement is greater than that of z_1, the new mean μ will be weighted in favor of z_1. With the consideration of measurements, the variance of the estimations will decrease, indicating the more precision of the filter.

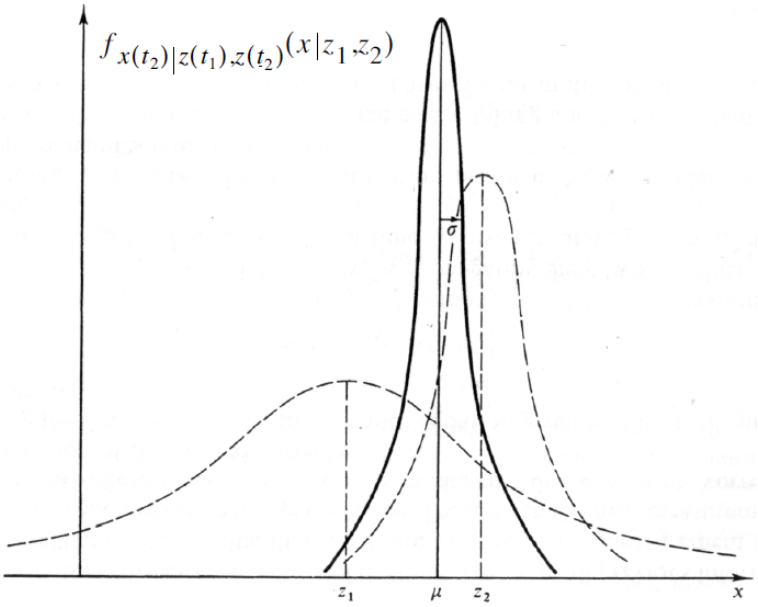


Figure 4.5: Conditional density of position based on measurements z_1 and z_2

The Kalman Filter solves optimally the estate estimation of a controlled process that is governed by a linear stochastic equation, given by the following expression:

$$x_k = Fx_{k-1} + Bu_{k-1} + w_{k-1} \tag{4.9}$$

where $x_k \in \Re^n$, F is the $n_\times n$ transition matrix that addresses the evolution of the system between the previous and actual step k, B is the $n_\times 1$ matrix that relates the optional control input $u_k \in \Re^1$ to the state x_k, and w_k is a random variable that represents the process noise that is assumed to be white, following a normal distribution $p(w) \sim \mathcal{N}(0, Q)$ with covariance matrix Q_k. The observation of the modeled system $z_k \in \Re^m$ is

$$z_k = Hx_k + v_k \tag{4.10}$$

where the matrix $H_{m_\times n}$ relates the state x_k with the measurement z_k, and v_k is a random variable that represents the measurement noise, assumed to be white and with a normal probability distribution $p(v) \sim \mathcal{N}(0, R)$ with covariance matrix R_k.

The variables w_k and v_k are assumed to be independent and, depend on the implementation, the matrices Q (process noise covariance) and R (measurement noise covariance) might change in each time step or measurement.

We can define $\hat{x}_k^-$ as the priori estate estimation at step k given knowledge of the process prior to step k, and $\hat{x}_k$ as the posteriori state estimate at step k given measurement z_k.

The filter is composed of two stages that run giving feedback each other: prediction and correction as seen in figure 4.6. During the prediction phase the filter obtains a prediction of the state vector using the system model and Bayes' rule, making the process noise zero. In the correction phase, the algorithm updates the estimation weighted with real and priori estimate measurements.

This means that given a process model of the system to be evaluated, a set of measurements delivered from sensors that reports the state of the system, a statistical description of the system noise, the measurement noise and the uncertainty of the model, with any initial state of the system, Kalman Filter can predict, in an optimal way, the future state of the system. The Kalman Filter estimates a process state using the feedback given by the evolution of the own process during the time. In general terms, the Kalman Filter algorithm can be represented in two stages, as follows:

Figure 4.6: Discrete Kalman Filter Cycle

During the prediction phase, also called propagation stage, the Kalman filter projects the state ahead in time according to the model equations, giving a priori estimates for the next time step. In the measurement update stage, the filter corrects the projected state with a real -and noisy- measurement to obtain an improved posterior estate.

We can define the priori and posteriori estimate errors as the difference between the measurement and the priori estimation of the measurement at step k, and the difference between the measurement and posteriori estimation of measurement at step k respectively, as follows:

$$e_k^- \equiv x_k - \hat{x}_k^- \tag{4.11}$$

$$e_k \equiv x_k - \hat{x}_k \tag{4.12}$$

Then a priori and posteriori estimate error covariance are

$$P_k^- = E\left[e_k^- e_k^{-T}\right] \tag{4.13}$$

$$P_k = E\left[e_k e_k^T\right] \tag{4.14}$$

The goal of the Kalman Filter consists in finding an expression that obtain a posteriori state prediction as a linear combination of the a priori state estimation $\hat{x}_k^-$ and a weighted difference between the actual measurement $\hat{z}_k$ and the measurement prediction based on $H\hat{x}_k^-$, as follows:

$$\hat{x_k} = \hat{x}_k^- + K(z_k - H\hat{x}_k^-) \tag{4.15}$$

The difference $(z_k - H\hat{x}_k^-)$ is called residual or measurement *innovation*, and shows the discrepancy between the actual measurement z_k and the predicted one $(H\hat{x}_k^-)$. The $K_{n_\times m}$ matrix in equation 4.15, named the *Kalman gain*, minimizes the posteriori error covariance in equation 4.14.

Discrete Kalman filter prediction (motion model) equations

Project the state ahead:	$\hat{x}_k^- = F\hat{x}_{k-1}^- + B\,u_{k-1}$	(4.16)
Project the error covariance ahead:	$P_k^- = FP_{k-1}F^T + Q$	(4.17)
Predicted measurement:	$\hat{z}_k^- = H\hat{x}_k^-$	(4.18)

Discrete Kalman Filter correction equations

Compute the Kalman Gain:	$K_k = P_k^- H^T (HP_k^- H^T + R)^{-1}$	(4.19)
Update the estimation with measurement z_k:	$\hat{x_k} = \hat{x}_k^- + K(z_k - \hat{z}_k^-)$	(4.20)
Update the estimation error covariance matrix:	$P_k = (I - K_k H)P_k^-$	(4.21)

In the above equations we can extract from equation 4.19 the predicted measurement covariance matrix (AKA innovation or residual covariance matrix), that contains the variance of each measurement dimension contained in the measurement vector.

$$S_k = HP_k^- H^T + R \tag{4.22}$$

Inside the equation 4.20 it is used the *innovation*, defined by the difference between the measurement and the predicted measurement:

$$\hat{r}_k = z_k - \hat{z}_k^- \tag{4.23}$$

During the propagation equations, the state estimate is updated from systems dynamics (equation 4.16) and the uncertainty (error covariance) grows (equation 4.17). Finally the filter computes the predicted measurement, that is, the expected value of sensor reading based on the previous state estimation (equation 4.18).

During the update or correction equations, the filter computes the covariance matrix associated to the predicted measurement (equation 4.22) that it is used to calculate the Kalman gain (equation 4.19) telling us if the filter weights more on real measurements than in predictions or vice versa when it will be used to weight between the *innovation* and the priori estate estimation when the filter computes the update of the estate in equation 4.20. So the *Kalman gain* can be understand as an index that tell us how much to correct the estimation. Finally, the filter updates the uncertainty, that is, the estimation error covariance, that will shrinks if the system can update the estate estimation using real measurements.

Extended Kalman Filter

As mentioned before, Kalman Filter addresses the problem of state estimation of a discrete-time controlled process that is governed by a linear stochastic equation, but there are many systems that are highly non linear. In such systems Kalman Filter can not be applied as it is.

Extended Kalman Filter allows to linearize state propagation equations and the sensor models about the current state estimate, with the collateral of increasing the residual of state error due linearization is not the best estimate. Assuming the process is governed by a non linear stochastic difference equation, the state vector $\hat{x}_k \in \Re^n$ is

$$\hat{x}_k = f(\hat{x}_{k-1}, u_k) + w_k \tag{4.24}$$

where the random variable w_k represents the process noise with $p(w_k) \sim \mathcal{N}(0, Q_k)$, and Q_k is the process noise covariance matrix. The measurement vector $z_k \in \Re^m$ is

$$z_k = h(\hat{x}_k) + v_k \tag{4.25}$$

where the random variable v_k represents the measurement noise with $p(v_k) \sim \mathcal{N}(0, R_k)$, and R_k is the measurement noise covariance matrix.

The predict and update equations are practically the same we already have in Kalman Filter, except when it computes the project state ahead and predicted measurement (now calculated by $\hat{x}_k^- = f(\hat{x}_{k-1}^-, u_k)$ and $\hat{z}_k^- = h(\hat{x}_k^-)$ respectively). The state transition and observation matrices (F and H respectively) are defined by the following jacobians:

$$F = \left.\frac{\partial f}{\partial x}\right|_{\hat{x}_{k-1}^-, u_{k-1}} \qquad H = \left.\frac{\partial h}{\partial x}\right|_{\hat{x}_k^-} \tag{4.26}$$

4.1.1 Tunning the filter parameters

The measurement noise covariance is represented by R and, generally speaking, it can be set before the filter goes ahead. We can set the measurement noise value in an easy way, by taking some off line measurements and determining the variation between each one. If measurement noise covariance changes during the filter operations R is named R_k. So, making R bigger reduces the Kalman gain K_k (by equation 4.19) and the filter trusts less in the measurement and more in the predicted state (see equation 4.20). On the other hand, making R smaller makes the Kalman gain K_k bigger and the filter weights rather on measurements than in predictions.

The determination of the process noise covariance Q is not an easy thing except if we could observe the process directly. In this case when a tracking system is being implemented we can reduce Q when the object is moving slowly and increase it when the object varies its acceleration. In this case when Q changes with time it is named Q_k. In the case the Q and R are constant, both the estimation error covariance P_k and the Kalman gain K_k will stabilize quickly and remain constant.

4.1.2 Occlusions

There are different situations in which a $Kalman\ filter$ goes ahead in prediction equations without having new measurements to feed the filter. For example when a tracked object can not be identified by the sensors, and therefore the filter can not have real measurements to correct the predictions. This is named as object occlusion.

In any case, it is possible to by pass the update equations in the $Kalman\ filter$ simply multiplying the *kalman gain* equation (see equation 4.19) by a zero Δ_k matrix in the absence of measurements, or by a unity Δ_k matrix when measurements will be available again.

$$K_k = P_k^- H^T (H P_k^- H^T + R)^{-1} \Delta_k \tag{4.27}$$

In order to have a more efficient code, in absence of measurements, is convenient to bypass all the equations of the correction phase of the filter, assigning the priori estate estimate $\hat{a}_k^-$ to the posteriori state estimate $\hat{a}_k$ and the priori estimation error covariance matrix P_k^- to the posteriori error covariance matrix P_k instead to zeroing the $Kalman\ gain$ K_k.

When a real or virtual occlusion happens, the immediate consequence is the growth of the measurement covariance matrix S_k.

4.1.3 Uncertainty Ellipses in state estimate and in sensor readings

The priori estimate error covariance P_k^- (see equation 4.17) can be showed as an uncertainty ellipsoid of the state estimate. The center of the ellipsoid is the state estimation mean and its volume represents the standard deviation from the mean value.

The innovation covariance matrix S_k (see equation 4.22) contains the variance of each element of the measurement vector and can be showed as an ellipsoid with center in the predicted measurement $\hat{z}_k^- = (x, y)^T$ and its shape is the covariance from the mean.

In the absence of measurements the uncertainty matrices grows, so the bigger axes of the error ellipse, the lower certainty in the estimation, and vice versa.

For example, in the case of a 2-dimensional Gaussian the uncertainty ellipse is represented as the intersections of the density function $f(x, y)$ with the parallel planes of plane $\{x, y\}$. We can observe different uncertainty ellipses sizes (in blue), according to different values of σ, in figure 4.7.

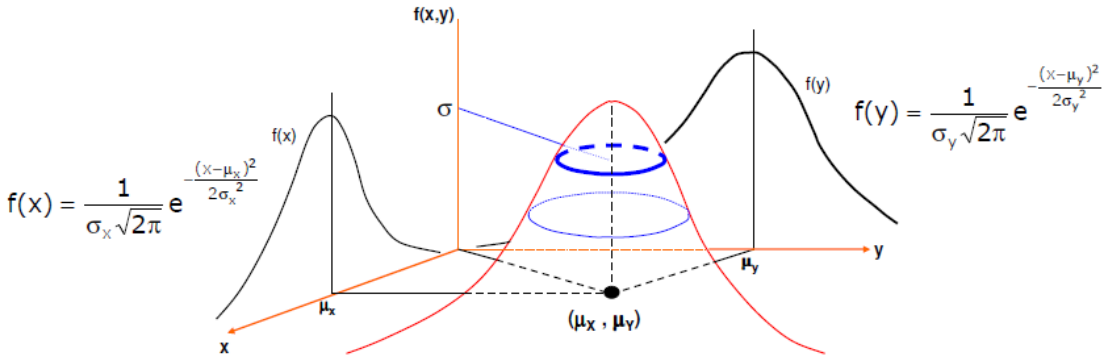


Figure 4.7: Uncertainty Ellipse of a two dimension Gaussian for predicted measurement

Considering a n-dimensional Gaussian random vector, with $Z = \mathcal{N}(\hat{z}_k, S_k)$ and $K_1 \in \Re$, the locus from which the pdf $f(z)$ is greater or equal to γ_1 is

$$\left\{ z : \frac{1}{(2\pi)^{n/2}\sqrt{|S_k|}} \left(-\frac{1}{2}\left[z-\hat{z}_k\right]^T S_k^{-1}\left[z-\hat{z}_k\right] \right) \geq \gamma_1 \right\} \tag{4.28}$$

With equivalent equation

$$\left\{ z : \left[z-\hat{z}_k\right]^T S_k^{-1}\left[z-\hat{z}_k\right]) \leq \gamma \right\} \tag{4.29}$$

where $\gamma = -2\ln((2\pi)^{n/2}\gamma_1|S_k|1/2)$ is n-dimensional ellipsoid centered at the mean $\hat{z}_k$ with axes aligned with the cartesian frame only if the covariance matrix S_k is diagonal.

The ellipsoid defined by equation 4.29 is the region of minimum volume that contains a given probability mass under the Gaussian assumption. When in equation 4.29 there is an equality rather than inequality the locus interpreted as the contours of equal probability is named *Mahalanobis distance* of the vector z to the mean $\hat{z}_k$

4.2 Probabilistic Data Association Filter

In target-tracking applications, the mechanism implemented for target detection can generate multiples measurements associated with a single object. In those cases, it is necessary to implement a *data association* technique to assign those measurements to a certain track.

The Probabilistic Data Association Filter (PDAF) solves the problem of data association of measurements by using a Bayesian approach, defining a validation region around the measurement prediction of each track (the uncertainty ellipse) and associating the measurements inside that validation region to the associated track weighting the influence of all these measurements in the innovation part of the filter as exposed in the following paragraphs.

4.2.1 Measurement Validation

Considering m_k the number of measurements, at each time step the sensors provides a set of measurements $z_j, j = 1, \cdots, m_k$. In figure 4.8 it can seen how only those measurements that

fall inside the validation gate of the predicted measurement will be retained for updating the state, the rest will be discarded, according to validation gate region defined by

$$\left\{ z : \left[z_j - \hat{z}^{t-} \right]^T S_t^{-1} \left[z_j - \hat{z}^{t-} \right]) \leq \gamma \right\} \tag{4.30}$$

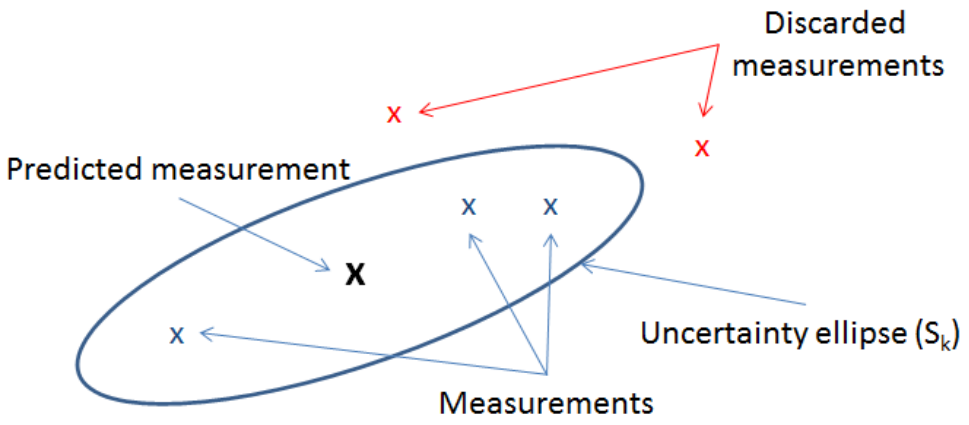


Figure 4.8: Validation gate derived from predicted measurement covariance S_k

For decision making purposes the error ellipse is used as the validation gate in data association algorithms where γ is the validation gate threshold corresponding to the gate probability P_G, which is the probability that the gate contains the true measurement if detected, and S_k is the covariance of the innovation corresponding to the true measurement.

The goal is to find the γ value according to certain probability we desire as *Gate Probability*.

n=1: $Prob\{z \text{ inside the ellipsoid}\} \quad -\frac{1}{\sqrt{2\pi}} + 2erf(\sqrt{\gamma})$ (4.31)

n=2: $Prob\{z \text{ inside the ellipsoid}\} \quad 1 - e^{-\gamma/2}$ (4.32)

n=3: $Prob\{z \text{ inside the ellipsoid}\} \quad -\frac{1}{\sqrt{2\pi}} + 2erf(\sqrt{\gamma}) - \sqrt{\frac{2}{\pi}}\sqrt{\gamma}e^{-\gamma/2}$ (4.33)

where n is the dimension of the evaluated vector and *erf(x)* is the error function, which is related with $\mathcal{N}(0,1)$, defined by:

$$erf(x) = \frac{1}{\sqrt{2\pi}} \int_0^x e^{-y^2/2} \mathrm{d}x \tag{4.34}$$

Considering the case n=2 the equation 4.32 can be solved with

$$\gamma = -2Ln(1 - P_G) \tag{4.35}$$

4.2.2 Requirements to be satisfied with PDAF

The requirements the PDAF algorithm needs to work properly are practically the same as Kalman Filter has, with the following additions:

1. Only one target of interest is present.
2. The track has been initialized.
3. At each time, only those measurements that fall inside the validation region around the predicted measurement are associated to the target in consideration.
4. The rest of the measurements outside the validation region are assumed to be false alarms or clutter originated, that can be modeled with uniform spatial distribution. The number of clutter or false alarms follows a Poisson distribution or a diffuse prior.
5. The target detections happens independently over time with probability P_d.

PDAF, with all these assumptions satisfied, is almost as simple as the basic Kalman Filter, being more effective in cluttered environments, with an increment of 50% approximately in computational cost.

In case the state and measurements are linear, the Probabilistic Data Association Filter will be based on Kalman Filter. On the other hand, if the state and measurements are non linear, the Probabilistic Data Association Filter will be based on Extended Kalman Filter. The equations for linear case follows:

Probabilistic Data Association Filter prediction equations

The prediction equations of PDAF are exactly the same as previously exposed for Kalman Filter.

Probabilistic Data Association Filter correction equations

Compute the Kalman Gain: $$K_k = P_k^- H^T (H P_k^- H^T + R)^{-1} \tag{4.36}$$

Compute the combined innovation: $$v_k = \sum_{i=1}^{m_k} \beta_{ik} r_{ik} \tag{4.37}$$

Udate the estimation with measurement z_k: $$\hat{x_k} = \hat{x}_k^- + K_k v_k \tag{4.38}$$

Update the estimation error covariance matrix: $$P_k = \beta_{0k} P_k^- + (1 - \beta_{0k}) P_k^c + \tilde{P}_k \tag{4.39}$$

The equations for PDAF are almost the same as for standard Kalman Filter except the use of the combined innovation in 4.38 where all the measurements are weighted according their relative position inside the validation gate, and the covariance associated with the updated state in 4.39.

In the expression 4.37 r_{ik} refers to each individual innovation as in 4.23, where

$$\beta_{ik} = \begin{cases} \dfrac{\mathcal{L}_{ik}}{1 - P_D P_G + \left(\sum_{j=1}^{m_k} \mathcal{L}_{ik} \right)^T} & i = 1, \cdots, m_k \\ \dfrac{1 - P_D P_G}{1 - P_D P_G + \left(\sum_{j=1}^{m_k} \mathcal{L}_{ik} \right)^T} & i = 0 \end{cases} \tag{4.40}$$

where $i = 0$ means that no measurement is correct, P_D is the target detection probability, and P_G is the gate probability that corresponds to 4.29, and

$$\mathcal{L}_{ik} \triangleq \frac{\mathcal{N}(z_{ik}; \hat{z}_k, S_k) P_D}{m_k / V_k} \tag{4.41}$$

is the likelihood ratio of measurements were originated from the target instead of the clutter. m_k is the number of measurements inside the validation gate of the current track according to 4.29 and V_k is the volume of the validation region when all the clutter meaurements has the same probability.

$$V_k = \pi \gamma |S_k|^{1/2} \tag{4.42}$$

In the covariance associated with the updated state expression (4.39), the covariance of the state updated with the correct measurement is

$$P_k^c = P_k^- - K_k S_k K_k^T \tag{4.43}$$

and the spread of the innovations term $\tilde{P}_k$

$$\tilde{P}_k \triangleq K_k \left[\sum_{i=1}^{m_k} \beta_{ik} r_{ik} {r_{ik}}^T - v_k v_k^T \right] K_k^T \tag{4.44}$$

4.3 Join Probabilistic Data Association Filter

The Join Probabilistic Data Association Filter (JPDAF) is an extension of PDAF that allows simultaneous tracking of several objects simultaneously, avoiding the situations where several tracks are locked into the same object. The key difference between both algorithms resides in how the association probabilities are computed. The PDAF assumes that all the measurements not associated with a specific track are false or clutter originated. On the other hand, JPDAF computes all the measurements probabilities jointly across all the active tarjets.

In order to simplify notation the time subscripts have been suppressed from all the equation that follows. Considering m_k the number of measurements and T_k the number of tracks. At each time step the sensors provide a set of measurements $z_j, j = 1, \cdots, m_k$. As well as PDAF, only those measurements that fall inside the validation gate of the predicted measurement for each target will be retained, the rest will be discarded.

As well as PDAF, the JPDAF algorithm calculates the conditional mean estimate $\hat{x}^t$ with combined weight innovation:

$$v^t = \sum_{j=1}^{m_k} \beta_j^t \hat{r}_j^t \tag{4.45}$$

where the innovation of each measurement is:

$$\hat{r}_j^t = z_j - \hat{z}^{t-} \tag{4.46}$$

And the probabilty of the j^{th} measurement belongs to the track t is defined by:

$$\beta_j^t = \begin{cases} \sum\limits_{\chi} P\left\{\chi | Z^k\right\} \hat{w}_{jt}(\chi) & j = 1, \cdots, m_k \\ 1 - \sum\limits_{j=1}^{m_k} \beta_j^t & j = 0 \end{cases} \tag{4.47}$$

Computation of the β_j^t factors depends on the construction of the validation matrix $\Omega_{m_k, T+1}$ that comprises all the targets T and all the measurements m_k with the form:

$$\Omega = [\omega_{jt}] = \begin{pmatrix} 1 & \omega_{1,1} & \omega_{1,2} & \cdots & \omega_{1,t} \\ 1 & \omega_{2,1} & \omega_{2,2} & \cdots & \omega_{2,t} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_{m,1} & \omega_{m,2} & \cdots & \omega_{m,t} \end{pmatrix} \begin{matrix} j = 1, \cdots, m_k \\ t = 0, \cdots, T \end{matrix} \tag{4.48}$$

Note that the first column of the validation matrix ω_{j0} is set to 1 indicating that measurement j could be originated by the clutter. For the remaining elements $\omega_{jt} = 1$ if the measurement j is inside the validation gate of the target t, otherwise $\omega_{jt} = 0$.

Once we had completed the Ω validation matrix we can generate the data association hypothesis Ω_i, by descomposing the Ω matrix according to the following restrictions:

- Each measurement can be originated only from a target or clutter

$$\sum_{t=0}^{T} \hat{\omega}_{jt}(\chi) = 1 \ \forall j \tag{4.49}$$

- No more that one measurement can be originated from a target

$$\sum_{j=1}^{m_k} \hat{\omega}_{jt}(\chi) \leq 1 \ \forall t = 1, \cdots, T \tag{4.50}$$

Considering the following example with two tracks and three measurement as represented in the figure 4.9. The validation matrix Ω and the hypothesis matrices $\hat{\Omega}$ considering the restrictions imposed by the algorithm follows:

$$\Omega = [\omega_{jt}] = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix} \qquad \hat{\Omega}(\chi_0) = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

$$\{z_1, z_2\} \Rightarrow \hat{\Omega}(\chi_1) = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} \quad \{z_1, z_3\} \Rightarrow \hat{\Omega}(\chi_2) = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{4.51}$$

$$\{z_2, z_3\} \Rightarrow \hat{\Omega}(\chi_3) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

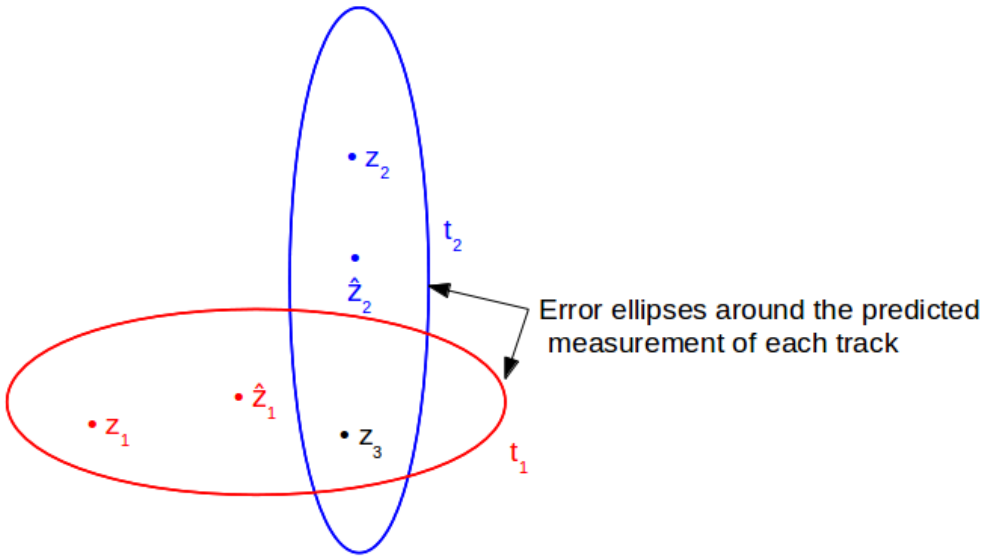


Figure 4.9: Association in JPDAF with two tracks and three measurements

Now we can define the measurement association indicator $\tau_j(\chi)$, the target detection indicator $\delta_t(\chi)$ and the number of false measurements in each event χ as $\phi(\chi)$.

- *measurement association indicator* becomes 1 if the measurement j falls inside the validation gate of any target:

$$\tau_j(\chi) = \sum_{t=1}^{T} \hat{\omega}_{jt}(\chi) \tag{4.52}$$

- *target detection indicator* will be 1 if the target t has any measurement inside its validation gate:

$$\delta_t(\chi) = \sum_{j=1}^{m_k} \hat{\omega}_{jt}(\chi) \tag{4.53}$$

- *number of false measurements in event* χ can be computed subtracting the measurement association indicator to the unit:

$$\phi(\chi) = \sum_{j=1}^{m_k} [1 - \tau_j(\chi)] \tag{4.54}$$

In the non parametric JPDAF we can compute the conditional probability of each feasible events χ with the observed measurements as:

$$P\left\{\chi|Z^k\right\} = \frac{\phi!}{cV^\phi}\prod_j\left\{\mathcal{N}[z_j^k;\hat{z}_{t_j}^k,S_{t_j}^k]\right\}^{\tau_j}\prod_t(P_D^t)^{\delta_t}(1-P_D^t)^{1-\delta_t} \tag{4.55}$$

where V is the volumen of the validation region, ϕ is the number of false measurements in event χ, and c is the normalization constant $\sum_\chi P\left\{\chi|Z^k\right\}\hat{\omega}_{jt}(\chi)$.

Chapter 5

Software Implementation

In this chapter a description of the implemented program is given for the three tracking techniques implemented. Three probabilistic filters were used to implement visual object tracking of multiple objects in two dimensions. In order to feed the tracking algorithms implemented, the objects were extracted from real images using color filter or learning background techniques. Also it was implemented a synthetic object generator that can be used with the each filter. The output of the filters execution shows the predicted position of each object and the associate uncertainty as an ellipsoid around it using superposed graphics over the original image.

The first filter implemented was Kalman Filter with the needed logic to handle simultaneous track of multiple objects. With the implementation of Probabilistic Data Association Filter and Join Probabilistic Data Association Filter it was possible to effectively track single or multiple objects in noisy environments.

The three probabilistic filters shared the same state and observation vectors. The vector state includes the 2D position on the screen with the speed of the object to be tracked, and the observation vector contains only the 2D position of the object to be tracked.

A GUI was also designed and implemented in order to provide a framework where the probabilistic filters could be checked and tunned.

5.1 General Design

A video tracking project must implement a first phase where the objects to be tracked are identified (Target location and localization phase) and a second part where the information

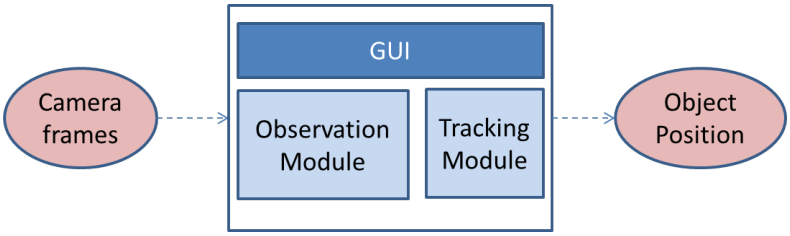


Figure 5.1: Black box data flow diagram

regarding the objects to be tracked with the dynamic associated to them are incorporated into the system (Filter and data association phase).

The general structure of the implemented program consists in a main loop where each frame is extracted from an external image provider (*cameraserver* process) and the input image is processed by the observation module. The output image shows the predicted position and its associate uncertainty in the form of an ellipsoid over the original images.

The observation module contains the logic for applying the different algorithms to extract the object centroids that will be used as measurements (or clutter depending the object extraction technique accuracy) in the different Kalman Filter based processes.

As shown in the image 5.1 the loop is composed of two well defined parts. The former regards the object recognition and refers to how to extract objects from scene images. It is represented in the *Observation Module*. The latter includes logic for the different probabilistic filters implemented.

Observation module implements different algorithms for extracting the objects from the input images: using HSV color filter or using learning background algorithms. It receives each frame from the video provider (either real images or simulated images) and transforms the image to find the Blobs through color filter or backgroud learning techniques, delivering the list of found centroids into the centroid structure.

The tracking module gets the list of centroids as observation and implements the tracking technique: Kalman / nKalman, PDAF/ nPDAF or JPDAF. After the filter step have been executed it generates output data that contains the target location prediction and the error ellipse associated with it. In the figure 5.2 we can observe the main interaction between such process.

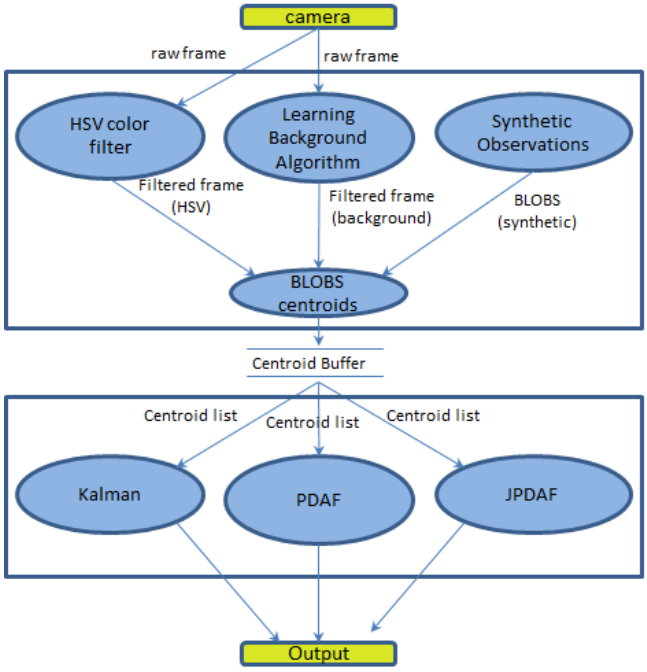


Figure 5.2: Data flow diagram with observation and tracking modules detailed

With the GUI module we can choose between synthetic or real images to feed the filters, the object extraction technique based of color filter or in a learning background approach or apply different morphological transformations if desired. We can also select the visual tracker to run, tune the main parameters of each one, such as the number of tracks, assigns detection probabilities, etc., save the tunned parameters to a configuration file, etc.

In summary, in each iteration a new frame is processed by applying the selected filters to the image, then the Blobs are extracted using less sophisticated approach such as color filter or using a better solution based on background learning algorithms, and finally all the Blobs centroids are stored to be processed by the probabilistic object tracker.

5.2 Observation Module

To properly extract objects from a video sequence, several methods for image segmentation have been implemented: the first one filters the image by colors of interest such as HSV filter techniques. The second one is based on detecting objects from background. The third generates blobs synthetically.

The objective of the observation module is to provide the minimum number of blobs per detected object. Ideally one blob per object, but the number of reported blobs will

vary depending of the implemented blobs extraction technique.

5.2.1 Object detection by HSV Color filter

HSV is one of the most common cylindrical-coordinate representations of color. It was developed in the 1970s and consists in rearrange the geometry of RGB in an attempt to be more intuitive and perceptually relevant than the cartesian (cube) representation, by mapping the values into a cylinder (or a cone) loosely inspired by a traditional color wheel. The angle around the central vertical axis corresponds to "Hue" and the distance from the axis corresponds to "Saturation". These first two values give the two schemes the 'H' and 'S' in their names. The height corresponds to a third "Value", the system's representation of the perceived luminance in relation to the saturation. The figure 5.3 shows the HSV color space cone.

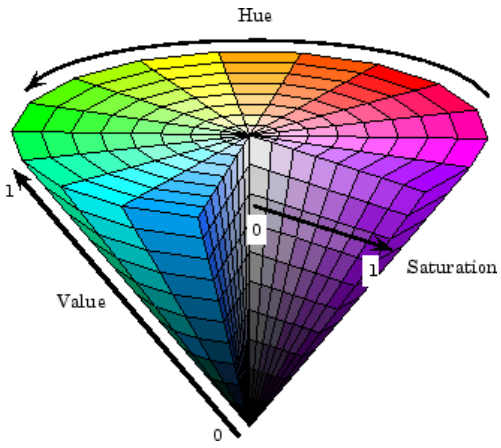


Figure 5.3: HSV color space

The HSV filter was selected as a first approach to extract objects from images due to its simplicity and because with this color space the image is less sensitive to intensity fluctuations than RGB.

In order to apply the HSV filter, the user can select in the GUI the maximum and minimum values for Hue, $Saturation$, and $Value$ parameters, and then the filtering method will select only those pixels in the image whose values fall into the selected range for each category.

In the figure 5.4 a rocket launch has been filtered by HSV. The objective was to filter the rocket tail and extract the found blobs, in order to track its trajectory. Applying the HSV parameters to the image filter by eliminating all the colors but the tail three blobs have been found, identified in the right frame of the image by blue circles.

Blobs detection based on HSV color filter

When filtering objects by HSV, for each frame, the program first selects only those pixels inside the color filter range, then converts the image to gray scale and extracts the detected

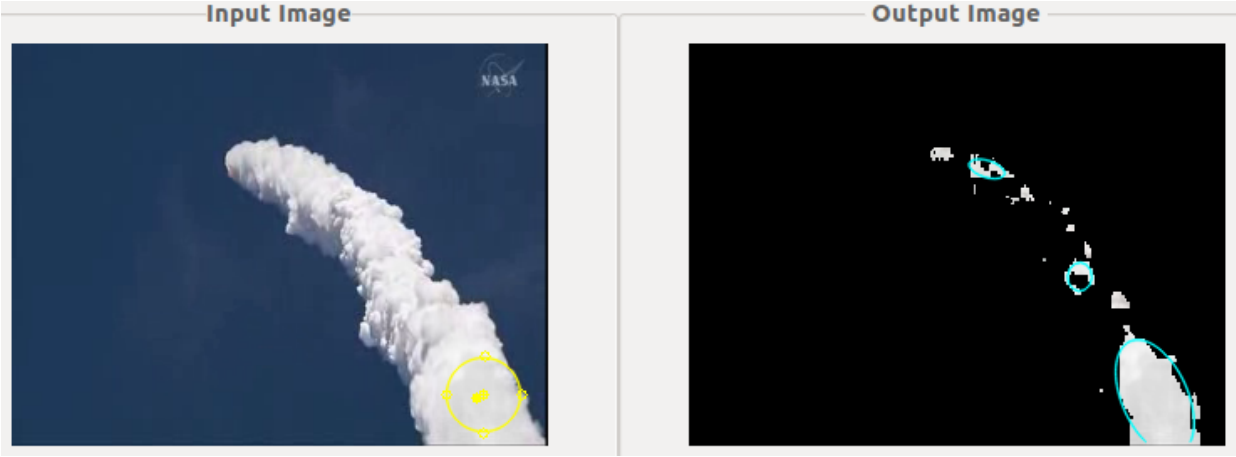


Figure 5.4: A STS-125 rocket color filtered by HSV

Figure 5.5: HSV filtering process

contours, finally inserts the centroid of each contour in the clutter structure, as shows the figure 5.5.

After the image has been filtered and converted to gray color space, it finds the contours by applying the algorithm 5.1, using the *findContours* function from *opencv*.

Once the contours have been identified, the program stores the blob centroids by calling the method

```
insert_point_in_clutter
  (Point(it->second->centroid.x, it->second->centroid.y))
```

```
void track::add_detected_point(Point p) {
  clutter.Z[clutter.elements] = *(Mat_<float>(2, 1) << p.x, p.y);
  clutter.assigned[clutter.elements] = -1;
  clutter.elements++;
}
```

Once all the detected centroids have been inserted into the clutter structure the program is ready to track the objects by the tracking module.

```
Data: input_image, output_image, show_image
Result: output
tracks.init_detected_points();
findContours(bimage, contours, CV_RETR_LIST, CV_CHAIN_APPROX_NONE);
for i = 0; i < contours.size(); i++ do
    Mat pointsf;
    Mat(contours[i]).convertTo(pointsf, CV_32F);
    RotatedRect box = fitEllipse(pointsf);
    if small_size(box) then
        continue;
    end
    ellipse(mat_image, box, Scalar(0, 255, 255), 1, CV_AA);
    image = mat_image;
    tracks.add_detected_point(box.center);
end
```

Algorithm 5.1: Blobs centroids extraction used with HSV color filtering

5.2.2 Object detection with background subtraction

In order to extract images in movement from a real video sequence, a background subtraction technique has also been implemented. Considering the fact that illumination often changes in real environments scenes, the algorithm to be implemented must be suitable to adapt to illumination changes.

A practical and efficient way to achieve this goal consists in recursively update the background model using temporal blending. This technique is also known as exponential forgetting,

$$B_t = \alpha I_t + (1 - \alpha)\, B_{t-1} \tag{5.1}$$

where B_t denotes the background image computed at time t, and the α parameter controls the speed of forgetting the old background information.

In the figure 5.6 we can observe the general schema used to implement the background exponential technique in order to extract the blobs.

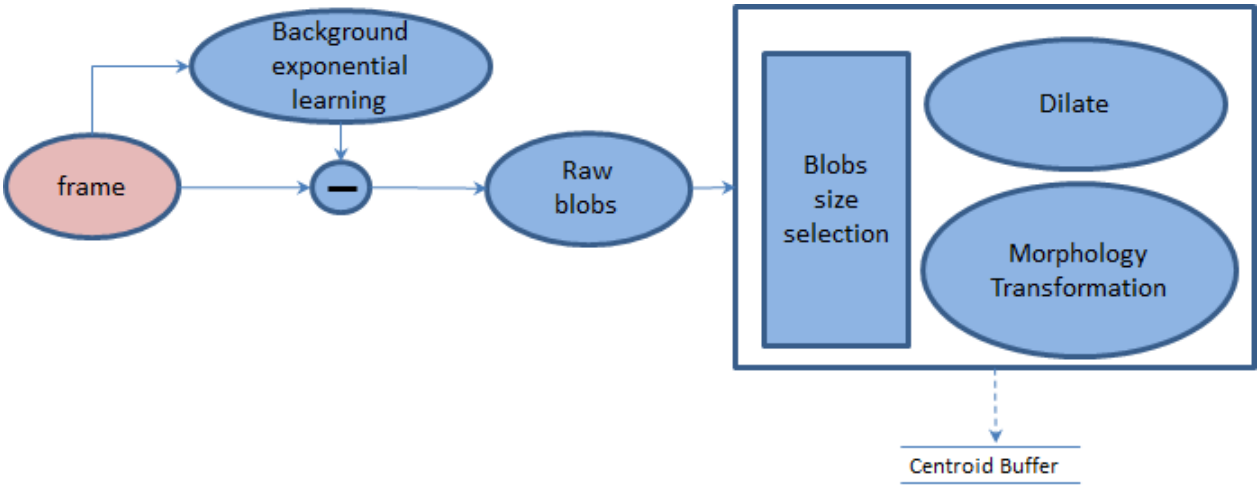


Figure 5.6: Background exponential schema

Data: $input\_image, output\_image$
Result: output
if *firstFrame(input_image)* **then**
  $B_t = input\_image$;
end
$B_t = (1 - \alpha)B_t + \alpha(input\_image)$;
$tracks.init\_detected\_points()$;
for $i = 0; i < (input\_image.witdh * input\_image.height); i++$ **do**
  diff = std::abs(B_t.data[i] - input_image.data[i]);
  if $diff < minDiff$ **then**
    output_image.data[i] = 0;
  else
    output_image.data[1] = 255;
  end
  `/* dilate operation if selected */`
  `/* morfology transformation if selected */`
  $cvb :: cvRenderBlobs(\cdots, foundBlobs, \cdots)$;
  for $const\_iterator\ it = foundBlobs.begin(); it! = foundBlobs.end()$ **do**
    `/* mask operation if selected */`
    $insert\_point\_in\_clutter(Point(it \rightarrow second \rightarrow centroid.x, it \rightarrow second \rightarrow centroid.y))$;
  end
end

Algorithm 5.2: Learning background based on exponential forgetting and blobs extraction

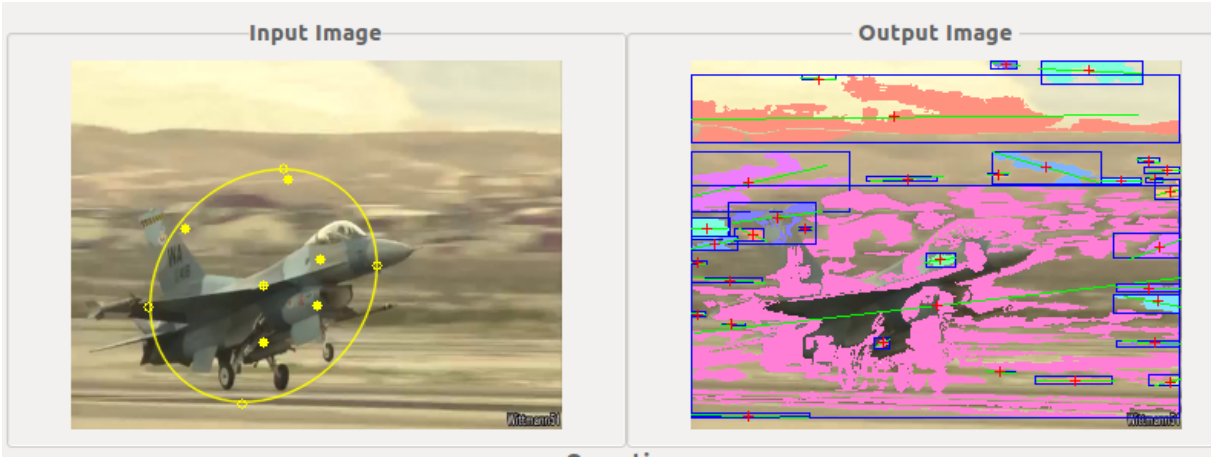


Figure 5.7: Blobs subtraction from background using learning algorithm

Although the algorithm 5.2 extracts the blobs in the image, and despite the blobs size can be selected, we can observe that multiple blobs that belongs to the same object were detected, making the tracking labor more complicated and expensive in computational terms. This can be seen in figure 5.7.

Dilating the detected blobs

In order to magnify the blobs detected, a dilating function has been used where we can select the size (x,y) of the square that contains the blob, as it can seen in algorithm 5.3. Combining this feature with the size selection of blobs to be detected allows us to reduce the number of blobs that belongs to the same object.

```
/* Insert instead the dilate comment in blobs detection algorithm   */
if dilate_box then
    Matkernel = cv :: Mat :: ones(cv :: Size(dilateX, dilateY), CV_8U);
    dilate(img_gray8_out, img_gray8_out, kernel);
end
```

Algorithm 5.3: Dilating the blobs detected

In the figure 5.8 we can observe how the number of blobs reported has been reduced, with the benefit of a less expensive PDAF or JPDAF tracking in computational terms.

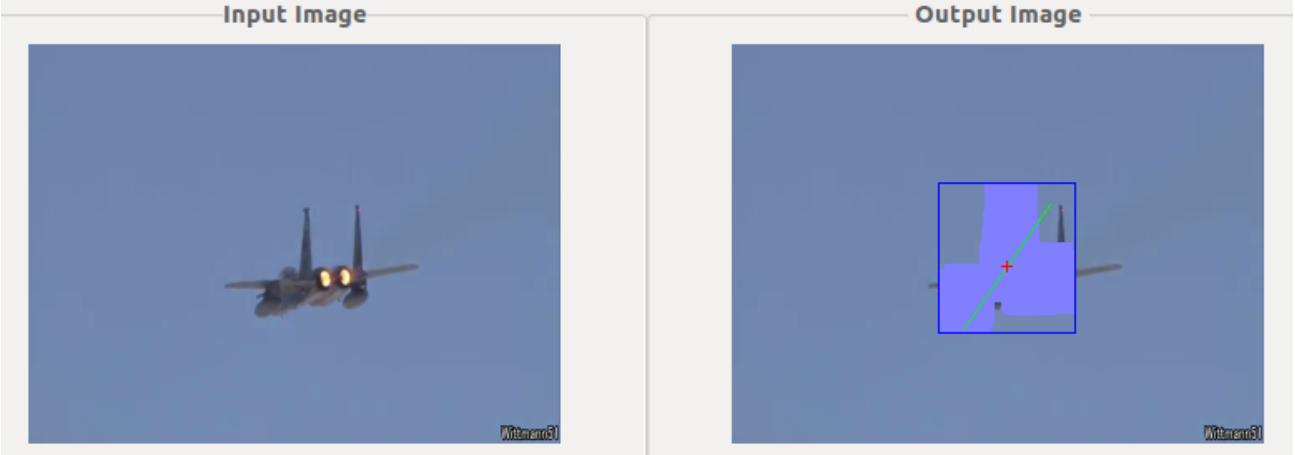


Figure 5.8: Blobs subtraction from background using learning algorithm and dilate

Applying morphology transformations to the detected blobs

To obtain an improved blobs extraction from background, a morphology transformation has been implemented for using with or without the dilate feature showed before, where it can been selected each one of the following transformations: ERODE, DILATE, OPEN, CLOSE, GRADIENT, TOPHAT and BACKHAT. The algorithm used is exposed in 5.4. If we have a scene with lots of mini blobs that belongs to the same object, after applying such transformations the resulting image will be just a little blurred, and the detection blobs functions will find less blobs linked to each object (see figure 5.8). We can also select the morphology rows and cols to be used with the morphology filter.

```
/* Instead the morphology comment in blobs detection algorithm      */
if morphology_box then
    IplConvKernel* structuringElement;
    structuringElement = cvCreateStructuringElementEx(morfology_rows,
    morfology_cols, cvFloor(morfology_rows / 2), cvFloor(morfology_cols / 2),
    CV_SHAPE_RECT, NULL);
    cvMorphologyEx(&iplInputGray, &iplInputGray, NULL, structuringElement,
    morfology_operation);
end
```

Algorithm 5.4: Aplying the morphology filter to the blobs detected

In the figure 5.9 we can see how to use the dilate and morphology filter in cascade to obtain better blobs.

Figure 5.9: Blobs subtraction from background using learning algorithm, dilate and morphology filter

Obtaining a better definition of the detected blobs by masking

By invoking the cvRenderBlobs twice in cascade we obtain a smoother blob detection, obtaining a better definition. This was done by the algorithm 5.5, with the observable results in the figure 5.10.

```
/* Instead the mask comment in blobs detection algorithm                    */
if mask_box then
    IplImage * mask = cvCreateImage(cvSize(iplNormal→width,
    iplNormal→height), IPL_DEPTH_8U, 3);
    for const_iterator it = foundBlobs.begin(); it! = foundBlobs.end() do
        cvb::cvRenderBlob(iplLabel, (*it).second, iplNormal, mask, 1, cv::Scalar(255,
        255, 255));
        insert_point_in_clutter(Point(it → second → centroid.x, it → second →
        centroid.y));
    end
end
```

Algorithm 5.5: Applying a mask to the detected blobs

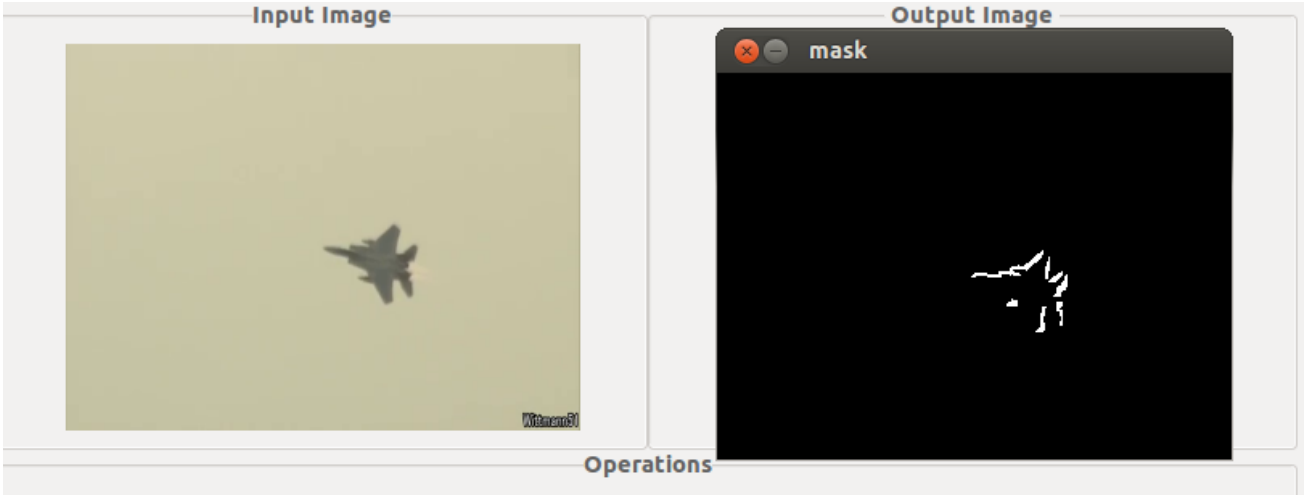


Figure 5.10: blobs subtraction from background using learning algorithm and masking

Using these blobs selection features jointly allows us to obtain blobs as required to better fit the tracking process needs, reducing the complexity of the calculations needed to execute the PDAF or JPDAF algorithms, resulting in a better response time and overall performance of the program. The figure 5.11 shows a single blob detected linked to the aircraft we want to track .

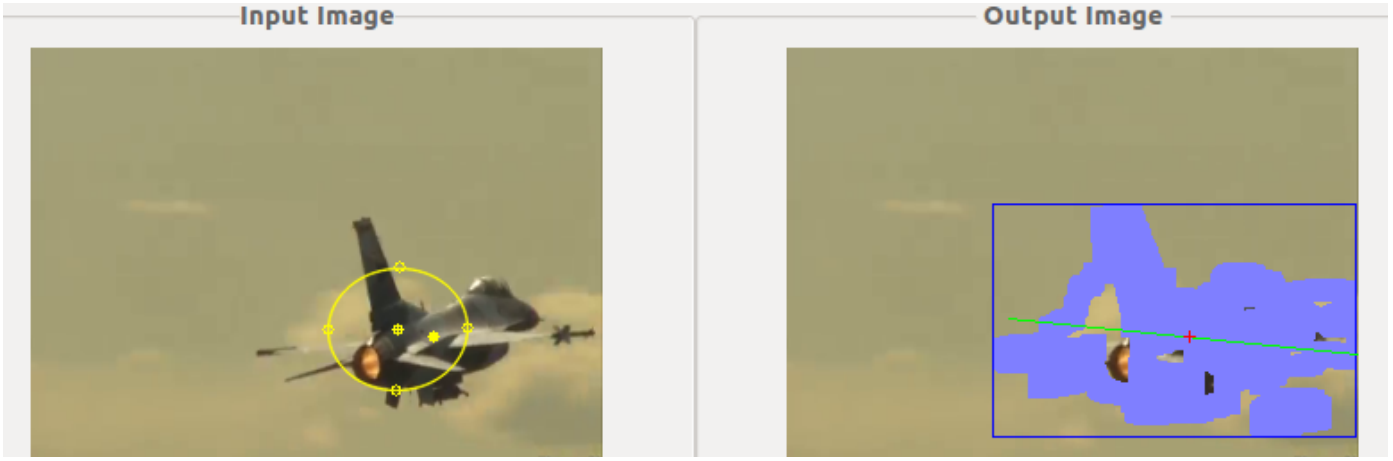


Figure 5.11: Blobs subtraction from background using learning algorithm, dilate, masking and morphology

5.2.3 Synthetic Objects

The generation of synthetic objects was a fundamental part in the development of this master thesis, because it provides a way to better test and refine the different tracking algorithms that will be described later.

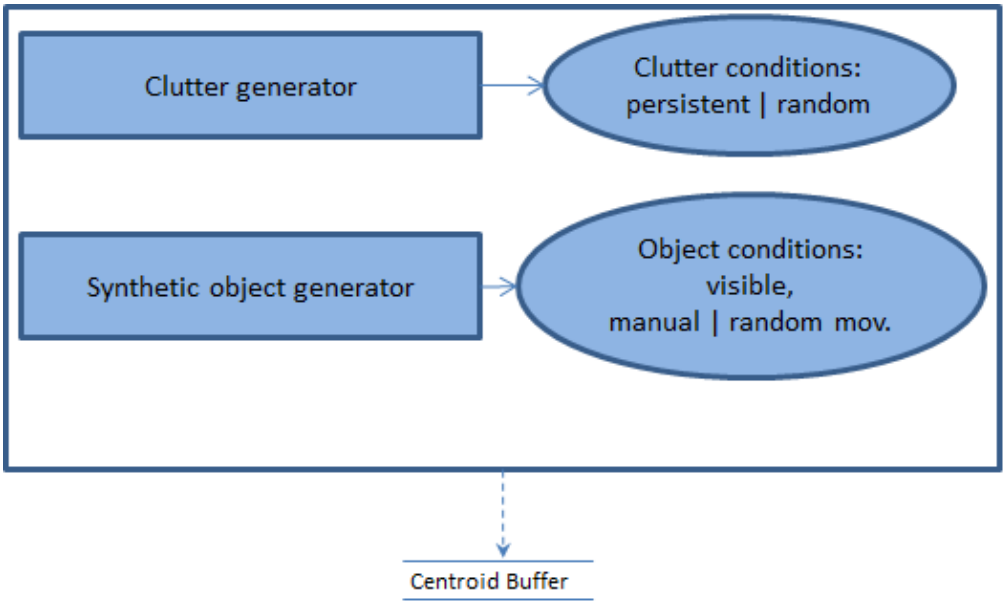


Figure 5.12: General schema of synthetic object generator

Using the synthetic generation of objects we can simulate one o more objects in the scene, clutter conditions with persistent or aleatory noise, occlusions in the objects and random or selected movements of the synthetic objects. The general schema used to implement the synthetic object simulator is in figure 5.12.

Simulated objects are represented as single points for simplicity, as shown in figure 5.13. The simulator module allows us to generate clutter conditions, simulate persistent noise, when the extra points generated are fixed, or random noisy conditions when the extra points are moving randomly, as showed in figure 5.14.

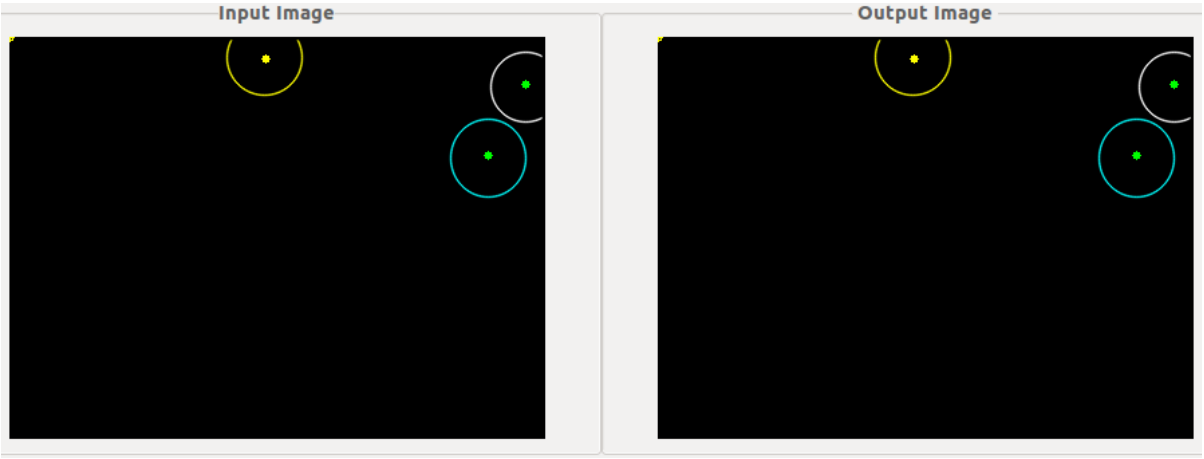


Figure 5.13: Synthetic objects

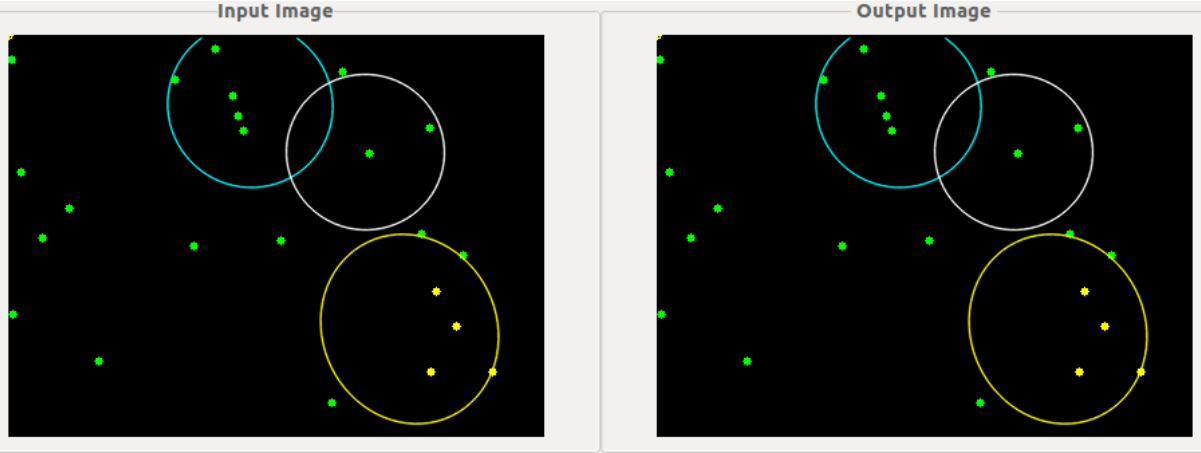


Figure 5.14: Generation of clutter conditions using synthetic objects

5.3 Tracking Module

This section will describe the system model used to implement the visual tracking module, showing the dynamic of the objects to be tracked, the state and observation vectors and noise models used.

5.3.1 State and Observations

The dynamic of the objects in the image to be tracked has been modeled considering the general movement equation, where the position of the object to be tracked is expressed in terms of previous position, speed and time. Considering we are tracking objects in a plane, the position and speed will have two dimensions, with coordinates x, y and v_x, v_y respectively. Therefore the state of the system will be represented with a $4x1$ matrix:

$$X_k = \begin{pmatrix} x \\ y \\ \mathrm{v_x} \\ \mathrm{v_y} \end{pmatrix}$$

The system will be modeled with constant speed, according to the next expression, where X_k is the state vector, F is the transition matrix that address the evolution of the system from the previous state and G_k is the auxiliary matrix that should be used with the acceleration vector A_k (see equation 4.9). Because the acceleration is not being considered in this work the G_k will be the zero matrix (the possible acceleration will be modeled as noise):

$$X_k = FX_{k-1} + G_k A_k = FX_{k-1}$$

$$\begin{bmatrix} x_k \\ y_k \\ v_{x_{k-1}} \\ v_{y_{k-1}} \end{bmatrix} = \begin{bmatrix} 1 & 0 & \triangle t & 0 \\ 0 & 1 & 0 & \triangle t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{k-1} \\ y_{k-1} \\ v_{x_{k-1}} \\ v_{y_{k-1}} \end{bmatrix} = \begin{bmatrix} \mathrm{v}_{\mathrm{x}_{k-1}} t + x_{k-1} \\ \mathrm{v}_{\mathrm{y}_{k-1}} t + y_{k-1} \\ \mathrm{v}_{\mathrm{x}_{k-1}} \\ \mathrm{v}_{\mathrm{y}_{k-1}} \end{bmatrix}$$

In this work I make three assumptions than can help to better assign the measurements of each target:

- The **position** of a target will be relatively unchanged between two consecutive frames.
- The **speed** of a target will be relatively unchanged between two consecutive frames. The acceleration will be modeled as noise.
- The **movement direction** will be relatively unchanged between two consecutive frames.

The measurements (observations) of the real or synthetic objects will have only two coordinates indicating its position in the image with coordinates x, y. The measurement matrix $Z_k = [x\ y]^T$ represents the measurements of the modeled system. The observation model matrix H, that relates the state X_k with the measurements Z_k (see equation 4.10) has been initialized as follows:

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

The above expressions and matrices are the same for the three probabilistic filters implemented (KF, PDAF and JPDAF) because the system dynamic model is the same for all them.

5.3.2 Adjusting the Filter Parameters

Sensor noise covariance matrix R

The measurement noise covariance matrix is quite easy to calculate considering it is the standard deviation of the sensor squared, or the variance of the sensor readings.

$$R = \begin{bmatrix} {\sigma_x}^2 & 0 \\ 0 & {\sigma_y}^2 \end{bmatrix} = \begin{bmatrix} 3.17 & 0 \\ 0 & 3.17 \end{bmatrix}$$

Instead of analyze multiple measurements in order to calculate the variance of each one, a fixed value for the sensor noise covariance matrix had been chosen. For the majority of the video samples used in this master thesis, a value of $\sigma_x^2 = \sigma_y^2 = 3.17$ offers a good performance.

Process noise covariance matrix Q_k

The possible acceleration of the objects to be tracked will be considered as noise, modeled by the process noise covariance matrix.

$$Q_k = \begin{bmatrix} \sigma_a^2 & 0 & 0 & 0 \\ 0 & \sigma_a^2 & 0 & 0 \\ 0 & 0 & \sigma_a^2 & 0 \\ 0 & 0 & 0 & \sigma_a^2 \end{bmatrix} = \begin{bmatrix} 10 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 \\ 0 & 0 & 10 & 0 \\ 0 & 0 & 0 & 10 \end{bmatrix}$$

The values $\sigma_{a_x} = \sigma_{a_y} = 10$ works fine for the majority of the samples used in this master thesis.

Estimation error covariance matrix P_k

The estimation error covariance matrix can be set to eventually any value with the exception of the zero matrix. In such case the filter does not consider the Kalman gain K_k and therefore the filter does not correct the estate and only takes the value of the predictions.

Selecting an initial estimation error covariance matrix to the identity matrix, makes the filter converges quickly.

$$P_0 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Depending of the accuracy of each filter technique implemented the overall performance of the tracking algorithm will vary. The more accuracy of the filter the less false measurements detected, and therefore the algorithm will be more agile to achieve the target tracking. New coherent observations will reduce the covariance. Conversely, new noisy measurements will expand it. Occlusions will prevent the update equations and therefore the covariance error will grow.

5.3.3 Kalman Filter for object tracking

The implemented solution allows to select how many objects will be visually tracked. In case we have only one object to track it is possible to detect multiple sub blobs that belongs to the same object, due the object extraction techniques implemented. Because of this it is needed an approach to assign the blob centroids to the current track, although a single track is in action.

Association Algorithm in Kalman Filter

When Kalman filter option is selected, the implemented data association algorithm assigns the closed measurement available in the clutter structure to each active track by calculating the Mahalanois distance between the blob centroid in consideration and the predicted position of each track, as showed in algorithm 5.6.

Multiple Kalman Filter tracker

The Kalman Filter does not resolve the multiple object tracking itself, because each track computes only the estimated position considering the actual measurement and the previous state for the current track. When multiple Kalman Filters are used to simultaneously track several objects a problem arise when trying to assigns a measurement to each track. A simple approach to deal with this could be to assign the closest measurement to the

```
float pdaf::Mahalanois(Mat Zk) {
        Mat pZkZk;
        Mat Mahal;

        pZkZk = Zk - pZk;
        Mahal = (pZkZk.t() * Sk.inv() * pZkZk);
        return (sqrt(Mahal.at<float>(0, 0)));
}
```

Algorithm 5.6: Mahalanois distance algorithm in KF association

predicted state in case that at least one measurement falls inside the uncertainly ellipse, or to assign the closest one to the uncertainly ellipse projection of each track in case all candidates measurement fall outside the uncertainly ellipse. Due the Kalman Filter does not implement any mechanism to prevent two or more tracks are locked with the same blob centroid, each time a clutter point have been assigned to an active track it is removed from the clutter list, so a previously assigned clutter point can not be considered as elective for a new target.

The multi track algorithm (showed in algorithm 5.7) has been implemented as an array of the specific single track filter, adding the control logic that allows the track creation or deletion. In each filter there is a loop that runs over the maximum number of targets, following the same equation sequence the simple track has but implementing the logic control for counting the actual number of active tracks. The creation of a new track, or deletion of a current track if there is no measurements associated to it over limited time, is done inside the association function. If the track in consideration has at least one measurement assigned to it the track is considered to be active. On the other hand, if the track has no points assigned to it for consecutive 50 iterations the track is removed from the active track list and it is no longer considered for Kalman Filter calculations.

The algorithm 5.8 shows the control logic for $association\_KF()$ method that controls the track creation or deletion depending if there are measurements inside the validation gate of the track in consideration.

In the figure 5.15 we can see two Kalman Filter tracks with the closest measurement assigned to each track, with stolen mechanism between them.

Data: detected points are in tracks.clutter
Result:
$active\_tracks = 0$;
for $track\_idx = 0; track\_idx < MAX\_TRACKS; track\_idx++$ **do**
  if *track_active(track_idx)* **then**
    $active\_tracks++$;
  end
  if $active\_tracks < MAX\_TRACKS$ **then**
    break;
  end
  `/* Prediction equations: state, errorCov and measurement */`
  `/* Correction equations: InnovCovariance and KalmanGain */`
  association_KF(track_idx);
  if *track_free(track_idx)* **then**
    continue;
  end
  `/* Correction equations: updateEst., uptadeEstErrorCov */`
  error_ellipse;
  paint_validation_gate_points;
end

Algorithm 5.7: Multi Kalman Filter implementation

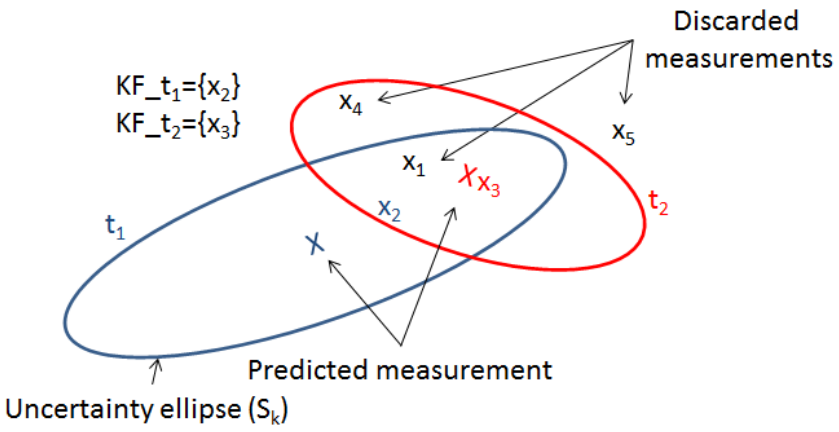


Figure 5.15: Kalman Filter association mechanism implemented

```
Data: track_idx
Result: output
active_tracks = 0;
for obs_idx = 0; obs_idx < tracks.clutter_elements; obs_idx++ do
    mahal_min=Mahalanois(track_idx,tracks.clutter.Z[obs_idx]);
    if mahal_min then
        /* associate the obs with lowest mahal distance                */
        /* there's a measurement assignment to the track               */
        /* add assigned point to the local clutter                     */
        tracks.KF[track_idx].set_status_active();
    else
        /* missing association for the actual track                    */
        num_missings++;
        if (track_active(track_idx) & missings ≥ 50) then
            tracks.remove_track(track_idx);
        end
    end
end
```

Algorithm 5.8: Association in nKalman implementation

5.3.4 Object tracking with PDAF

Association algorithm in PDAF

Unlike the Kalman Filter, either with single or multiple tracks, where a single measurement is assigned to a unique track using Mahalanois distance between the candidates and the tracks with a measurements stolen mechanism between them, in PDAF implementation all measurements that fall into the validation gate of a track are assigned to it.

As well as in the Kalman Filter approach, when PDAF filter has been selected the association algorithm runs after the Kalman gain has been calculated by calling the method:

```
tracks.association_PDAF(track_idx, shared_points_box);
```

When considering the PDAF approach, the validation gate is used to assign all the measurements that fall inside for each track, using the algorithm 5.9.

Multiple Probabilistic Data Association Filter Tracker

The PDAF algorithm has been designed to track a single object, considering from all measurements registered only those that falls into the validation region of the current track, and dropping the rest.

The common approach for tracking multiple objects with *PDAF* consists in implementing one *PDAF* track for each object to be tracked. When multiple tracks are considered, there is a risk that the validation region of several tracks overlap, originating persistent interference that can invalidate the association of measurements to the right track, and therefore, originating a track missing, or having multiple tracks locked into the same object.

To avoid that several PDAF tracks will be locked with the same set of measurements, once a measurement has been associated to a track, it is removed from the clutter structure in order that measurement will not be available to be associated with any other track in the same frame time.

The main algorithm to implement multi track with Probabilistic Data Association Filter (showed in 5.11) is almost the same as used with single track PDAF except in the association function (showed in 5.10) that ultimately determines the creation or deletion of the tracks through the use of *ValidationGate* function.

```
void pdaf::ValidationGate(&create_track, &remove_track
{
  Mk.elements = 0;
  for (int i = 0; i < clutter.elements; i++) {
    pZkZk = clutter.Z[i] - pZk;
    V = pZkZk.t() * Sk.inv() * pZkZk;
    if (abs(V.at<float>(0, 0)) <= C2) {
    // founded measurements inside the Sk of the current track
      Mk.elements++;
      num_missing = 0;
      create_track = true;
    }
  }
  if (Mk.elements == 0) {
  // measurements not found for actual track
    num_missing++;
    num_assigns = 0;
    if (num_missing > 50) remove_track = true;
  }
}
```

Algorithm 5.9: Validation gate used with PDAF

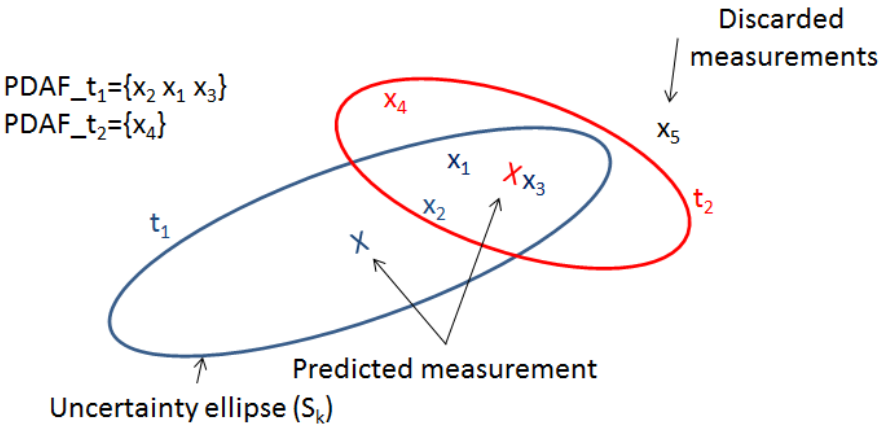


Figure 5.16: PDAF association mechanism implemented

In the figure 5.16 we can see two PDAF tracks in action where the first one (t_1) considers all the measurements that fall inside its validation gate (x_2, x_1, x_3), meanwhile the second one (t_2) only considers the measurement x_4 due the stolen mechanism implemented between them.

```
void track::association_PDAF(int track_idx, bool shared_points)
  bool create_track, remove_track;

  // copy the entire clutter to the actual KF
  KF[track_idx].clutter

  // validation region filter all the points according to the
  // covariance matrix innovation Sk
  KF[track_idx].ValidationGate (create_track, remove_track);

  if (remove_track) this->remove_track(track_idx);
  if (create_track) KF[track_idx].set_status_active();
```

Algorithm 5.10: Association algorithm implemented with PDAF

When a track has points inside its validation region is considered to be alive. On the other had, if a track has missed points for 50 consecutive frames is considered to be died and no longer is used.

The Validation gate method runs over all the entire clutter structure for each track, computing the volume of the innovation error ellipse with regards the predicted point and the actual measurement in consideration. If the computed value is less or equal than the value associated with a probability of 0.9999 ($\gamma = 18.4$) then the point is considered to be associated to that track, and the track is considered to be alive because it has at least one assigned point.

Finally, if there are measurements that were not associated to any available track and the distance from the first measurement is greater than the minimum track distance specified by the variable *minDist* a new track is created and it will be compute in the next iteration.

```
active_tracks = 0;
for track_idx = 0; track_idx < MAX_TRACKS; track_idx++ do
    if track_active(track_idx) then
        active_tracks++;
    end
    if active_tracks < MAX_TRACKS then
        break;
    end
    /* Prediction equations:  state, errorCov and measurement        */
    /* Correction equations:  InnovCov and K. Gain                   */
    association_PDAF(track_idx);
    if track_free(track_idx) then
        continue;
    end
    updateEstimation;
    updateEstimationErrorCov;
end
for track_idx = 0; track_idx < MAX_TRACKS; track_idx++ do
    if track_free(track_idx) then
        if num_tracks_detected < num_BLOMs then
            if distace to any track > minDist then
                set track as active;
            end
        else
            continue;
        end
    end
end
```

Algorithm 5.11: Multitrack PDAF algorithm

5.3.5 Join Probabilistic Data Association Filter

Association Algorithm in JPDAF

When the JPDAF is used, the association algorithm works exactly the same as PDAF does, assigning the measurements from the clutter structure that falls into the validation region (the volume of the innovation covariance matrix according to certain probability) of the track in consideration, with regards the predicted measurement.

As well as in PDAF implementation, if at least one measurement has been assigned to the track in consideration the track is treated as active, and on the other hand if there is not clutter assigments to the track in 50 consecutive frames, the track is removed from the active track list.

Once the number of active tracks is known, the event matrices are calculated in order to compute the JPDAF algorithm using the method.

```
tracks_jpdaf.eventMatrix_JPDAF(tracks_jpdaf.Omega.num_tracks());
```

The above method builds the Ω matrix that represents all the possible associations between all the validated points and all the active tracks. The detail of that method is showed in algorithm 5.12.

```
measurements = Omega.num_validated_measurements();
// Build the base event matrix with dimensions:
// measurements x tracks +1
Omega.init_event_matrix(measurements, num_tracks);
for (track_idx) {
    int t = Omega.get_track_number(track_idx);
    if (!track_idx.status_active())
       continue;
     for each(validated_points (j))
     {
       Omega.valMatrix.M[0].at<float>(j, 0) = 1;
       Omega.valMatrix.M[0].at<float>(j, track_idx + 1) = 1;
     }
}
```

Algorithm 5.12: Building the Ω matrix that represents all the associations

After the Ω matrix has been built, it is time to construct the feasible event matrices $\hat{\Omega}_i$ as showed in the example (4.51), considering the following assumptions:

- Each measurement has exactly one source or clutter originated
- No more than one observation can be originated from a target

The matrices $\hat{\Omega}_i$ can be constructed by scanning the Ω matrix and selecting one nonzero element for each row and one nonzero element for each column, with the exception for the column $t = 0$, as showed in algorithm 5.13. The feasible events matrices $\hat{\Omega}_i$ will be used in the conditional mean estimates with combined weight innovation (see equation 4.45) by using the probability of each j^{th} measurement belongs to the track t under consideration as defined by equation 4.47

Tracks handle in JPDAF

Due the nature of the JPDAF assumes the multi track itself, the general algorithm used in the previous section has been adapted in order the event matrices for all the active tracks are available when the join probabilities were computed. So after the measurements were associated to each track according its validation gate the algorithm computes the event matrices according to section 5.3.5 in page 69

Finally, if there are measurements that were not associated to any available track and the distance from the first measurement is greater than the minimum track distance specified by the variable *minDist* a new track is created and it will be computed in the next iteration. The algorithm implemented to handle the tracks creation or deletion is showed in 5.14. In the figure 5.17 we can see two JPDAF tracks in action, where each track considers all the measurements that fall inside its validation gate.

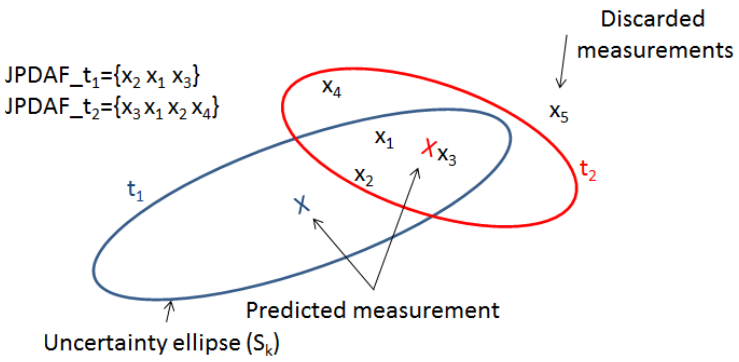


Figure 5.17: JPDAF association

```
for (int t = 0; t < tracks; t++) {
  if (t == 0) {
  // Matrix when no measurements belongs to any track
    for (int j = 0; j < measurements; j++)
      Chi(j, 0) = 1;
    if (only one track active) {
      Chi = Mat::zeros(measurements, Omega.
          get_num_cols_event_matrix(), CV_32F);
      for (int j_t1 = 0; j_t1 < measurements; j_t1++) {
      // for each measurement in Omega matrix
        Chi.(j_t1, t + 1) = Omega.get(0, j_t1, t + 1);
        if (Chi.(j_t1, t + 1) == 1) Chi.(j_t1, 0) = 0;
        else Chi.(j_t1, 0) = 1;
      }
    }
    continue;
  }
  // now t >= 1
  for each measurement j_t1 in Omega matrix in the track 1 {
   for each measurement j_t2 in Omega matrix in the track 2 {
      if ((Omega(0, j_t1, t)) && (Omega(0, j_t2, t + 1)) &&
          (j_t1 == j_t2))
        continue;
      if ((Omega(0, j_t1, t)) || (Omega(0, j_t2, t + 1))) {
        for (int j_t0 = 0; j_t0 < measurements; j_t0++) {
          Chi.(j_t0, 0) = 1;  // t0 cols = 1
        }
        Chi.(j_t1, t) = Omega.get(0, j_t1, t);
        Chi.(j_t2, t + 1) = Omega.get(0, j_t2, t + 1);
        if (Chi.(j_t1, t)) Chi.(j_t1, 0) = 0;
        if (Chi.(j_t2, t + 1)) Chi.(j_t2, 0) = 0;
      }
    }
  }
}
```

Algorithm 5.13: Feasible events building algorithm for JPDAF

```
Data: detected points are in tracks.clutter
Result:
active_tracks = 0;
for track_idx = 0; track_idx < MAX_TRACKS; track_idx++ do
    if track_active(track_idx) then
        active_tracks++;
    end
    if active_tracks < MAX_TRACKS then
        break;
    end
    /* Prediction equations:  state, errorCovPred, measurement      */
    /* Correction equations:  InnovCov, KalmanGain                  */
    association_JPDAF(track_idx);
end
tracks.eventMatrix_JPDAF;
for track_idx = 0; track_idx < MAX_TRACKS; track_idx++ do
    if track_free(track_idx) then
        if num_tracks_detected < num_BLOMs then
            if distace to any track > minDist then
                set track as active;
            end
        else
            continue;
        end
    end
    updateEstimation;
    updateEstimationErrorCov;
end
```

Algorithm 5.14: Multitrack JPDAF algorithm

5.4 GUI module

The GUI module has been developed to better interact with the different visual trackers implemented. With this module we can select how to extract the objects from the video sequences, tune the filter parameters to modify the operation of the filters, show the results of the tracking over multiple objects in real time and also activate the simulator tool where we can generate synthetic objects in order to test the filters more in depth.

We can distinguish two clear parts in the Graphical User Interface as it shown in the figure 5.18. The upper zone includes a couple of frames where the left one shows the original images the application received from the camera server component and the output of the tracking process, and the right one shows the intermediate transformations required to tracks the objects.

Figure 5.18: General view of GUI

In the image 5.19 it can be seen how the right frame shows the detected blobs and the left one shows a couple of tracks over the ping pong players.

In the middle left we can find the *Image filter* section as showed in figure 5.20, where a check button allows to select filtering the image by HSV color or locating the blobs using background extraction techniques, as explained in section 5.2. If using the

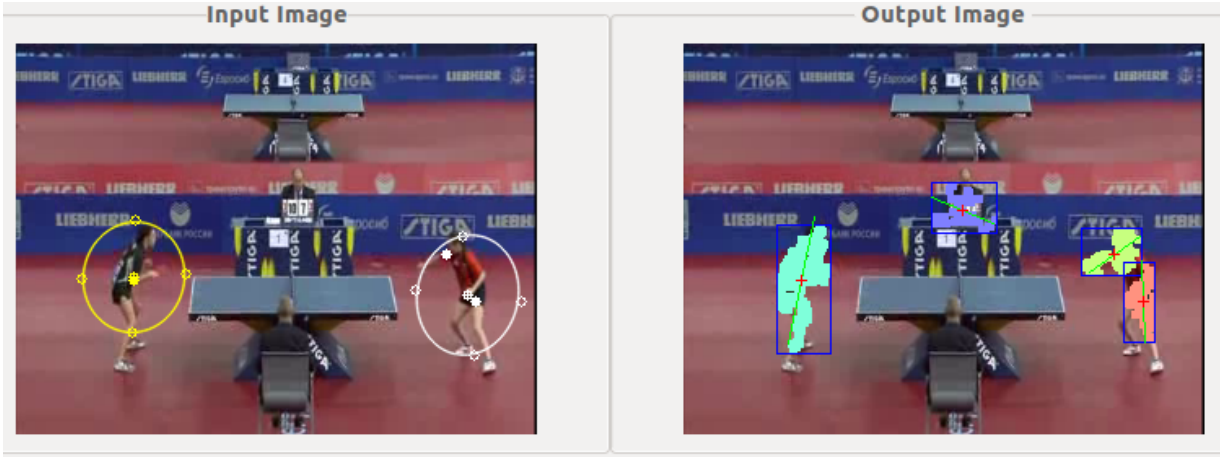


Figure 5.19: GUI frames showing transformations and the resulting track

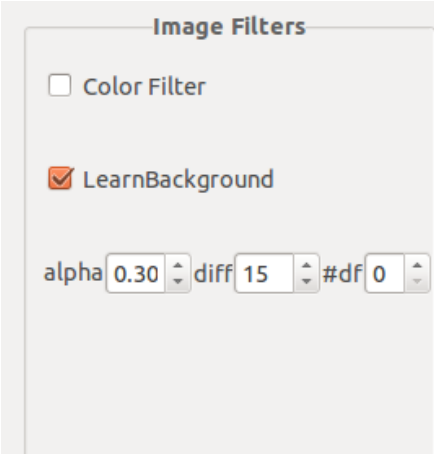


Figure 5.20: GUI image filters

LearnBackground we can select the *alpha*, $diff$ and $\#df$ parameters to control the behavior of the background extraction process. When selecting the *Color Filter* we can select the HSV range in the bottom of the GUI to control the filtering process.

In the middle of the window we can access to the *Feature Detection controls*, as showed in figure 5.21. Through this control panel we can detect the blobs filtered by the *Image filter framework*. If the image was filtered by color, then we need to check the button *Detect Blobs (color filtered)*. On the other hand, if the *Learning Background* button was selected, we need to select the *Detect blobs (background)*.

When the blobs are detected by background learning method, we can also select the minimum and maximum area of the detected blobs or apply different morphology transformations to better aisle the detected blobs.

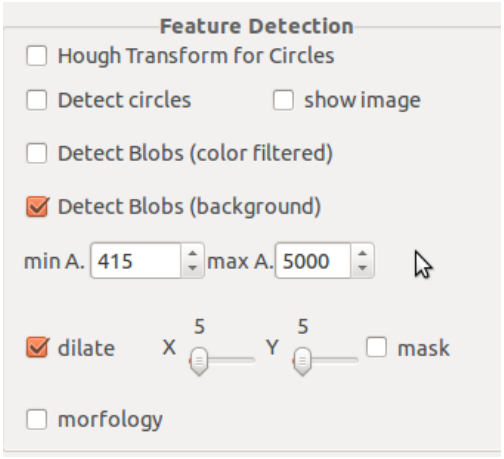


Figure 5.21: GUI features detection

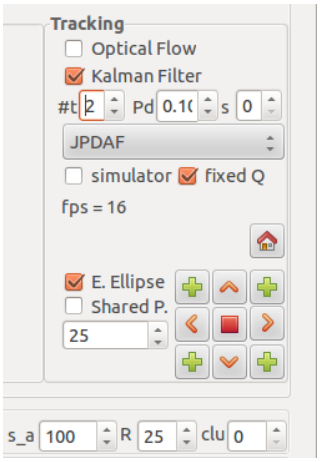


Figure 5.22: GUI tracking framework

The tracking framework, showed in figure 5.22 allows us to select if working with real images or with synthetic objects (switching the *simulator box*).

Activating the *Kalman filter* check box we start the tracking method (Standard, PDAF or JPDAF) selected by the below *combo box*, indicating the number of tracks to start with the *#t spin button*. The *Pd spin button* allows us to specify the target probability detection to be used by the PDAF or JPDAF.

With the *R spin button* we can inject the measurement process noise values into the filter process ($\sigma_x = \sigma_y = R_k$).

Drawing the error ellipse associated with certain probability

The uncertainty ellipse can be showed selecting the *E. Ellipse check button* into the tracking panel. If the *error_ellipse_box* has been selected in the GUI the error ellipse corresponding

to the innovation covariance is showed centered in the predicted state. The explanation about how to draw the error ellipsoids follows.

Considering the covariance matrix of the innovation corresponding to the true measurement

$$S_k = \begin{pmatrix} \sigma_x^2 & \sigma_{xy} \\ \sigma_{xy} & \sigma_y^2 \end{pmatrix} \tag{5.2}$$

the angle of the error ellipse is defined by

$$\sigma_{x'}^2 = \sigma_x^2 \cos^2\theta + 2\sin\theta\cos\theta\sigma_{xy} + \sigma_y^2 \sin^2\theta \tag{5.3}$$

$$\sigma_{y'}^2 = \sigma_x^2 \sin^2\theta + 2\sin\theta\cos\theta\sigma_{xy} + \sigma_y^2 \cos^2\theta \tag{5.4}$$

$$\tan(2\theta) = \frac{2\sigma_{xy}}{\sigma_x^2 - \sigma_y^2} \tag{5.5}$$

Depending of the sign of the numerator and denominator in equation 5.5 the quadrant of 2θ changes according to:

Num sign	Den Sign	2θ quadrant shift
+	+	+0
+	-	$+\pi$
-	-	$+\pi$
-	+	$+2\pi$

After determining the right quadrant of 2θ we can calculate the θ angle adding the quadrant shift.

$$\theta = \arctan\left(\frac{2\sigma_{xy}}{\sigma_x^2 - \sigma_y^2}\right)/2 + \text{quadrant shift} \tag{5.6}$$

The error ellipse associated with a certain probability has the following semi axes:

$$\sqrt{\gamma}\sigma_x \tag{5.7}$$

$$\sqrt{\gamma}\sigma_y \tag{5.8}$$

The *error_ellipse* method has been developed to extract the ellipse parameters needed for its representation from the innovation covariance matrix, as shown in algorithm 5.15. This method returns the axes and the angle of the ellipse in order it can be painted over the source image by the *opencv ellipse* function as described in algorithm 5.16.

```
if (error_ellipse_box) {
  track.error_ellipse(&axes, &angle);

  ellipse([track.pXk.(0, 0), track.pXk.(1, 0)], axes, angle);
  track.paint_validation_gate_points();
}
```

Algorithm 5.15: Uncertainty ellipse draw algorithm

```
void pdaf::error_ellipse(CvSize *axes, int *angle) {
  float rho_xy = Sk.(0, 1);
  float rho_x2 = Sk.(0, 0);
  float rho_y2 = Sk.(1, 1);
  float s, t;
  float num = 2 * rho_xy;
  float den = (rho_x2 - rho_y2);
  float dos_theta = atan(num / den);

  *angle = dos_theta * 180 / CV_PI;
  if (num >= 0 && den >= 0)     *angle = *angle + 0;
  if (num >= 0 && den < 0)      *angle = *angle + 180;
  if (num < 0 && den < 0)       *angle = *angle + 180;
  if (num < 0 && den >= 0)      *angle = *angle + 360;
  *angle = *angle / 2;

  s = (rho_x2 + rho_y2) / 2;
  t = sqrt((pow((rho_x2 - rho_y2), 2) / 4) + pow(rho_xy, 2));
  axes->width = (int) _scale * SQRT_C * sqrt(s + t);
  axes->height = (int) _scale * SQRT_C * sqrt(s - t);
}
```

Algorithm 5.16: Uncertainty ellipse axes and angle extracted from S_k

Drawing the validated measurements inside the validation gate

As part of the GUI too, all the centroid blobs that fall inside the validation gate are showed in the image framework, by the method *paint_validation_gate_points*, that runs over the M_k structure that contains the validated points inserted during the validation gate filtering process, as showed in algorithm 5.17.

```
void pdaf::paint_validation_gate_points(Mat img) {
  // shows the validation gate points
  for (int i = 0; i < Mk.elements; i++)
    circle(img, Point(Mk.Z[i].(0, 0), Mk.Z[i].(1, 0)));
}
```

Algorithm 5.17: Painting validation gate measurements

An example of validated points painted over the experiments module (simulator) implemented can be observed in the figure 5.23.

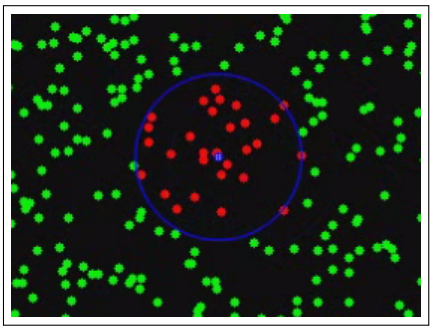


Figure 5.23: Painting the measurements that falls inside the validation gate

5.4.1 Simulator GUI area

The access to the simulator is done through the check box *simulator*, placed at the middle right of the GUI. When this box is selected a new control set is showed. The simulator module was implemented as a help in developing and testing the different filters to be implemented, being a fundamental key in testing the right behavior of the different modules.

The simulator provides testing with the basic Kalman filter and Probabilistic Data Association Filter, using single track, and with the multitrack Join Probabilistic Data Association Filter.

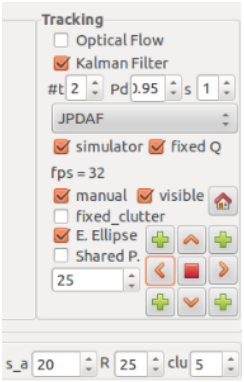


Figure 5.24: Simulator key controls

The control panel, showed in figure 5.24, allows to control all the features implemented by the simulator:

- *Selecting the filter to simulate.* The *kalman_combo box* allows to select the filter operation to be used. The default entry *Standard* is used to simulate the basic Kalman Filter with a single track. The second entry *PDAF* allows to used the Probabilistic Data Association Filter, also with a single track. The third selection *JPDAF* is for use the Join Probabilistic Data Association Filter with multi track implementation.

- *number of tracks* (*#t*) When using the JPDAF filter with the spin button *num_kalmans* (*#t*) the number of tracks to simulate can be selected. Due the other filter implementation do not fit well in multi track environments only one track will be allowed when Standard (KF) or PDAF are selected inside the simulator module.

- *Detection Probability* (*Pd*) Refers as the target detection probability used in the PDAF and JPDAF implementation. If so, the target probability detection is used in the terms of the combined innovation where a association probability is assigned to each measurement according the rightness of each one.

- *Target Selection* (*s*) This spin button, located in the upper right of the simulator control box, is used to select the target movements in the scene, whatever the target moves in random or manual way.

- *House box* This icon is used to put in the middle of the scene the simulated target selected by the target selection spin button.

- *Direction icons* This movements icons can be selected to move the simulated target selected by the target selection spin button in any of the implemented trackers.

- *upper right plus icon* this movement icon moves the simulated target specified by the target selection spin button in a diagonal way to the upper right.
- *upper left plus icon* this movement icon moves the simulated target specified by the target selection spin button in a diagonal way to the upper left.
- *lower right plus icon* this movement icon moves the simulated target specified by the target selection spin button in a diagonal way to the lower right.
- *lower left plus icon* this movement icon moves the simulated target specified by the target selection spin button in a diagonal way to the lower left.
- *stop button* this icon stop the simulated target specified by the target selection spin button.
- *Fixed Q* When this check box is selected the static process noise covariance matrix is calculated and used with all the tracks methods implemented. When the check box is deselected a dynamic process noise covariance matrix is calculated.
- *manual check box* When selected, the simulated target specified by the target selection check box can be moved manually. If it is not selected, the simulated target moves randomly. The action raisen by this button applies to all the targets in the scene.
- *visible* When deselected, an occlusion over the simulated target specified by the target selection spin button happens.
- *Error Ellipse* (*E Ellipse*) This check box allows to observe the error ellipse around each simulated target. The selection of this button applies to all simulated targets in the scene.
- *Process noise sigma* (s_a) This spin button, located in the lower left of the simulator control box, allows to specify the process noise covariance value to build the process noise covariance matrix.
- *Measurement noise sigma* (R) This spin button is used to select the covariance value that will be injected in the measurement noise covariance matrix.
- *Simulation of the clutter* (*clu*) Using this spin button clutter conditions are simulated by specifying the number of random and non persistent measurement that will interact with the simulated targets.

- *Fixed Clutter* when this check box is selected all the simulated points in the clutter remains quite, making a persistent noise.

Chapter 6

Experiments

This chapter contains different experiments with the aim of verify the proper operation of the implemented observation and tracking modules, with a special focus in relevant features developed for each visual tracking algorithm.

It begins with the observation module, showing how the different implemented approaches work. After that it continues with the tracking module, testing the synthetic object generator with the three implemented visual tracking systems, and finalizes showing the use of the visual tracking systems with real images.

6.1 Experiments with observation module

In the next subsections it can be seen how the different object extraction techniques work, identifying the strengths and weakness of each one.

6.1.1 Object extraction using color filter

Choosing to extract the objects to be tracked using an HSV color filter, or any other color filtering technique, causes that multiple false measurements arise, complicating the proper tracking of the targets. If we use, for instance, a basic Kalman Filter approach, because of the complexity to select the right measurement to be assigned to an active track, it makes the filter behavior quite nervous. On the other hand, this lack of accuracy determining where in the image the object to be tracked is can help us to demonstrate how accurate is the PDAF or JPDAF algorithm when tracking in a noisy environment, as we will seen in experiments 6.2.2 and 6.2.3.

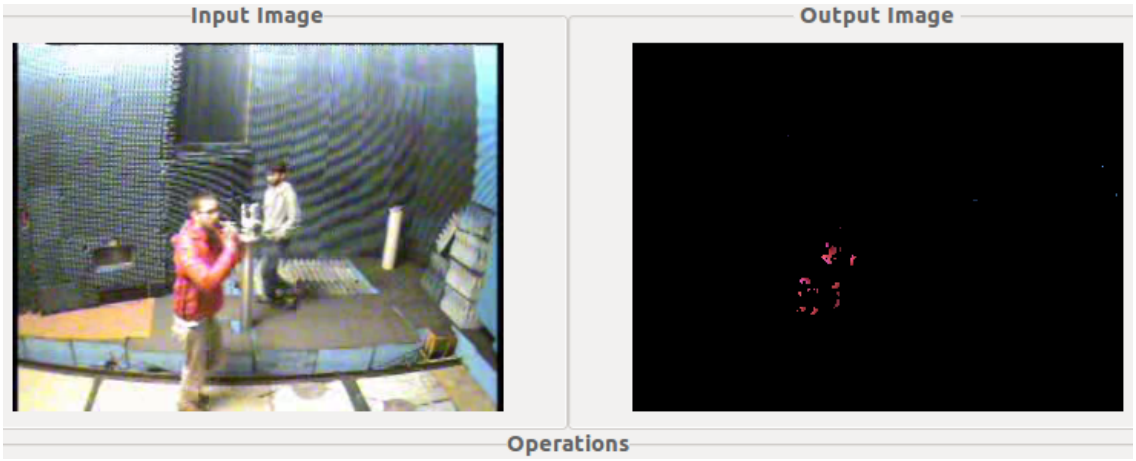


Figure 6.1: Color filtering

In the figure 6.1 we can observe a person wearing a red jacket. The images were filtered using HSV to get the brilliant regions. As we can see, the blobs were identified without the desired accuracy.

One advantage of using color filtered images is that the objects can be identified whether the camera is moving or not, in contrast to extracting the objects using learning background techniques, which assumes that the camera is static.

6.1.2 Object extraction using background subtraction

Object identification using background subtraction requires the use of tunable blob clustering to avoid that multiple blobs were identified as part of the same target. The image 6.2 shows how a lot of blobs that belong to the same object were identified. If this happens the tracking will be complicated because of the multiple computations the system will do to evaluate each measurement probability. After applying morphological transformations over the resulting blobs we obtain grouped blobs, as it can be seen in figure 6.3.

Unlike the object extraction using color filtering, using background subtraction does not allow us to easily maintain the track in case the object remains stable, making possible to miss the track. If there is no movement there is no blob to be detected. In this case we can observe how uncertainty will grows until a new blob (a new movement) is detected, making the filter converge quickly. In the same way, object extraction from background does not fit well when using a mobile camera.

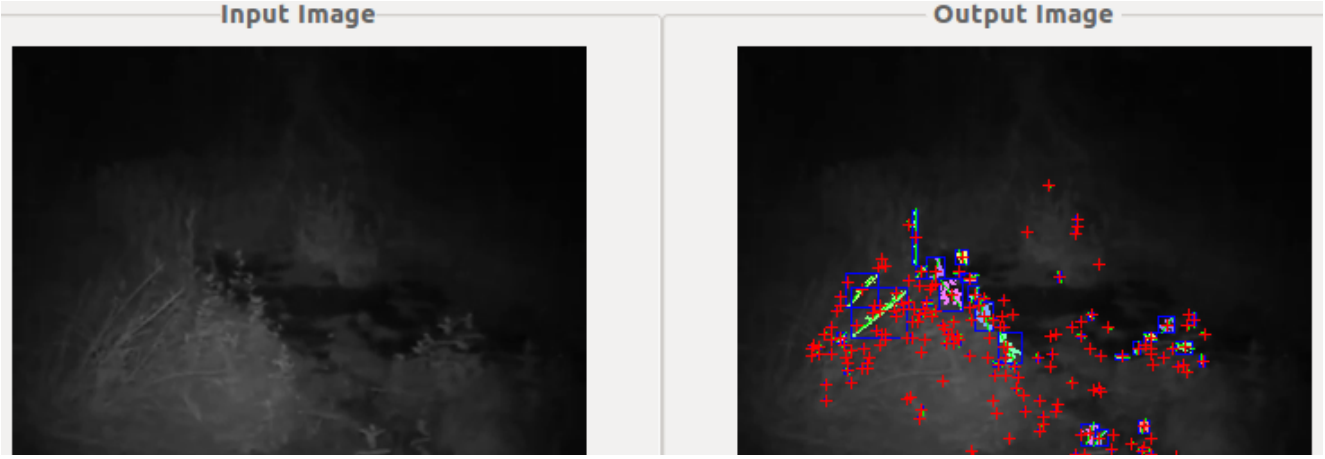


Figure 6.2: Blobs extraction from background without morphological transformation

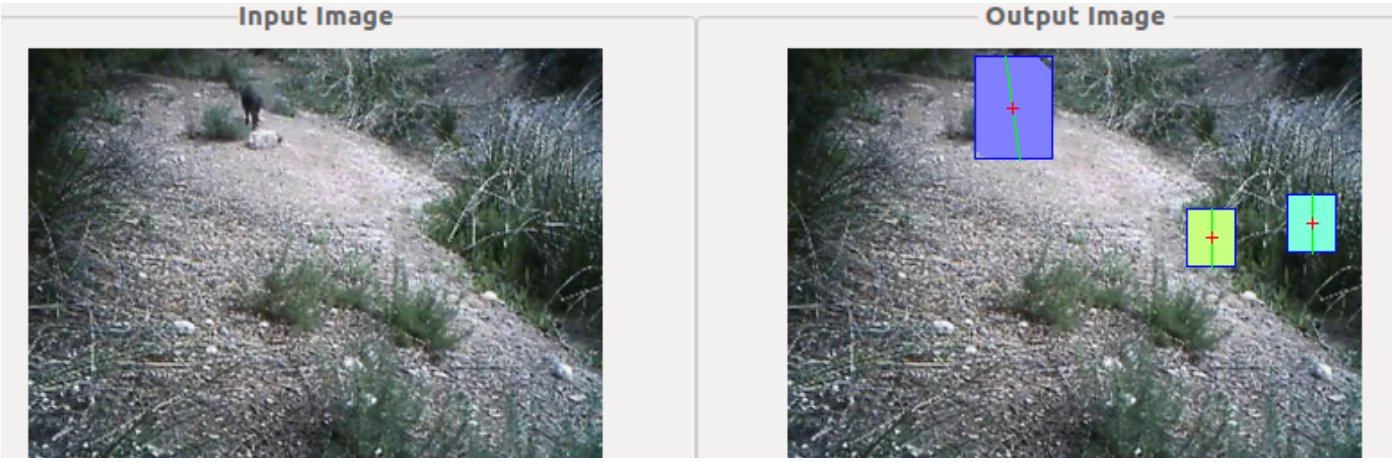


Figure 6.3: Blobs extraction from background after morphological transformation

6.2 Experiments with tracking module

In the next section different experiments with the three visual tracking systems implemented are shown. For each implemented filter it shows experiments with synthetic objects and real images showing a typical case of usage.

The section begins with a test using Kalman Filter with a single synthetic object in order to verify its basic behavior in the absence of noise. An experiment also with Kalman Filter using real images for tracking a single person with occlusions follows. Finally, a multi Kalman Filter experiment will show the tracking of a couple of ping pong players.

After that, the section continues with Probabilistic Data Association Filter experiments, beginning with a demonstration of the robustness of PDAF when it deals with noise using synthetic objects. It continues with multi tracking of ping pong players and finalizes tracking a couple of persons that cross their trajectories where noise and occlusions are present.

The final experiments show how JPDAF solves the problem of tracking several objects with cross paths using synthetic objects with noise. After that, it is the turn of ping pong players to be tracked with JPDAF, and finalizes with a tracking of two persons that cross their paths in a noisy environment with occlusions.

6.2.1 Experiments with Kalman Filter

Different experiments will be showed in this section to demonstrate how the Kalman Filter implementation effectively performs the tracking of both synthetic and real objects.

Testig Kalman Filter with synthetic objects

The synthetic objects module allows us to simulate virtual objects tracking, testing behavior of the tracker in noisy conditions, occlusions, etc. This experiment shows how the uncertainty ellipse grows in absence of new measurements and how it is reduced when visual measurements are available again.

Kalman Filter has been initialized using a synthetic object that moves up from the center of the screen in figures 6.4(a) and 6.4(b). The synthetic object suddenly changes its position moving back to the center of the screen in image 6.4(c). We can observe how in absence of measurements the uncertainly ellipse grows in figures 6.4(d) and 6.4(e) to reduce gradually its shape when the object is found again in images from 6.4(f) to 6.4(h).

In this experiment we have seen how basic Kalman Filter performs well to track objects that can be occluded, where noise is not present.

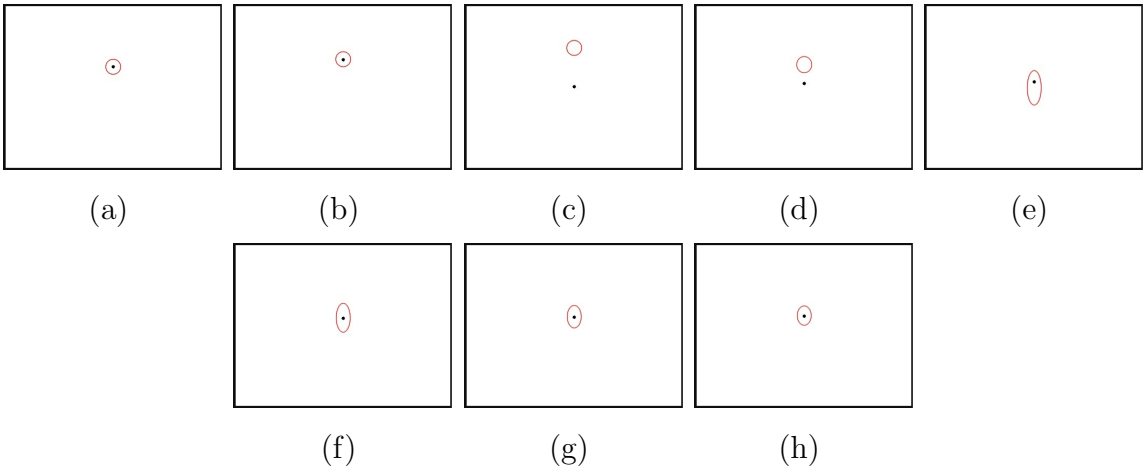


Figure 6.4: Kalman Filter with synthetic objects

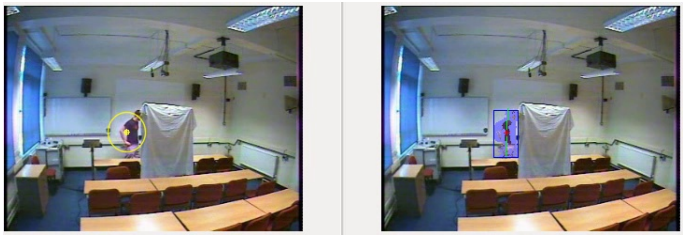
Testing single object Kalman Filter with real images in noisy environment

In the next experiment the basic Kalman Filter has been used to track one person that walks through a room. In each scene the right framework shows the blobs detected and the left one shows the tracking result with a yellow ellipsoid over the interest area.

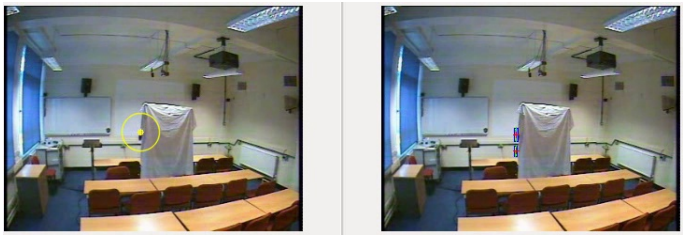
Beginning with the initial stage in figure 6.5(a) the person hides him after a wall in 6.5(b) and because of the object occlusion the uncertainty ellipse grows from image 6.5(b) through image 6.5(c). When the person appears again at the right of the scene in figure 6.5(d) the system continuous to track him. In image 6.5(e) there is an occlusion due the person was not detected and therefore the uncertainty ellipse has grown. In image 6.5(f) the person is detected again and the uncertainty ellipse reduces its shape. This experiment shows how the Kalman Filter performs well to track single object in the presence of noise and occlusions, with the implementation of techniques that facilitates the measurement association as explained in algorithm 5.6.

Testing multi track Kalman Filter in noisy environments

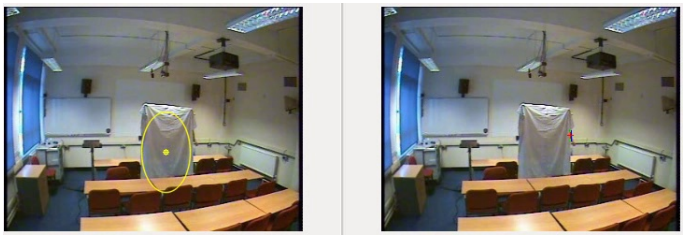
In the next experiment the basic Kalman Filter has been used to track two ping pong players filtered with learning background object extraction technique. Beginning with the initial stage (figures 6.6(a) and 6.6(b)), where both players are tracked, we can observe how



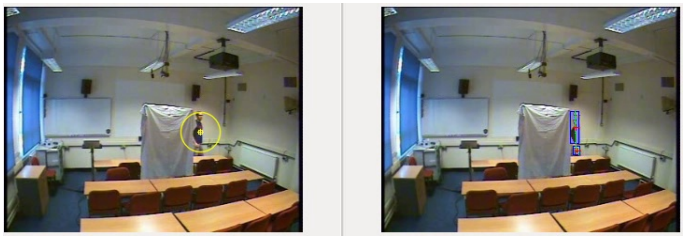
(a)



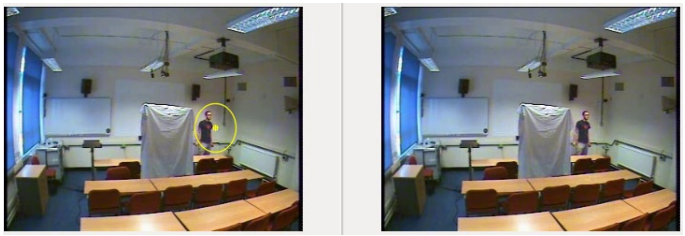
(b)



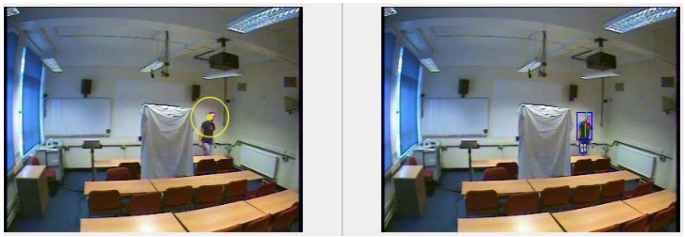
(c)



(d)



(e)

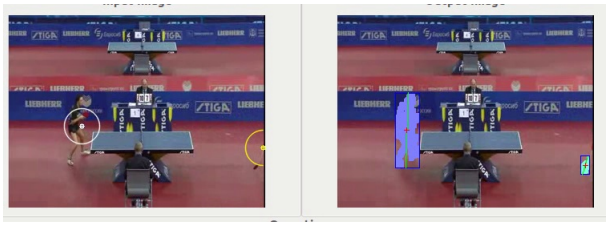


(f)

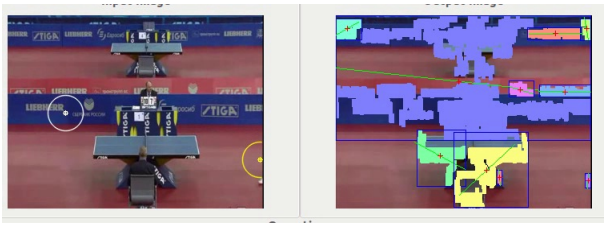
Figure 6.5: Kalman tracking with real images in noisy environments

when both players suddenly disappear from the image (6.6(c)) the Kalman Filter based tracker misses the targets due the filters have been fed with unassigned blob centroids. Due to this, when the players come back to the image (figure 6.7(a)) both tracks miss again the objectives going after any available and unassigned blob centroids (figures 6.7(b), 6.7(c) and 6.7(d))

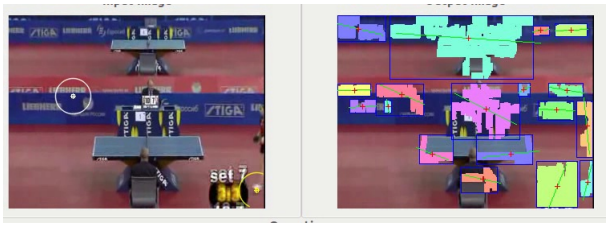
(a) Initial stage



(b) Players tracking



(c) Players occlusion



(d) Players occlusion

Figure 6.6: ping pong players tracking with KF in noisy environment. Part I.

(a) Players comming

(b) Yellow track miss

(c) Tracks missing

(d) Tracks missing

Figure 6.7: ping pong players tracking with KF in noisy environment. Part II.

Kalman Filter makes the tracker quite nervous if noisy conditions and object occlusion happen simultaneously, with independence on the number of tracks. It shows big oscillations between available points assigned to each track in each iteration, unless validation gate is used to find new measurements instead of using Mahalanois distance to assign the closest measurement to each track. To minimize this behavior high values in the measurement noise covariance matrix can be used, making the filter convergence slower. Because of this, Kalman Filter behavior does not perform well when used to track several objects in the presence of noise.

6.2.2 Experiments with PDAF

In this experiment section we will how PDAF can deal with heavy noisy conditions maintaining the tracks. The experiments were developed using synthetic and real observations.

Testing PDAF with synthetic objects

To test the robustness of the visual trackers implementation, the simulator module was used to create heavy noisy conditions. In this experiment each frame generated by the simulator creates 60 random false measurements at 60 frames per second. In such conditions a synthetic object was created at the center of the image that remains stable, meanwhile a new PDAF track was started without a persistent measurement inside its validation gate. As it can be seen through the images 6.8(a) to 6.8(d) the filter diverges because of the noisy conditions (there is no persistent measurement inside the validation gate and then the uncertainty grows), until the image 6.8(e), where the synthetic object remains stable inside the validation gate, making possible the convergence of the filter through figures 6.8(e) to 6.8(j) around the simulated object.

This experiment shows that PDAF approach performs very well to track a single object with heavy noisy conditions.

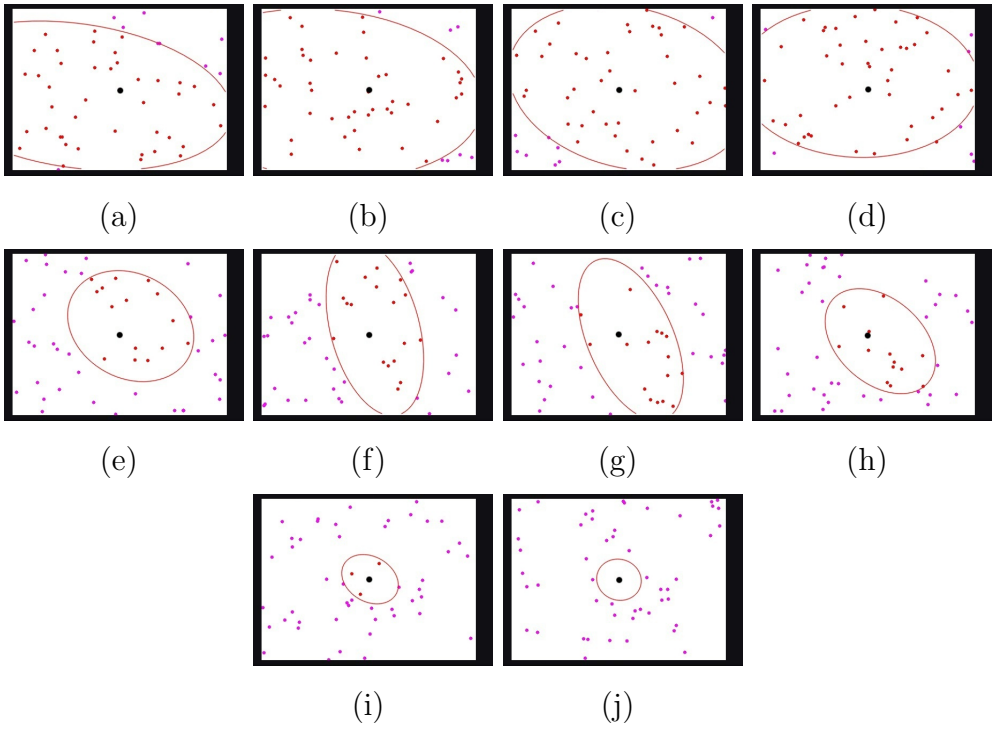


Figure 6.8: PDAF convergence with one measurement and heavy noise

Testing multi track PDAF in noisy environment

The next experiment consists in PDAF tracking the ping pong players that were filtered using the learning background algorithm. In the sequences of figures 6.9 and 6.10 each selected frame is composed of two images. The right one shows all the detected blobs and the left one shows the target tracking using a colored circle. In several figures, it can be seen that inside each colored circle (inside each validation region of each track) points of the same color appear that correspond to the blob centroids that fall inside the validation gate. This experiment can we easily reproduced using the following configuration settings: Blobs detection by background, with $minA = 105$, $maxA = 150$, dilating the blobs with $(5, 10)$ squares. $Pd = 0.95$, $s_a = 25$ and $R = 25$, and fixed Q_k.

Beginning with the initial stage (figure 6.9(a)) where the two players are tracked, we can observe that when the images suddenly change (figures 6.9(b) and 6.9(c)) both players are out of the frame, but the tracking system remains constant around the last position the players had, increasing the error ellipsoid in absence of measurements inside each validation gate. So, when the left player returns to the image, the left track is around it. On the other hand, the right player suddenly appears out of the right track boundaries and so its error ellipsoid begins to grow until new free points were found (figure 6.9(d))

Figure 6.9: Ping Pong players tracking with PDAF in noisy environment. Part I.

When the white tracker is big enough to find new and free measurements inside its validation gate (figure 6.10(a)) it begins to gather new measurements, reduces its error ellipsoid and the tracker quickly converges around the right player again (figures 6.10(b) and 6.10(c)) avoiding the noise introduced by the blobs detection system and the assigned points that belong to the yellow tracker.

This experiment shows that PDAF implementation performs well when tracking several objects, when noisy conditions and occlusions simultaneously arise.

Testing multi track PDAF with cross paths

This experiment shows how PDAF performs with multiple tracking with noisy conditions originated by blob detection technique. We can observe how two persons are walking in figures 6.11(a) and cross their paths in figure 6.11(b). Both tracks share its validated measurements from figure 6.11(b) through 6.11(d) increasing the uncertainty ellipse of each track, to finally differentiate and separate each path in figure 6.11(e). This experiment can we easily reproduced using the following configuration settings: Blobs detection by background, with $Pd = 0.10$, $minA = 300$, $s_a = 10$ and $R = 1$, $dist = 251$ and fixed Q_k.

This experiment demonstrates PDAF implementation is useful when tracking several objects that cross paths, when noisy conditions and occlusions simultaneously arise, whenever stolen mechanism was implemented to avoid assignment a single measurement to multiple tracks.

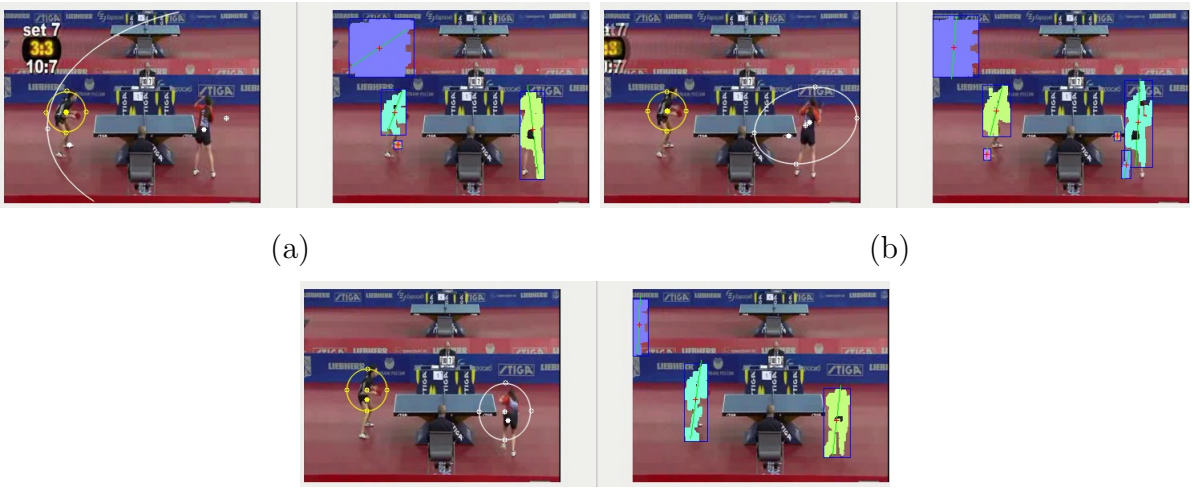


Figure 6.10: Ping Pong players tracking with PDAF in noisy environment. Part II.

6.2.3 Experiments with JPDAF

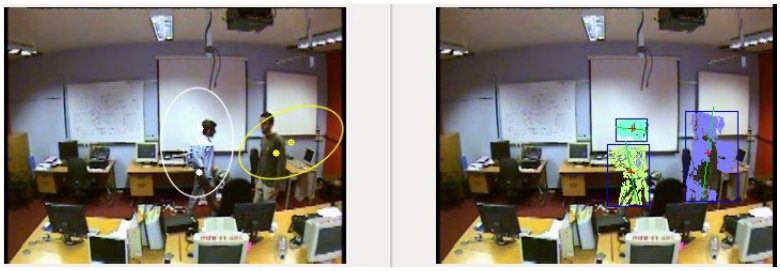
The following experiments consist in use the JPDAF with synthetic observations that cross their paths in perfect conditions (without noise nor occlusions). After that, the same synthetic objects will be used with heavy noisy conditions. The final part consists in tracking red balls and persons with cross paths.

Testing JPDAF with synthetic objects

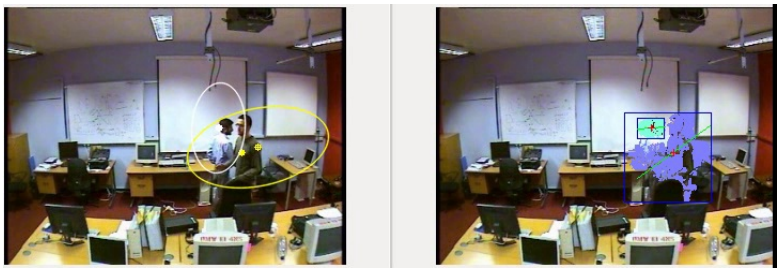
With this experiment we can test all the main characteristics the filters must handle: occlusions, cross paths, clutter conditions, and the convergence or divergence of the filter.

The figure 6.12 shows a cross path testing for JPDAF implementation with two simulated targets in the presence of noise. The ping track (located in the right of image 6.12(a)) is going from right to left, and the black one (located in the left of image 6.12(a)) in opposite sense. When the validation gates find the object that belongs to the other track (from images image 6.12(e) to image 6.12(h)), its uncertainty grows, until the objects are fare away each other making both filters converge without missing the path, from image 6.12(i).

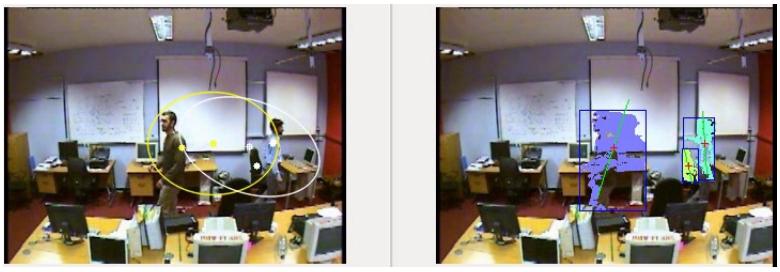
In this experiment we have seen that JPDAF performs well when tracking virtual objects in absence of noise or occlusions, maintaining the path of each track without disruption.



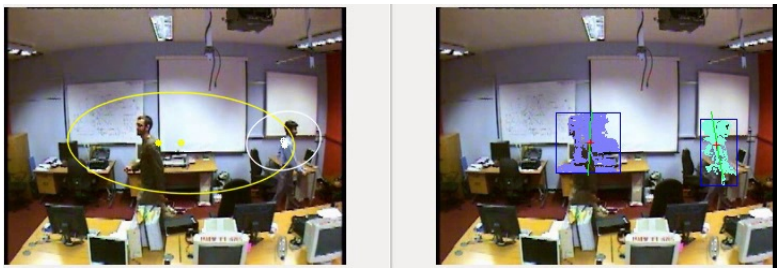
(a)



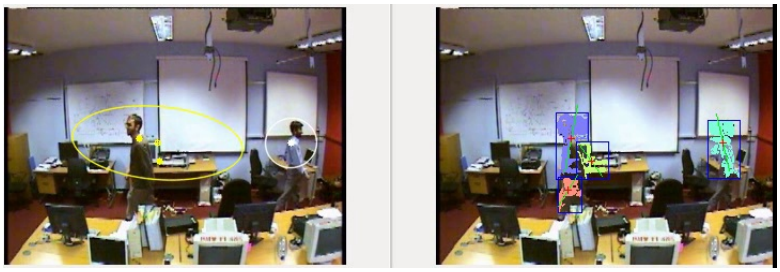
(b)



(c)



(d)



(e)

Figure 6.11: PDAF multiple tracking with real images in noisy environments

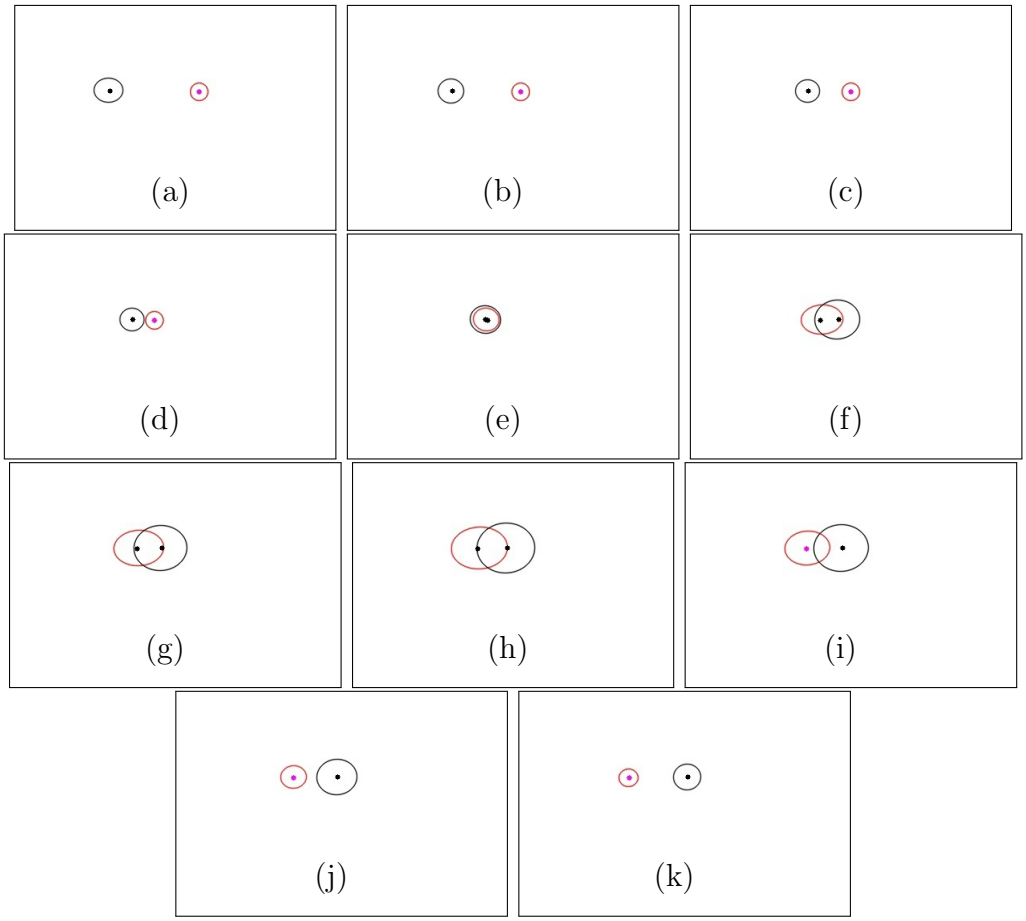


Figure 6.12: Crossing paths with JPDAF

A second experiment, that consists in adding heavy noisy conditions to the previous test, can be shown in figure 6.13, showing that JPDAF is a good approach to track objects that cross their paths even in noisy environments.

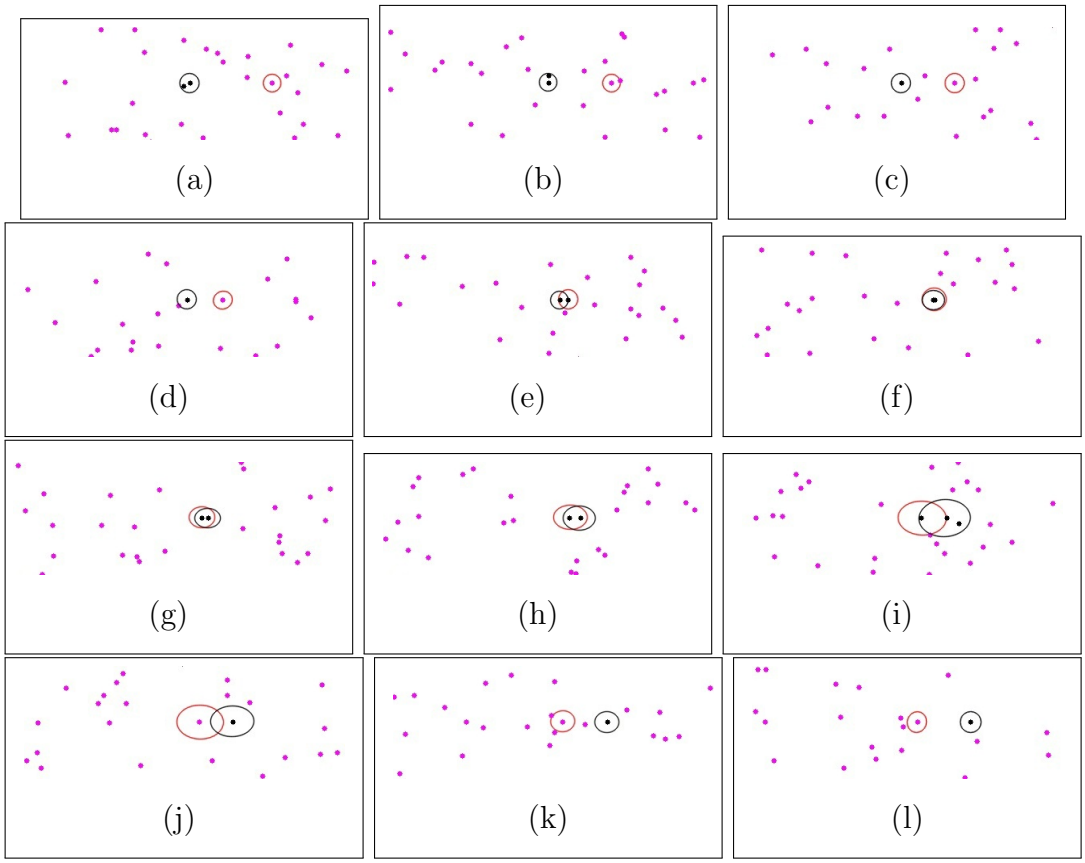


Figure 6.13: Crossing paths with JPDAF in clutter conditions

Testing JPDAF with real images

In the following experiment, two crossing paths objects have been tracked with Join Probabilistic Data Association Filter. In this case the objects to be tracked are two red balls at the end of a stick that cross their paths, and the technique used for identifying the objects was color filtering. The small blue circle around each red ball shows the boundaries of the detected object using HSV filter. This experiment can we easily reproduced using the following configuration settings: Blobs detection with color filter $H(3, 6.28)$, $S(0.65, 1)$ and $V(133, 255)$, $Pd = 0.95$, $s_a = 60$ and $R = 10$ and fixed Q_k.

In the figure 6.14(a) we can see how the right hand ball is under the white track and the left one is under the yellow track. After track initialization, we can see how in figures 6.14(c) and 6.14(d) both objects are considered as one because each ball is close enough to the other one and the filtering technique does not distinguish each one. This situation can be clearly identified because there is only one blue circle instead of two.

(a) Just after crossing (b) After crossing (c) After crossing (d) Path differentiation

(e) Path differentiation (f) Track continuance (g) Track continuance

Figure 6.15: JPDAF with crossing paths experiment. Track continuity

(a) Initial stage (b) Just before croosing (c) Object crossing (d) Object crossing

Figure 6.14: JPDAF with crossing paths experiment. Object crossing

As tracking goes on, while the blobs centroids still continue inside both tracks the error ellipses of each one are growing as shows from the figure 6.15(a) to 6.15(e). When both objects are far away each one, the system continues to track each object individually as it can see in figures 6.15(f) and 6.15(g).

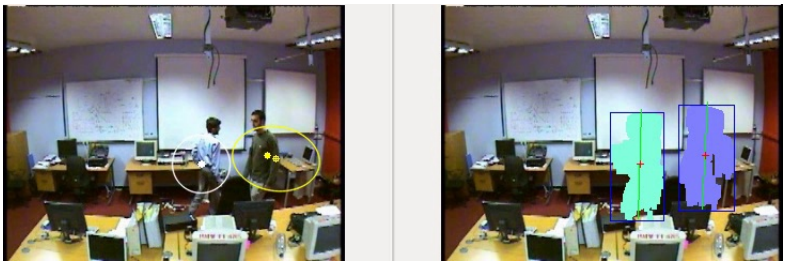
A second experiment has been done with JPDAF using real images in the presence of noise, where two persons are walking in figure 6.16(a) that cross their paths in figure 6.16(b). Both tracks share its validated measurements from figure 6.16(b) through 6.16(c), to differentiate each track from figure 6.11(d) to 6.11(e). This experiment can we easily reproduced using the following configuration settings: Blobs detection by background, using dilate with $Pd = 0.10$, $minA = 800$, $s_a = 10$ and $R = 1$, $dist = 25$ and fixed Q_k.

This experiment shows that JPDAF implementation can be used for tracking several objects with cross paths in presence of noise and occlusions.

6.3 Performance of the implemented visual trackers

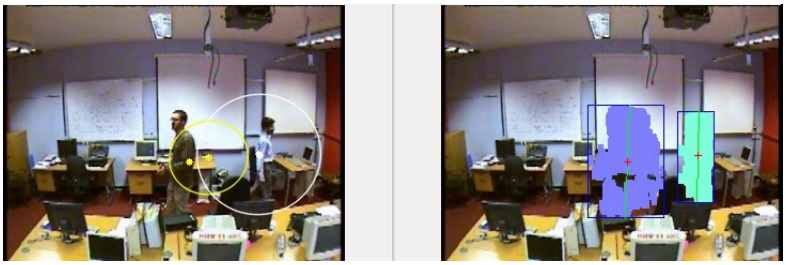
This section describes the computational costs of the three implemented visual trackers. Four videos were used in order to test the execution time and consumed CPU for each algorithm. The suffix "*1 person*" in the figures 6.17(a) and 6.17(b) refers to the video showed in figure 6.5 that has 1077 frames, and consists in a single tracking of a person with occlusion and minimal noise around the region of interest. The suffix "*2 Ping Pong*" refers to the video showed in both figures 6.7 and 6.9 that has 1259 frames and runs two tracks over the ping pong players with heavy noisy conditions. The label ending with "*2 persons*" refers to the video showed in both figures 6.11 and 6.16 that contains 1073 frames with dilated blobs and minimal noise around the region of interest, and the label ending in "*2 persons noise*" refers to the same video with noise around the tracked persons.

In the figures 6.17(a) and 6.17(b) it can be shown the CPU and execution time required by the three visual tracking implementations. According to this metrics, Kalman Filter implementation requires less CPU usage and it is suitable in those scenarios with low noise as showed in the experiment 6.2.1. When several objects are tracked in noisy environments, KF can miss the track as shown in experiment 6.2.1 (see figures 6.7 and 6.9). PDAF has a good behavior with slight increase in CPU, and can be applied to those noisy scenarios where the objects to be tracked are far away each other. JPDAF shows higher CPU usage in scenarios where noise is present, with increase in execution time too when noise appears close to the interest region.



(a)

(b)

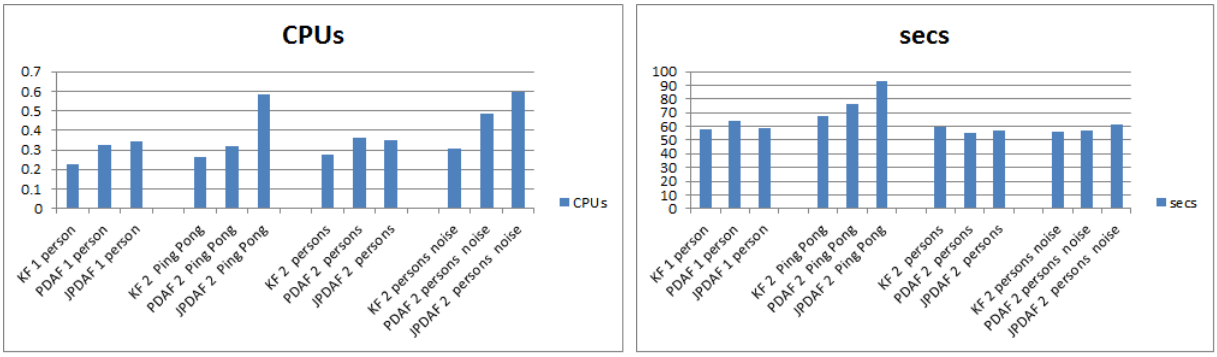


(c)

(d)

(e)

Figure 6.16: JPDAF experiment tracking people with cross paths



(a) CPUs required (2.4Ghz) (b) execution time in seconds

Figure 6.17: Performance of the implemented visual trackers

Chapter 7

Conclusions

This master thesis had the main aim to show, in a practical approach, how three probabilistic visual tracking algorithms work, from the basic Kalman Filter to the more sophisticated Probability Data Association Filter and Join Probability Data Association Filter. The multi track implementation for all of them was considered. This work provides a deep knowledge about the theoretical basis and a first approach implementation in C++ language of them, with more than 5700 lines of code, fulfilling the main goal of this master thesis.

Regarding the first subgoal, in order to track objects from real images a couple of object extraction techniques were developed: selecting the objects filtering by color in HSV space and extracting from the background. The former has the advantages of being very light in computational terms, and works even if the camera is moving. It has the disadvantage of producing numerous false positives or noise in different regions of the image, as explained in section 6.1.1. The latter technique produces blobs were the objects are moving on, giving the possibility to specify thresholds and transformations over the region of interest to better isolate the objects to be tracked. On the other hand, trying to extract an object from background with mobile cameras is not feasible unless additional methods for object identification are considered, because the movement of the camera produces additional blobs. In addition, it can not be used to track objects than remain static for a long time. Object extraction from background experiments were described in section 6.1.2.

In order to help in the development and testing of the implemented probabilistic filters, a synthetic images module was developed. It was able to generate several virtual objects that move with a random or manual pattern, and to generate noise and occlusions. A description of implemented synthetic module can be seen in section 5.2.3

Regarding the second subgoal, there probabilistic filters for visual tracking were developed: Kalman Filter, Probabilistic Data Association Filter and Join Probabilistic Data Association Filter. All of them were adapted to simultaneously track several objects (Kalman Filter implementation was described in 5.3.3, PDAF in 5.3.4 and JPDAF in 5.3.5).

Regarding the third subgoal, all of the implemented visual trackers were tested with noise and occlusion conditions, using virtual objects and video sequences. Once several tracking experiments with the implemented probabilistic filters were done we can conclude that depending on the accuracy of the object detection technique used and the amount of noise present, the basic Kalman Filter should not be considered. To minimize the inaccuracy of KF based trackers in the presence of noise, several techniques could be used that help to better track objects, such as considering only those unassigned points to any other track to be considered as candidates for a new track or defining the minimum distance a new track candidate must be apart of the other tracks, in order to avoid two or more tracks were locked over the same target. Despite of these considerations in implementing a multi track KF, it misses the track when heavy noise arise (new blobs are identified close to the tracked object) and an occlusion of the tracked object happens simultaneously, as we can see in experiments described in section 6.2.1.

The use of Probabilistic Data Association Filter deals very well with noisy environments due the use of validation gate, that avoids that the filter assigns measurements that probably do not belong to the current track because are outside of its uncertainty ellipsoid. The filter can predict very accurately the position of a target even when occlusion happens or in clutter conditions, as it can be seen in experiments described in section 6.2.2. In the implementation of multi tracking in PDAF, and with the objective of avoiding two or more tracks lock over the same targets, each new track only considers those unassigned objects centroids for compute the weights of all measurements inside each validation gate. Using this consideration, the multi track PDAF implemented works fine when noise arise out of the boundaries of the objects, as seen in experiment 6.2.2. In this sense, PDAF with multi tracking capabilities performs very well in low to medium complexity environments, as described in section 5.3.4.

The Join Probabilistic Data Association Filter is able to track multiple objects in the presence of noise with dynamic number of objects to track, with the unique consideration to add the logic for handle the tracks creation or deletion, as it can seen in experiments 6.2.3. It is suitable in those clutter environments where two o more tracks can cross their paths but it requires more programming efforts due the complexity of the algorithm, and therefore its execution could have higher computational costs than other choices.

7.1 Future works

Despite this master thesis is a first approach to probabilistic estimators in single and multiple target tracking (MTT), it can be used as a starting point for future research or to implement automation or surveillance systems based on artificial vision.

The implemented multi track probabilistic filters have been developed using an array of single track filters, where each track is executed in sequence frame by frame. A future development could be to parallelize the track execution, implementing communication mechanisms between all the tracks in order to simplify the logic for handle the number of tracks, and providing a method that allows communication between them. Another area of interest could be to adapt the implementation to use 3D coordinates in order to track objects in the space and use the extended version of the implemented probabilistic filters that takes into consideration non linear systems.

Developing particular applications using the probabilistic filters implemented in this work is also possible. For instance, they can be used to improve the overall performance of the robot soccer league players, giving them a tool for effectively track and predict the position of each rival player in the land, or having a soft tracker of the ball giving them the possibility to implement better collaborative strategies for the team. There is a wide application area for tracking systems. From those to help us to determine if a ball goes beyond the line in several real sports such as soccer, tennis, rugby, etc., to other systems able to count ships using the cameras a drone has installed on it. Another area of interest could be those security systems that will be able to track a person over a campus in order to record her activity. It can also be useful to track pedestrians in traffic areas to avoid they run over by a car, or to control the traffic lights where pedestrians shows the intention to cross over the road.

In such cases, it will be needed a better approach to effectively identify objects before track them. For instance, if the object features are known, a robust feature detector like SURF can be used to better identify the objects in the scene. In other cases it can be useful a mixture of techniques such as object extraction from background and automatic color learning for identifying the objects in the image.

Bibliography

[1] Salman Aslam. Radar tracking. *Georgia Institute of Technology. Center for Signal and Image Processing*, December 2010. University Lecture.

[2] Yaakov Bar-Shalom, Fred Daum, and Jim Huang. The probabilistic data association filter. Estimation in the presence of measurement origin uncertainty. *IEEE Control Systems Managize*, December 2009.

[3] Erik Blasch. Kalman Filter and applied estimation. *Wright State University. College of Engineering Computer Science*. University Lecture 5, 6.

[4] Anton J. Haug. *Bayesian Estimation and Tracking: A Practical Guide*. Chapter 3. General Concepts of Bayesian Estimation. John Wiley and Sons, 2012.

[5] Víctor Manuel Hidalgo. Deteción visual de velocidades de vehículos en la plataforma JDEC. *Universidad Rey Juan Carlos*, 2008. Proyecto Final de Carrera. Ingeniería Informática.

[6] JdeRobot. A software framework for developing applications in robotics, computer vision and home automation, jun 2014. Available at `http://jderobot.org`.

[7] Alberto Martín. Navegación visual en un cuadricóptero para el seguimiento de objetos. *Universidad Rey Juan Carlos*, 2014. Trabajo fin de grado. Grado en Ingeniería de Computadores.

[8] Sara Marugan. Seguimiento visual de personas mediante evolución de primitivas volumétricas. *Universidad Rey Juan Carlos*, 2009. Proyecto Final de Carrera. Ingeniería Informática.

[9] Sara Marugan. Sistema autónomo de detección de caídas con recepción de alarmas en un teléfono móvil. *Universidad Rey Juan Carlos*, 2010. Trabajo Fin de Máster. Máster Sistemas Telemáticos e Informáticos.

[10] Luis Menendez. Master thesis media wiki, jun 2014. Available at `http://www.jderobot.org/index.php/Menendez-tfm`.

[11] Eloy Montero. Análisis de algoritmos para seguimiento visual de objetos. *Universidad Rey Juan Carlos*, 2014. Trabajo Fin de Máster. Máster Sistemas Telemáticos e Informáticos.

[12] María Isabel Ribeiro and Pedro Lima. Introduction to Kalman Filtering. *Instituto Superior Técnico / Instituto de Sistemas e Robótica*, October 2008. University Lecture.

[13] Welch, Greg, and GaryBishop. An introduction to the kalman filter. *TR 95-041, Department of Computer Science. University of North Carolina at Chapel Hill*, 2002.

[14] Wikipedia ZeroC. Internet communications engine, jun 2014. Available at `http://en.wikipedia.org/wiki/Internet_Communications_Engine`.