Master's degree in Automatic Control and Robotics

ON-LINE HUMAN ACTIVITY RECOGNITION BY MONITORING THE INTERACTION WITH THE OBJECTS IN A KNOWN ENVIRONMENT

Author: Enrique Javier Ajenjo Escolano
Director: Joan Aranda Lopez

Escola Tècnica Superior d´Enginyeria Industrial de Barcelona

Dec. 2015

Acknowledgements

First of all, I would like to express my gratitude to my supervisor Joan Aranda Lopez. I would also like to appreciate the efforts of the other team members. Thanks to Manuel Vinagre for sharing his past experiences and knowledges with me. Thanks to both of them, the project is a reality.

Second, I would like to express my gratitude to all my family. Thanks to their unconditional support, I could finish this master program. Thanks to all their efforts, I have become the person I am today. I would also like to express my gratitude to my partner, who stood there and helped me during all this time.

And I finally want to thank the UPC and the Research GRINS group because working in this enriching environment has allowed me to grow as a professional.

Resume

Until now, the robots environment was characterized for being parametrized. This means, that all tasks must be defined and the robot only needs to repeat the programed action continually. Furthermore, the robot and the user don't share the same workspace. But now, new environments have appeared, bringing a new robotic paradigm where the robots and the users share the same workspace in semistructured environments. The tasks in this new paradigm have a plasticity, where they evolve depending on the user. It is therefore important that the system evolves at the same time than the user.

In this project, a solution of an activity recognition system to be used in a robotic assistance on domestic environments is presented. This system works in real-time to monitor the user's activities at the same moment that the action is occurring, computing an early prediction and offering to the system future information about the possible necessities of the user's to finalize the activity that the system predicted.

The solution proposed in this project must solve the following sub-objectives:

1. **Object Segmentation:** The system separates the objects from the scene. We used one model of scene to compare the differences between the model and the actual frame.

2. **Object Recognition:** The system recognizes what objects are involved in the action. To capture the characteristic points of the objects, we first use the SURF Key-Point of the objects and, then, we convert this information in the Bag of Features representation. In the second step of the process, we use a Support Vector Machine (SVM) to build the classifier structure.

3. **Object Tracking:** The system obtains the dynamic information of the objects in the scene. The system uses a CamShift algorithm to monitor the state of the objects.

4. **Activity Recognition:** The system identifies the action that the user is doing taking the previous information into account.

Other important issues we need to take into account in the development of this project are:

- **Scalability:** The system needs to be able to grow so that it can consider more cases. The installation of new hardware or the use of new algorithms are two examples of this.

- **Working in Real Time:** The system needs to predict the actions at the same time than they are happening.

- **Consideration of Different Type of Objects:** In domestic scenarios, different types of objects with different characteristics need to be considered. For example, labeled objects, smooth surfaces with a dominant color and textured objects.

- **The Use of Standard Robot Operation System:** We use the Indigo ROS version to develop this project and share the intermediate information of the project so that other projects can use it.

The results obtained applying all the process is an accuracy detection of the activity only using the static information. The dynamic information we used to increase the detection of the more complex

activities or more complex situations, and the actual version of the project grows in complexity with respect to the previous project but it needs to grow more to take the results of this project into account to increase the trustworthiness of the project.

Contents

List of Figures

Chapter 1

Introduction

Nowadays, nobody doubts that the use of robotic systems in the industry is a great advantage for the companies. The companies increase their productivity making it possible to expand their business to unimagined limits. With this new paradigm and the constant evolution of the technology and the creation of new communication channels, the companies can increase their market. In most of these cases the robots were built to substitute repetitive human tasks. For this reason, usually the construction of the robots uses anthropomorphic structures.

What will be the evolution of these systems? The evolution of this type of robots goes to increasing their specifications to make it possible to produce faster, with more quality to satisfy a global market at the minimum time as possible.

Now, one new horizon begins to develop in the robotics. Many research groups are working to create new applications for this robotic systems to use them in assist people in different daily situations. Examples of these situations can be the following ones:

- Robotics in the rehabilitation therapy.
- Mobility solutions for seniors.
- Incorporation of the robots for the assistance of handicapped or senior people.

What is the main difficulty of using the robotics in this new environment? Basically, the difficulty resides in programing a big number of cases that the system needs to take into account. The main characteristic about the industrial environment is that all the activities in the process are very structured and the users don't share the same workspace at the same time, in contrast to the assistance environment, in which the workspace isn't structured, it is continually changing and the most important issue is that the robot and the user need to share the same workspace. Therefore it is important to ensure the security of the users.

The group of "Intelligent Robotics and Systems" (GRINS) is one of these groups that try to use this new robotic environment to increase the quality of the life of the final users that need to supply physical disabilities or that are in process of rehabilitation after one intervention. The project we present was developed in this group, making it possible to lean on their experiences in this scientific field and to try to implement improvements using previous informations about the topic of this project.

In the following points, we will explain all about this environment, the previous works that exist in the group and the main objectives we will accomplish with this project.

1.1 Objective of this project

The main objective of the project is to create a new layer in the robot ROS that allows to predict the activity that the user is doing during the execution. To do this prediction, the system has to accomplish the following sub-objectives:

1. **Object Recognition and Learning:**
   - **Object Segmentation:** We need to be able to capture the images using different number or types of cameras.
   - **Recognition of Segmented Objects:** The system needs to classify the objects we segmented to know the different objects in the scene by extracting the main object features that each one has.
   - **Create an Object Database:** It is important to add an object database to the system. It makes it easy to manage the information and the possibility to add objects in an efficient way.
   - **Two Different Working Modes:** If we need to make an object learning process, we need to measure if the number of samples of the objects is enough to assure good detections for the system. So, to add a new object in the scene, we need first to execute the object recognition in a supervised learning mode and, then, when we finish to add the new object at the scene, we work in the normal mode.
2. **Activity Recognition and Learning:**
   - **Model of Activity:** We need to parametrize the main features of the activity to use it like a model.
   - **Create an Activity Database:** The system must store the model of the activity to use the previous executions in the following predictions.
   - **Early Recognition Activity:** The system must predict the activity that the user is doing during the execution.

1.2 Available Scenario

The activity recognition layer was implemented in the *project InHands* . This project tries to adapt the use of a robotic kitchen environment to help the final users with handicaps to do the different tasks in a known domestic scenario, in this case in the kitchen, increasing the quality of their life. We can see the actual Inhans scene in the figure 1.1.

Figure 1.1: *InHands Robotic Kitchen Scene*

The InHands environment consists in different specific modules responsible for one specific task. So, all the modules in this environment need to work together to accomplish the assigned task that the robot needs to do. It is important to facilitate the incremental development to make the integration of the different projects to grow the main system in an efficient way possible.

One year ago, the project suffered big hardware and software changes that also produced big changes in the robot environment. For this reason, one of the first activities we needed to do was the actualization of the environment. The most important changes were the following ones:

- **Static Scene:** Actualize the kitchen scenario to add new improvements: mobile shelves, projection surface area.
- **Robot Hardware Maintenance:** Do the main robot maintenance task, change the worn parts of the robot, change the connectors, grease the axis, etc..
- **End Effector:** Change the old End Effector to new Robotiq 3-Finger Gripper. Install the device, integrated in ROS environment and test the good behavior.
- **Robot-Human Interaction Device:** Install and integrate the Leap Motion device in the main system of the robot.
- **Actualize ROS Version:** Change the version of ROS system, from ROS Hydro to ROS Indigo and test that all the packages work correctly.
- **Actualize the Virtual Robotic Scene:** Add the main changes implemented in the improvement. Add the new end effector and the new static scene to the robot model.
- **Adapt the Working Space:** Adapt the environment to make it possible to work in a distributed configuration where all the clients in the same network can subscribe at all the topics that the robot environment publishes and work with them.

During my internship period, I participated in these main tasks with the Inhands team to develop the previous improvements. All these changes produced in the system the improvement of the quality of the system. When I finished my Internship period, the InHands Robot was prepared to work in the standard environment, making it possible to work together with other commercial robots. So, in this moment, we could develop new robotic layers to make it possible to add new characteristics to the system.

The hardware situation after this improvements were applied is the following one:

- **Robot Arm:** The scene has a 3 axis arm. This robots was developed by the group.
- **Robotiq 3-Finger Gripper:** This hand is connected to the robot arm. The main characteristics of the *robotiq hand* are the following ones:
  - **Gripper opening:** 0 to 155 mm.
  - **Gripper weight:** 2.3 Kg.
  - **Grip force (fingertip grip):** 15 to 60 N
  - **Closing speed:** 22 to 110 mm/s
  - **Finger position repeatability:** 0.05 mm
  - **Electrical specifications:** 24V , 1.5A, 4.1W
  - **Communication protocol options:** EtherNet/IP, TCP/IP, DeviceNet, CANopen, EtherCAT, Modbus RTU
  - **Programmable gripping parameters:** Position, speed and force control of each finger

  - **Feedback:** Grip detection, motor encoder position and motor current.

- **Kinect Cameras:** The system has 2 *kinect v2* installed. The main specifications of the cameras are the following ones:

  - **Color stream:** 1920 x 1080
  - **Depth stream:** 512 x 424
  - **Depth range:** 0.4m $\rightarrow$ 4.5m
  - **Infrared stream:** 512 x 424
  - **USB:** 3.0v

- **Server Pc:**

  - **OS:** Linux Ubuntu 14.04 LST.
  - **Processor:** 2 x Intel Core Xeon(R) CPU X5670D @2.9GHz
  - **RAM:** 32 Gb 1333
  - **Hard Drive:** Solid State Drive 250 Gb
  - **Graphic card:** Thesla C2050.

- **Client Pc:**

  - **OS:** Linux Ubuntu 14.04 LST.
  - **Processor:** Intel Core i7 CPU 860 @2.80GHz
  - **RAM:** 8 Gb 1333
  - **Hard Drive:** Solid State Drive 250 Gb
  - **Graphic card:** Nvidia Quadro FX 580/PCIe/SSE2

- **Main Static Scene:** The static scene consists in a big work table and 2 big shelves to try to reproduce the normal kitchen work space.

1.3 Software

The software used in the implementation of the project was conditioned by the previous projects implemented before this one. Taking the existent main structure into account, we decided to make this project using C++.

1.3.1 ROS

The Robot Operating System *(ROS)* is a flexible framework to write robot software. It is a collection of tools, libraries and conventions that aims to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms.

ROS is a large project with many ancestors and contributors. The need for an open-ended collaboration framework was felt by many people in the robotic research community, and many projects have been created towards this goal.

The ROS ecosystem now consists of tens of thousands of users worldwide, working in domains ranging from tabletop hobby projects to large industrial automation systems.

The version we used to implement this project is the ROS Indigo.

1.3.2 OpenCV

(OpenCV) (Open Source Computer Vision) is a library of programming functions mainly aimed to real-time computer vision. It is released under a BSD license. It is free for academic and commercial uses. It has C++, C, Python and Java interfaces and supports Windows, Linux, Mac OS, iOS and Android. OpenCV was designed for computational efficiency and with a strong focus on real-time applications. Written in optimized C/C++, the library can take advantage of multi-core processing. Enabled with OpenCL, it can take advantage of the hardware acceleration of the underlying heterogeneous compute platform. OpenCV has more than 47 thousand people of user community. The version we used to implement the project is OpenCv 2.4.9.

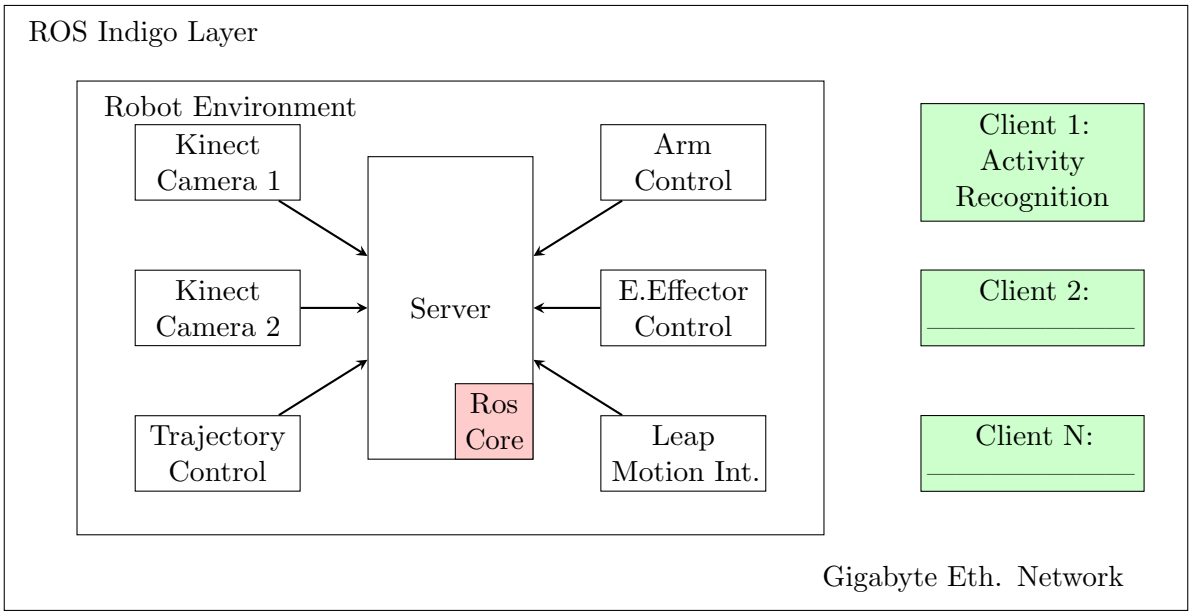
1.4 Hardware Configuration

Using all the possibilities that the ROS framework offers, we can build in an easy way a distributed system where all the information we create in the different layers of the project can be shared with all the devices in the same network. For example, in our case, the activity recognition process we implemented in one of the pc clients.

The server is the main device of the robot environment. In the device runs the core of the Ros Environment. That means that it controls the publications and subscribers of all topics in the system. But it is not only the center of the Ros environment, it also means that the main hardware devices that conform the system are also connected in the server. In the following list, we can see the main devices that the system has:

- **Kinect Cameras:** The server gives us the image of the different cameras in ROS image Topic Cameras.
- **Robot Connection:** The robot is connected to the server using a CANbus protocol.
- **End Effector Connection:** The gripper is connected to the server using Ethernet protocol.
- **Leap Motion Device:** To make the user interaction with the robot possible using the motion of the hands. This system allows to monitor the position of both hands.

The actual situation of the main robot system is represented in the following scheme:



1.5 Initial Design Considerations

To do the initial study of the design we needed to implement, we needed to take the previous projects of the same topic into account to value the main results that they obtained and to evaluate the main problems they had.

If we consider the previous projects, we conclude that most of them have the same main big blocks, because they resolve the activity classification using the same inputs. This main inputs are the following ones:

1. **Object Segmentation:** First we need to separate the objects from the background scene.
2. **Object Classification:** Then, computing the main features of this region, the following part is the recognition of the object using previous information.
3. **Object Tracker:** In some projects, the system adds an object tracker to monitor the main states of the objects.
4. **Skeleton Tracker:** In other specific solutions, the project uses the information of the pose of the user to use it like an activity descriptor.
5. **Activity Recognition:** Using the previous information, the system makes an activity classification and offers one estimation of the user's activity.

In the InHands group a similar project was implemented previously. It had some of these modules explained. The implementation of this project was developed using Python integrating the following steps:

1. **Object Segmentation:** The system only monitored a little object area where the object was stored.
2. **Object Classification:** To classify the objects, this project used color features of the objects.
3. **Activity Recognition:** Using the objects that the system detected and the detection of the variabilities of the store area, it made an user's activity classification and brought an estimated value of the activity to the final user.

Considering the previous information and evaluating the available inputs, we build the first general blocks diagram of our proposal. In a resumed way, our initial proposal has the following main blocks:

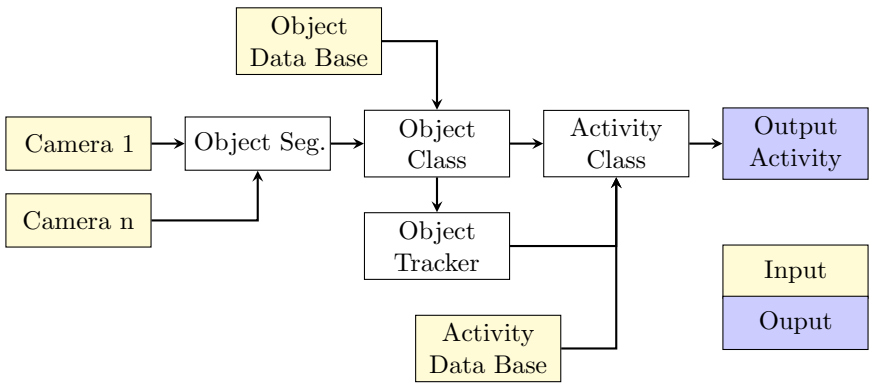


Figure 1.2: *General Initial Design*

Another important issue to consider is the segmentation of the project to make the execution of the code in different clients of the system possible, allowing the system to use the distributed ROS structure to balance the computational load of the system.

Chapter 2

Object Segmentation

2.1 Introduction

In this chapter, we will introduce the first process needed to build a main activities recognition system, using only the objects used in the scene to make a prediction of the activities that the user is performing.

In this project, we try to use all the objects that take part in the action to make a prediction of the activity that the user is doing at the moment. The main objectives are to estimate the future actions in a robotic environment and to try to increase the human & robot user interactions perception.

If our system uses the objects to estimate the activities, it is logical to think that the first big objective of this project is to try to build an object segmentation system to catch all the objects in the environment. For this reason, the main objective of this part of the project is to build a robust system that will be able to catch all the objects in the scene reducing as much as possible the time that the system needs to spend and package all this information for following processes which can use it in an efficient way.

Now, we start by making an introduction of all the techniques to separate the background and to obtain only the objects in the scene using the methods that *OpenCV* has. Then, we will make a little evaluation of each technique and which are the advantages and problems to use them. And, finally, we will comment our proposal.

2.2 Background Subtraction

2.2.1 Simple Image Difference

The difference of image is a good method when using fix cameras observing the scene without changing illumination conditions. In this situation, the background remains unchanged and we can observe all the objects moving around the scene previously known, the background model. In order to extract the objects, we only need to compare the background model with the current frame to detect the changes and, finally, we discrete the results applying a threshold and creating a mask of all the objects in the scene.[1]

For our application, the cameras are fixed on the main structure of the robot and the main objects in the scene (table, shelves, sink) remain in a fixed position, but in our scene there are important changes of illumination for different reasons (the position of the sun, the weather, when the user enters in the scene, etc...). Taking this into account, we tested this method and the results were not acceptable. The movements of the camera when the robot was working produced errors in the output mask, detecting parts of the background and labeling them like objects. Another problem we detected using this method was the variability of the pixel intensity when the conditions changed. This effect forces the system to actualize the background model to adapt the results. The big advantage of using this method is the velocity of computation, but the bad results forced to discard the first method and to find another one more robust.

Trying to improve the first results, we added more complexity adding a dynamic model of the background. In this case, the system computes the variability of each pixel observing the scene in a period of time. Observing the scene in a period of time involves the necessity to store and compute a big number of frames that the system needs to manage. This method can produce problems if we increase the comparison time, needing to use a big quantity of system memory. To solve this problem, we used the "moving average", computing the average value of a temporal signal taking the latest value received into account following this formula:

$$\mu_t = (1 - \alpha)\mu_{t^{-1}} + \alpha p_t \tag{2.1}$$

2.2.2 Mixture of Gaussian Method

The Mixture of Gaussians (MOG) is similar to the previous method explained, increasing the complexity by adding new characteristics to obtain more stable results:

- MOG maintains more than one model per pixel. If a background pixel fluctuates between two values, it can be stored. The algorithm computes two median average values to use in the decision. One new pixel value will be declared as foreground only when the value of this pixel is different from the two model values.
- It doesn't only take the average values into account. The variance of the values is also important. This parameter follows the following relation:

$$\sigma_t^2 = (1 - \alpha)\sigma_{t-1}^2 + \alpha(p_t - \alpha_t)^2 \tag{2.2}$$

- MOG Algorithm computes the probability of the pixel to pertain in the background or the foreground by using a Gaussian model to estimate it and express the results with a probability. Using this parameter, it determines an appropriate threshold.
- Taking the Gaussian model into account, the system changes the pixel belonging into object or background depending on the current values.

In the appendix C.1.1 we explain how we implemented it in OpenCV using C++.

The results we obtained by integrating this method in the main code of the project are better in comparison with the previous performing. Now, by using this algorithm, we can compute the changing information of the frame in a temporal changing model. We have the possibility to change the learning period value as we need. Adapting the value, we manage the time in which an invariant pixel changes from foreground to background.

Once integrated, we tried to evaluate the results by changing the main parameters of the system. But the results weren't right for our application. All static objects were detected on the first moment, but they then disappeared in a few seconds (depending on the learning period value we established). We didn't like that the static object disappeared to facilitate the following parts of the project, but it is a good method to determine what pixels are moving in the scene, reducing the problem and decreasing the computational time.

We can see a few techniques integrated in OpenCV to separate the background of the scene. In order to increase the best results in our system, we decided to implement another method, trying to capture the objects in the scene. This new method we tried to create penalizes the pixel difference from the actual frame with the model. This penalty value increases or decreases depending on the difference. For example, if we have two pixel values in which the difference goes to 0, the penalty value goes also to 0. In the other case, when there is a big difference between both pixel values, the penalty value increases, facilitating the segmentation task.

In our project, we pretend to offer a real time object segmentation, but only of the objects which remain stable, because it has more benefits to implement it:

1. It is easier to build a system which separates the objects from the background of the scene.
2. When the raw image belongs to a static object, the quality descriptor increases. The main characteristics of the object remain without blurring.
3. If we only consider the static pixels, we can eliminate the user's perturbation produced when it is inside the scene.
4. Reducing the pixel information has an important impact in the total computational cost. We can increase the velocity and our system can go faster.
5. The object segmentation part needs to allow to use one or more cameras to make the process in an efficient way.

Observing all the results we obtained, the changing illumination conditions are a big problem we need to try to solve, because it produces important interferences in the building mask process. So we need to change the pixel representation to reduce the dependency of the pixel value with the intensity that the sensor can capture.

Situations in which those effects appear:

- Depending on the hour of the day, the influence illumination changes and produces errors in the detection.
- When the user is inside the scene, it produces a big perturbation in the color pixel values.
- The external conditions modify de pixel value. One example is the weather condition.

We need to find a good relationship of all these problems to make the comparison of both images easier. In the following points, we will present the different relations we studied to solve the problem.

2.2.3 Simple Frame Subtraction

The first technique we used to represent the variability of the frames is the subtraction operator. This method is explained in point 2.2.1.

$$\frac{I_{out}}{I_{in}} = abs(I_{frame} - I_{background}) \tag{2.3}$$

Now, to assure more stable results, we change the pixel representation of the image to minimize the changing effects. This new pixel representation consists on weighting all channels which the total energy that the sensor can detect. Once the pixel representation was made, we apply the module difference of two images. In figure 2.1 we can see how this relationship works for three values of background image going through all possible values of the frame values.

The main advantages that we observe using this method are the following ones:

- It is easy to compute and implement. It doesn't need to spend a lot of time to compute.
- This method has a simple lineal output. This output can facilitate the next steps of the process.
- When the output tends to zero, it means that both pixels are equal.
- It facilitates the creation of the mask process only by applying a threshold.
- A system with a variable threshold taking the number of pixels where differences into account can be built.
- Changing the value of the threshold, the semblance of the actual frame with the image control can be configured.

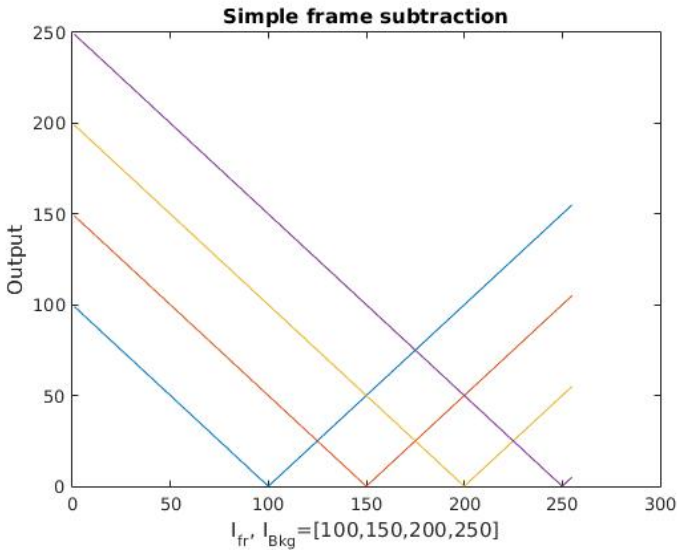


Figure 2.1: *Output Simple Frame Subtraction*

However, when we tried to implement it, the results were not so good as expected. Using the linear relationship, the system had problems to find the best threshold level because, when the threshold level is high, the system can't detect important information about the objects when these parts are similar in the background. On the other case, when the threshold is low, in the output mask appear background parts that difficult the proper use.

2.2.4 Image Differences Evaluation Using Percentages:

In this case, we try to change the main relationship used in the comparison image process. Here we try to express and quantify the difference using percentages. To do this, we only need to divide the Img_{fr} with the Img_{bkg} 2.4. Here, meantime the Energy Image Representation commented previously because of the stable results we obtained.

$$\frac{I_{out}}{I_{in}} = \frac{I_{frame}}{I_{background}} \tag{2.4}$$

In the figure 2.2 we can see different outputs changing the same parameters with the previous expression. Implementing this expression, we can see the following characteristics:

- The output is more enclosed than in the first case. It facilitates the next steps of the algorithm.
- The output keeps being linear. So it probably doesn't reduce one of the main problems that appeared in the previous case.
- The output is not adapted to the same image range, complicating the pixel representation adaptation. To adapt it, we need to apply a scale factor to situate it. Another important issue is the pivoting point situated in value "1". So, finding this scale factor takes on more important to can use it.
- The slope changing depends on the condition. It is not so good if we need to propose a general relationship.

Taking all these considerations into account, we decided to discard this relationship because trying to solve the problems generates more problems to the system. We need to find one expression that resolves the main problems seen in the other cases and that can separate the objects from the background in the easiest way.

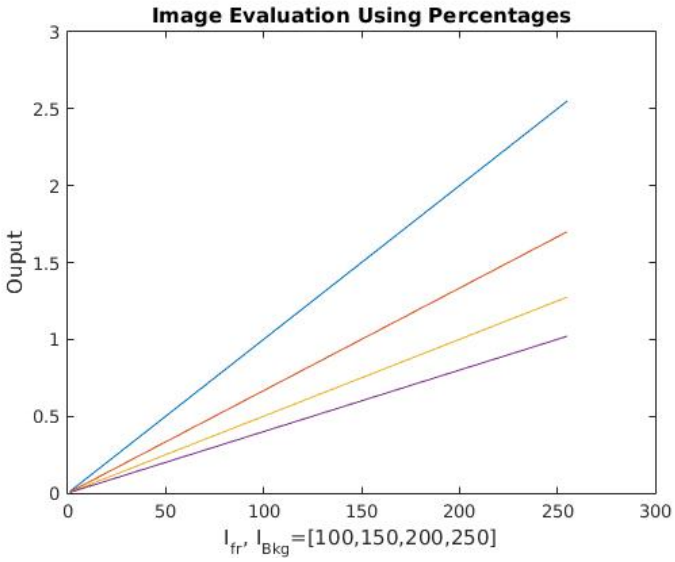


Figure 2.2: *Output of I. Diff. Evaluation Using Percentages*

2.2.5 Image Divisor Relation:

Taking the results obtained in the two previous cases into account, we tried another expression to find the best relationship to do the comparison of both images, minimizing the problems as much as possible and also facilitating the next steps of the system.

This new expression has to be able to:

- Separate the object from the background previously known.
- The results need to be stable despite of the existence of perturbations: Illumination changes, users inside the scene, shadows of the objects, etc..
- The behavior doesn't have to be linear. In this new expression, the output will depend on how similar both images are.
- The output of the process must be a binary image to create the mask to being used in the segmentation part.
- The system must work in real-time

To accomplish all this point, we need to find one expression to facilitate the main characteristics explained before and, also, another important issue we need to accomplish is that this expression must be easy to compute to make it possible that the system works in real-time. We decided to use other knowledge areas with the inspiration to find the solution to our problems. In our case, we tried to adapt the electronic circuit formulation to find the best expression to solve it. We decided to use the easiest formula in electronic, the "resistor devided" 2.5. The transfer function depends on two resistor values to modify the output voltage depending on the relation of both resistors.

When both values (Z_2 and Z_1) are equal or more or less similar, the output of the function is just situated in the half of the values of Z_1 and Z_2. When this relation doesn't appear, the output changes to values under or upper to 0.5, depending on what value changes.

$$\frac{V_{out}}{V_{in}} = \frac{Z_2}{Z_2 + Z_1} \tag{2.5}$$

The "resistor divided" expression has one advantage, the results are not lineal, depending on the value of both parameters. That means that the range of the output changes when more different the

two resistors are. For little changes between resistors, the output remains with the same value. In the other case, when there are important changes, the output changes faster.

Now, adapting this previous considerations to our necessities using the intensity of the energies of the images, the initial formula used for the comparison was the formula 2.6. We expected the same behavior with the original one:

$$\frac{I_{out}}{I_{in}} = \frac{I_{frame}}{I_{frame} + I_{background}} \tag{2.6}$$

Doing the same study than in the other relationships, we can see the results in the figure 2.3. If we see the output, we can see good things that resolve the main problems we had and other problems that appear when try to implement the solution:

- This expression can be used for the comparison process.
- This easy equation produces enclosed values between 0 to 1. The range of value is just the range on the image representation values. Just in the middle of this range, we find the equilibrium point, the point were both pixels are the same.
- The output is not linear. This relationship increases the penalization when the pixel in both images are more different.
- Unification of different slope independent of the values of both images.

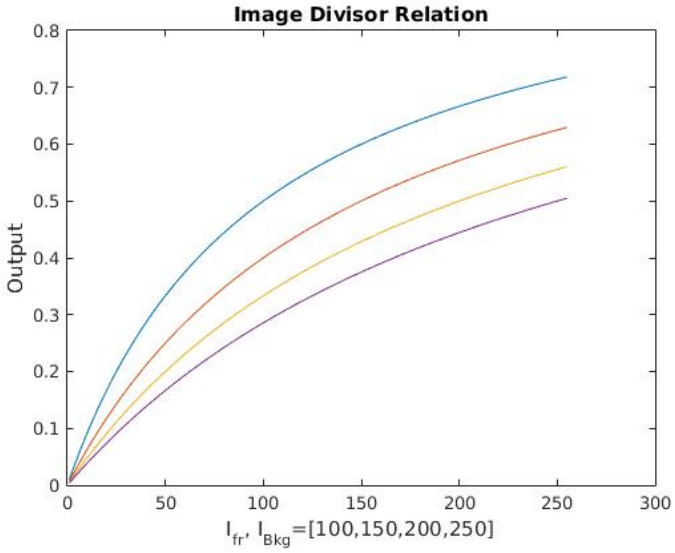


Figure 2.3: *Image Divisor Relation*

Once this equation is implemented, we can see that the good results increase in comparison with other alternatives. We can see that if the pixel values are situated in the central part of range, the behavior is so good. But the problems appear outside this range. For example, in abrupt changes when the energy of the region of the image decreases, or to use it in detection of dark objects.

2.3 Proposal

Once evaluated all the results of the different methods to separate the objects from the background, now we can introduce our implementation proposal. To do this explanation, we can see first the figure 2.4, where the main blocks implemented are explained. Then, in the following points, we will explain in detail the different blocks of this diagram. And, finally, we will observe one example of the segmentation output.

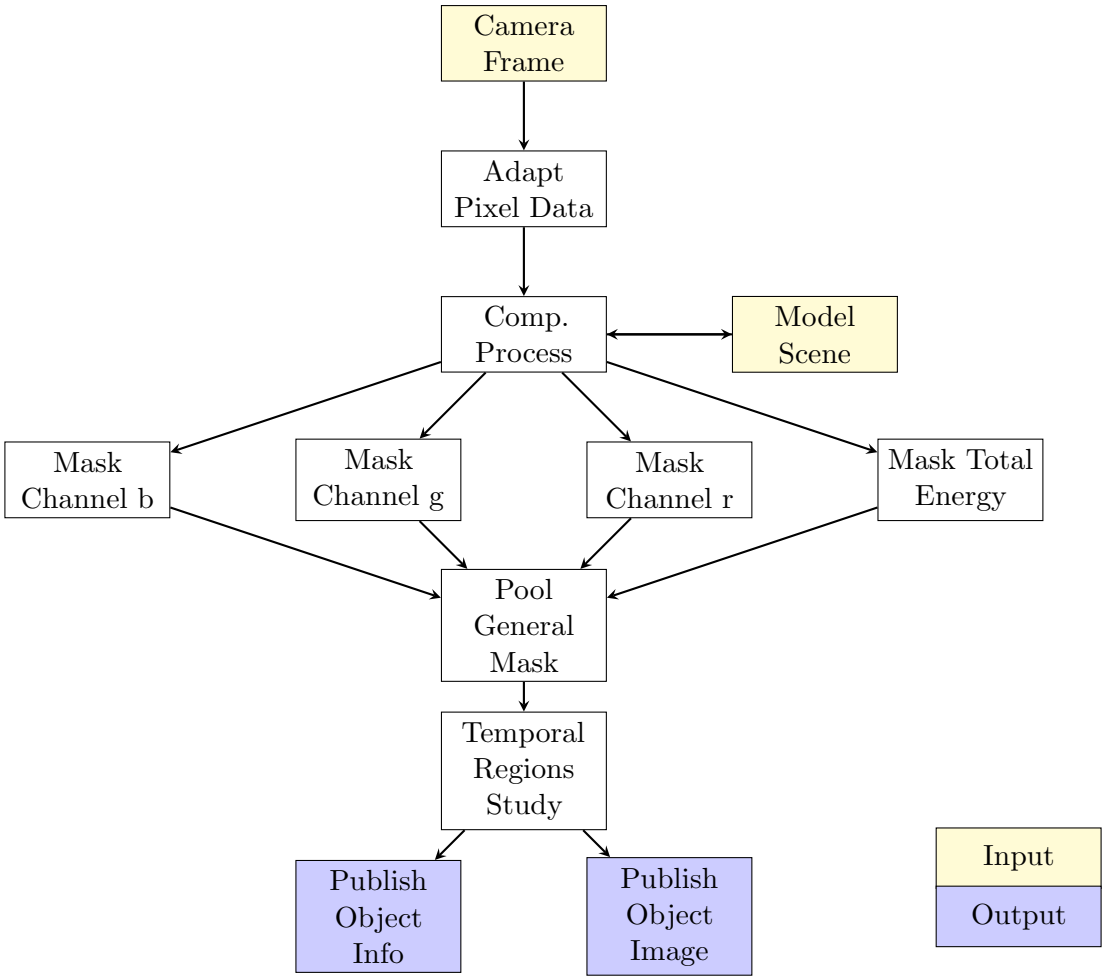


Figure 2.4: *Diagram of the Object Segmentation*

2.3.1 Acquisition of the Raw Image

The first of these steps is to capture the raw image to use in the system. Remember that one of the specifications of the project must be built in ROS framework to facilitate the process of adapting this layer in the core system of the Capdi Robot. To capture the images, we had at our disposal 2 kinnect cameras v.2 to monitor the scene. This hardware allows to capture images in different spectrum and qualities of this image:

- **Color Spectrum:** BGR images in 3 different resolutions (high, medium, low), 8 bits codification.
- **Infrared Spectrum:** The camera has only one channel representation using 16 bits. The resolution of this camera was 640x480
- **Depth Spectrum:** The camera has the same specification that the infrared spectrum.

In the two pieces of codes in appendix C.1.2, we can see the callback function associated to BGR and infrared spectrum respectively. The callback functions in ROS work like interruption, they only enter in the callback function when there is a new frame of the video. The main function of these functions was only to actualize the variables, convert the representation and store in the memory the visual information in the variables of the code.

In the case of infrared and depth channels, the information was represented in 16 bits codification. So, if we would like to see these channels like a normal image, we would need to apply the scale factor

in 8 bits representation. This consideration increases the possibilities of using it taking other channels of the camera into account, for example, the Color Channel C.1.3.

2.3.2 Adapting the Pixel Representation

One of the big problems we tried to solve is the necessity of reducing the perturbation dependency to increase the good results. To increase the efficiency of the process, the first step we did was to apply one Gaussian filter in the raw image to reduce the edges of the image. Testing all the methods explained in the previous points, we saw that when the comparison process is done, just in the edges of the objects and scene appeared differences between both images. Blurring the image, this bad effects are reduced making the system more efficient and the results are more stable reducing the quantification noise error of the sensor. Then, we can change the pixel representation values weighted 2.8 the value using the total energy 2.7 that the sensor can detect.

$$I_{Energy} = ch_b + ch_g + ch_r \tag{2.7}$$

$$I_{Energy_{ch}} = \frac{Image_{ch}}{I_{Energy}} \tag{2.8}$$

2.3.3 Comparative Process

When the adaptation of the pixel representation is finished, the next step of the progress is to define the relationship to make the comparison process. Observing the results that we obtained, we can see that using the "Image Divisor Resistor" we obtained the best result of the other process tested. But this method had an important limitation, explained before. To reinforce the result, we used a "simple frame subtraction" to solve this limitation. Both methods complemented the information, one for the middle of the ranges and the other used in extreme situations.

In the implementation of this part of the code, a few changes were added in the original formula to facilitate the implementation process. We centered the output variation around zero and, then, computed the module. To finalize the process, we added a fixed gain to expand the results in all the possible range of the image representation zero to one. In the figure 2.5, we can see how the output of the pixel comparator process changes using the variation mentioned before. In the appendix C.1.4 we can see how it was implemented.

The system needs to repeat the process for each channel used in the process. In our case, we only used the color BGR channel. Using this implementation, will allows in future versions to add new channels to increase the efficiency of the system. For example, we can easily add the infrared and depth channels to use all the possibilities that the hardware allows. For example, if we talk about the depth channel, we see that it has a nice advantage of using it, because it can know the distance of the objects represented like a normal image, so we can add another channel of the BGR image (BGR-D) and make the same process, and the same with the infrared channel (BGR-D-I).

Using the depth channel has advantages because we can know important differences between the model background and the actual situation, but it is not perfect. We can detect two important problems of using this channel. First, in detecting little object like tools (spoon, knife) or other accessories (heat protected) that its measures are near to fix objects (table, shelf, etc). The resolution and the quantification error produce problems to detect these objects. The second problem has to do with an important hardware limitation. The kinnect camera version 2 uses the time of light technology to measure the distance of the objects. In definitive, the system uses the reflexion of the light in the body of the object and measures the waste time. If we try to know the distance of one dark object, the quantity of this light returned to sensor decreases. In extreme situations when the object is black, the light can't return to the object because the body of the object absorbs all the light, producing a false distance detection.

Using the infrared channel, the quality of the detections can also increase, but we don't study the effects and possible disadvantages of using it.

Taking the benefits and the disadvantages of using different channels into account, we can build a mathematical expression to involve all the channels in the same equation to increase the results of the system.

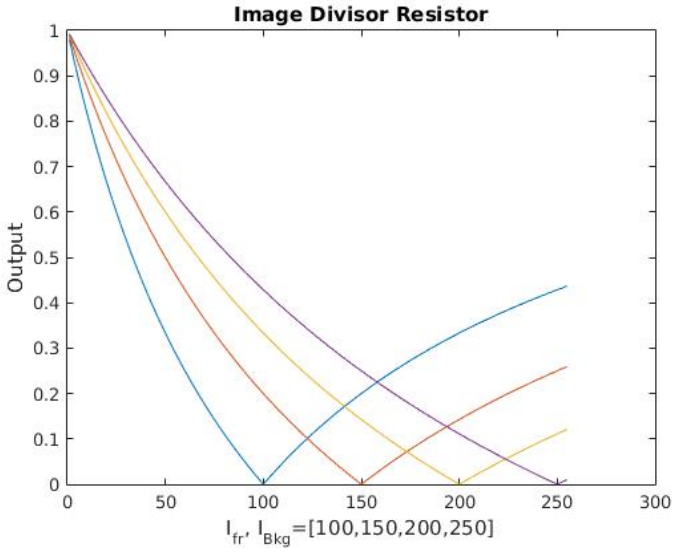


Figure 2.5: *Adaptation of the "Image Divisor Resistor" formula*

To finish the comparative process, we only need to apply a threshold level to create the partial mask. We have one partial mask for each channel we use to detect the differences. To compute the main mask, the system only needs to add all the contribution of the partial mask to build it 2.9.

$$Mask_{Main} = Mask_{ch_b} + Mask_{ch_g} + Mask_{ch_r} \tag{2.9}$$

2.3.4 Study of all Mask Regions

Before starting to describe the process of computing the final region when the objects are situated in the scene, it is a good moment to remember one of the main objectives of this process. This main objective is that the system only computes the characteristics of the **static objects**, discarding all the objects moving in the scene. Fixing this objective, we can increment the quality of the key points of the object and the efficiency of the system.

Later, we need to take the apparition of bad regions that the system detects like objects but they really aren't into account. This situation appears for several reasons. For example, the quantification error of the cameras, noise and perturbations. To eliminate this perturbation is not easy, but we can reduce as much as possible the contribution of this reason by monitoring the position of all region. The best way to monitor the contribution is by implementing a temporal evaluation storing the mask that the system can create in a determinate period of time. The system will eliminate the areas of the image that don't stay stable. In the appendix C.1.5 we can see how this filtering process was implemented.

2.4 Example Output

In the following picture we can see an example of the output of the object segmentation process.

Figure 2.6: *Output Image Segmentation Example*

Figure 2.7: *Mask Example*

In the appendix D we can see more examples of the output of the process.

Chapter 3

Object Recognition

Another important block of this project is the computation of the main features that define the objects inside the scene. In the previous points, we said that the best way to capture the object is in a static mode because the system can find the key points of the object easily. Trying to find the best solution of our project, we studied different solutions to find the best identification key point of the objects. In this chapter, we will talk about the main techniques and how they work. Finally, we will also talk about the final implementation and comment the results we obtained.

First of all, we defined 3 fundamental properties that an object recognition system must accomplish:

- **Velocity:** We need to compute this task in real-time, so the velocity is important.
- **Precision:** To make a good detection, we need to assure that the features of the object are robust and that the key-point defines the characteristics of the object.
- **Repeatability:** The repeatability is an important parameter because if it doesn't exist, we can't make the classification process and, then, we can't recognize the objects.

3.1 Methods to Extract Features

3.1.1 Color Histograms

One of the most usual object descriptor we have is the use of the color information of the objects like a descriptor, using the histogram values of the 3 image channels. We can use this information like a descriptor to use it in recognition and classification objects inside the scene. One of the interesting things about this method is that the system can preserve the raw information of the objects. [3]

Using the color information to try to identify objects will be a good strategy because the manufacture companies use really visual product designs to differentiate their products from the competitors ones. The companies try to make their products using a combination of colors and specific forms to define the objects. If the marketing company makes a good job, the final user can identify the trademark even when the label is damaged. In figure 3.1, we see one example of this affirmation: though the label is removed, the users can recognize the object using only the colors information.

So, taking all these considerations into account, using a low computational level, can be a priori a good identifier of the products. The output of the system is a global descriptor of the object, represented only by one vector with "X" bin for each channel of a RGB image, making the integration in comparator structures to build the recognition machine easier.

Figure 3.1: *Products without the layer can remain a trademark using the color*

3.1.2 Scale-Invariant Feature Transform (SIFT)

Another possibility to compute the main features of an object is SIFT [4] . Using this algorithm allows to find specific local characteristics of the objects creating specific descriptors using this point. It is a good way to find these specific points that define the object: trademark forms, abrupt changes, etc.

The main steps that define the algorithm process are the following ones:

1. **Scale-space Extrema Detection:** To detect the key-points in different scales, we need to compute a Lapalcian of Gaussian with a different σ configuration. For example, if we use a low σ, it gives high values for a small corner and, in the other case, a high σ fits well for a larger corner. We can find the local maximum across the scale and space, which gives us a list of (x,t,σ) values and means that there is a potential key-point at (x,y) at σ scale. Computing the LOG is a little costly. SIFT algorithm uses different Gaussians because it is an approximation of LOG with less computational costs. The main values that the paper [4] proposes are the following ones:

$$N_{octaves} = 4;\ N_{scalelevels} = 5;\ \sigma_{initial} = 1.6;\ k = \sqrt{2}. \tag{3.1}$$

2. **Keypoint Localization:** When potential key-point locations are found, they have to get more accurate results. Using a Taylor series expression of scale space is a way to get a more accurate location of extrema, and if the intensity at this extrema is less than a threshold value, it is rejected.

3. **Orientation Assignment:** Now, an orientation is assigned to each keypoint to achieve invariance to image rotation. A neigbourhood is taken around the keypoint location depending on the scale, and the gradient magnitude and direction are calculated in that region.

   An orientation histogram with 36 bins covering 360 degrees is created. It is weighted by a gradient magnitude and gaussian-weighted circular window with σ equal to 1.5 times the scale of keypoint. The highest peak in the histogram is taken, and any peak above 80% of it is also considered to calculate the orientation. It creates keypoints with the same location and scale, but with different directions. It contributes to the stability of matching.

4. **Keypoint Descriptor:** A keypoint descriptor is created. A 16x16 neighbourhood around the keypoint is taken. It is devided into 16 sub-blocks of 4x4 size. For each sub-block, a 8 bin orientation histogram is created. So, a total of 128 bin values are available. It is represented as a vector to form a keypoint descriptor. In addition to this, several measures are taken to achieve robustness against illumination changes and rotation.

5. **Keypoint Matching:** The keypoints between two images are matched by identifying their nearest neighbours.

3.1.3 Speeded-Up Robust Features (SURF)

Another descriptor we studied to introduce in the project is the SURF descriptor [5]. This descriptor is based on a SIRF algorithm, so both have similar properties.

The first step consists on fixing a reproducible orientation based on information from a circular region around the interest point. Then, we constructed a square region aligned to the selected orientation, and extracted the SURF descriptor from it.

Now, we will explain the main parts to compute this descriptor in a detailed way:

1. **Orientation Assignment:** In order to be invariant to the rotation, we need to take a reproducible orientation for the key-points. So, the algorithm computes the Haar-wavelet responses in x and y directions in a circular neighborhood with radius 6s, and s is the scale at which the interest points were detected.

   Once the wavelet responses are calculated, they are represented as vectors. The dominant orientation is estimated by calculating the sum of all responses. The longest of such vectors lends its orientation to the interest point.

2. **Descriptor Components:** To extract the descriptor, we first need to construct a square region centered around the key-point of the object and oriented selected in the previous point. Then, the region is split up regularly into smaller 4 x 4 square sub-regions. For each sub-region, we need to compute a few simple features at 5 x 5 space sample points: Haar-wavelet in horizontal direction d_x, Haar-wavelet in vertical direction d_y. Both parameters are weighted with a Gaussian ($\sigma = 3.3s$) centered in the interest point. The next step is to sum the wavelet responses d_x and d_y over each subregion and construct a vector with the results. We also need to extract the sum of the absolute values of the responses to know the polarity of the intensity changes ($|d_x|$,$|d_y|$).

   So each sub-region has a four-dimensional descriptor vector v for its underlying intensity structure:

$$v = \left(\sum d_x, \sum d_y, \sum |d_x|, \sum |d_y|\right) \tag{3.2}$$

   The result of this process is a descriptor invariant to a bias in illumination (offset), invariance to contrast (a scale factor) is achieved by turning the descriptor into a unit of 64 bins vector.

It is possible to modify the main structure of the descriptor to increase the accuracy (using SURF with 128 bines, 5 x 5 subregion structure) or to reduce the accuracy by increasing the velocity to compute (SURF with 36 bines, 3 x 3 subregion structure) depending on our necessities.

The main benefits of using SURF in comparative with SIRF are the following ones:

- Using a SURF algorithm increases the speed to compute the descriptor. The SURF descriptor only needs around 354 -391 ms in comparison with the 1036 ms from SIRF.

- It increases the recognition rate. The SURF algorithm had an 85% of accuracy in comparison with the 78 % of SIRF.

Figure 3.2: *Example of output using SURF algorithm*

3.1.4 KAZE Features

Searching different ways of building a descriptor to the object, we found the KAZE algorithm [6]. In this case, the system builds in a nonlinear scale space up to a maximum evolution time using AOS techniques. The output of the system is similar to the techniques we saw previously.

To build the descriptor, we need to make the following steps:

1. **Change the Representation Image:** The image is represented in a non linear space using the following formula:

$$L^{i+1} = \left(I - (t_{i+1} - t_i) * \sum_{i=1}^{m} A_l(L^i) \right)^{-1} * L^i \tag{3.3}$$

2. **Feature Detection:** To detect points of interest, the algorithm computes the response of scale-normalized determinant of the Hessian at multiple scale levels. For the multi-scale features detection, the set of differential operators needs to be normalized in relation to the scale.

$$L_{Hessian} = \sigma^2 \left(L_{xx} L_{yy} - L_{xy}^2 \right) \tag{3.4}$$

where $(L_{xx} L_{yy})$ are the second order horizontal and vertical derivates and L_{xy} is the second order cross derivate.

3. **Features Description:**

   Finding the Dominant Orientation: To obtain a rotation invariant descriptor, it is necessary to estimate the dominant orientation in a local neighborhood centered in the key-point location, similar to SURF [5].

   Building the Descritpor: The algorithm uses the M-SURF descriptor adapted to nonlinear scale space. For detected features at scale σ_i, L_x, L_y of size σ_i, they are computed over a $24\sigma_i$ x $24\sigma_i$. This grid is divided into 4 x 4 subregions of size $9\sigma_i$ x $9\sigma_i$ with an overlap of $2\sigma_i$. Then, each subregion is weighted using a Gaussian ($\sigma_2 = 1.5 * \sigma_i$). The descriptor vector for each subregion is the following one:

$$d_v = \left(\sum L_x, \sum L_y, \sum |L_x|, \sum |L_y| \right) \tag{3.5}$$

The result is a normalized vector of 64 bins for each key-point found.

Using this algorithm, the repeatability parameter increases. KAZE increases the score a 40% in comparison with SIFT algorithm and a 20% in comparison with SURF. But it has a penalization in the computational time because it spends more time to compute (KAZE needs to spend approximately 2.4 more time than SURF).

3.1.5 Acelerate KAZE Features

This algorithm is a variant of the previous one, trying to improve the results obtained by KAZE. In this case, it uses a FED scheme (Fast Explicit Diffusion) to build a nonlinear scale space considering the anisotropic diffusion. To improve the speed of the construction, the algorithm embedded the FED scheme into a fine to coarse pyramidal framework to create a robust features detection and description.[7]
The main steps to build this features are the following ones:

1. **Build a Nonlinear Scale Space with Fast Explicit Diffusion:**

   (a) The algorithm first needs to define a set of evolution times from which we can build the nonlinear scale space. The scale space is discretized in a series of O octaves and S sub-levels. The octave and the sub-level indexes are mapped to their corresponding scale σ (pixel) using the formula:

   $$\sigma_i(o,s) = 2^{o+s/S}, o \in [0...O], s \in [0...S-1], i \in [0...M] \tag{3.6}$$

   where M is the total number of filtered images.

   (b) Convert the set of discrete scale levels in pixel units σ_i using:

   $$t_i(o,s) = \frac{1}{2} * \sigma_i^2, i = \{0...M\} \tag{3.7}$$

   (c) The imput image can be convolved with a Gaussian of standard deviation σ_0 to reduce noise and possible artefacts.

   (d) From the smoothed input image, the algorithm computes the contrast factor λ as a 70 % percentile of the gradient histogram.

   (e) Given the input image and the contrast factor, the FED scheme can work. Using $M-1$ outer FED cycles and for each cycle computes the minimun number of inner steps n. For example, in a 2D image the maximal step size that doesn't violate stability conditions is $\tau_{max} = 0.25$

   (f) Once the last sublevel in each octave is reached, the algorithm downsamples the image by a factor of 2 using the smoothing mask $\left(\frac{1}{4}, \frac{1}{2}, \frac{1}{4}\right)$ and uses that downsampled image as the starting image for the next FED cycle in the next octave.

   (g) After down-sampling the image, it needs to modify the contrast parameter λ. The smoothing mask reduces the contrast of an ideal step edge by 25% and, then, the contrast parameter needs to be multiplied by 0.75.

The next pseudo code explains the previous steps in a resumed way:

Algorithm 1 Pyramidal FED approach for non linear diffusion filtering

Input: Image L^0,contrast parameter λ, τ_{max} and set of evolution times t_i
Output: Set of filtered images $L^i, i = 0..M$

for $i = 0 \rightarrow (M-1)$ **do**
  Compute diffusivity matrix $A(L^i)$
  Set FED outer cycle time $T = t_{i+1} + t_i$
  Compute number of FED inner steps n
  Compute steps sizes τ_j
  Set Prior $L^{i+1,0}$
  L^{i+1}=FEDCycle($L^{i+1,0}, A(L^i), \tau_j$)
  if $\sigma_{i+1} > \sigma_i$ **then**
    Downsample L^{i+1} with mask $\left(\frac{1}{4}, \frac{1}{2}, \frac{1}{4}\right)$
    $\lambda = \lambda * 0.75$
  end if
end for

Algorithm 2 FED Cycle

function FEDCycle($L^{i+1,0}, A(L^i), \tau_j$)
  for $j = 0 \rightarrow (n-1)$ **do**
    $L^{i+1,j+1} = (1 + \tau_j A(L^i))L^{i+1,j}$
  end for
  Return $L^{i+1,n}$
end function

2. **Feature Detection:**

   When the previous process finishes, it is the moment to detect the main features that define the image in the new non linear space representation. To make this process, we follow these steps:

   (a) Compute the determinant of the Hessian for each one of the filtered images L^i in a non linear scale space. The set of operators is normalized taking the scale into account with:

$$\sigma_{i,norm} = \frac{\sigma_i}{2^{o^i}} \tag{3.8}$$

$$L^i_{Hessian} = \sigma^2_{i,norm}(L^i_{xx}L^i_{yy} - L^i_{xy}L^i_{xy}) \tag{3.9}$$

   (b) Compute the second order derivate, the algorithm uses concatenated Scharr filters with step size $\sigma_{i,norm}$

   (c) At each evolution level i, the system checks that the detector response is higher than a pre-defined threshold and that it is a maximum in a windows of 3x3 pixels.

   (d) For each of the potential maximum, it checks that the response is a maximum with respect to other key-points form level i+1 and i-1

   (e) The 2D position of the key-points is estimated with a sub-pixel accuracy by fitting a 2D quadratic function to the determinant of the Hessian response (3x3 pixels neighborhood) and finding its maximum.

   (f) The invariance of the rotation is obtained by stimating the main orientation of the key-points as in KAZE

The results of using this algorithm to calculate the local features of the object are compared with the main characteristics of KAZE, SURF and SIRF algorithms. In this case:

- **The repeatability** scores using A-KAZE are close to the ones obtained by KAZE (20% better in comparison with SURF, 40% better in comparison with SIRF)
- **The velocity** of computing it increases in several orders. A-KAZE is faster than SIRF, KAZE and SURF.

3.2 Final Features Proposal

Taking the results obtained in the review of different techniques to extract information that we explained before into account, we created our implementation using the library OpenCV 2.4.9. In this version, the first 3 algorithms (Color Histogram, SIFT and SURF) are implemented inside but the final two (KAZE and aKAZE) aren't. If we would like to use it, we would need to wait at the stable new version of OpenCV 3.0 to use it in our environment. Externally of OpenCV, we can install aKAZE package and compile it to use. But the implementation of these packages increases the implementation time, because there are incompatibilities with different libraries of both packages. So, finally, in this version of the project we will use the SURF algorithm to extract the features of the objects and take the aKAZE algorithm into account to implement it in futures versions of the project.

If we remember the last part of the explanation of our code, we said that only the regions that pass all filters are candidate to compute their main features. Now, the next step is to crop the image to reduce the size of the frame taking the size of the detected object into account. The finality of this process is to reduce the area as much as possible when applying the extractor features process, reducing the computational time as much as possible. To study all the methods explained before, we created a specific function for each one of the algorithms, facilitating the task of comparison and commented the results we obtained by using it.

Finally, we decided to use the SURF algorithm but, to evaluate the behaviors of the different methods we explained before, we need to implement the different methods to select the algorithm with better results and less computational time for our application, because the velocity of the extractor features is an important issue to make it possible that the final system works in real-time. In the following points we can see the different information that the algorithm can extract:

3.2.1 Color Histogram Descriptor

The result of implementing a Color Histogram Descriptor is a vector with its length defined by the "histSize" variable multiplied by the number of image channels, one for each channel of the image. In appendix C.2.1 we can see how it is implemented.

3.2.2 SIFT Descriptor

The output of a SIFT descriptor is a vector of key-points that defines the image or, in our case, the object. The vector is the type "KeyPoints" and inside this type it stores the following information about the key-point:

- **Angle:** Computed orientation of the keypoint.
- **Class id:** Object ID, that can be used to cluster keypoints by an object they belong to.
- **Octave:** Octave (pyramid layer), from which the keypoint has been extracted.
- **Point:** Coordinates of the keypoint.
- **Response:** The response, by which the strongest keypoints have been selected.

- **Size:** Diameter of the useful keypoint adjacent area.

In the appendix C.2.2 we can see the code implementation of this algorithm.

3.2.3 SURF Descriptor

In the case of SURF algorithm, we must compute the key-points of the object like SIFT, but then we need to compute the vector of 64 explained in point 3.1.3. Additionally, we order the key-point taking the quality of the key-points into account and we only use the best key-point detected. In the output there is a matrix defined with n keypoints with a length of 64 bins in the rows. In the appendix C.2.3 we can observe the implementation of SURF algorithm. Taking its characteristics into account, we decided to use it in the implementation proposal of this project.

3.2.4 AKAZE Descriptor

In the output, we have the same structure as in the SURF Algorithm. Depending on the number of key-points found, the length of the matrix changes and the number of rows is 64 like in the SURF algorithm. In the appendix C.2.4 we can observe the implementation of aKAZE algorithm. The use of this algorithm remains in standby until the OpenCV v3.0 is in a stable version.

3.3 Association of Features Method:

The result of applying the previous techniques to compute the keypoints of the images is a group of interest points that define the object (except in the first technique, because it is a general descriptor of all the image).

The next necessary step is to group all these keypoints and create a main descriptor taking the first keypoints computed into account. To do this, we use a Bag of Features or Bag of Words to compute a main descriptor. The result of applying this technique is only one vector with n element to define the object. This form increases the facility to enter this data into classifier structures to create an automatic recognition of the objects in the system. In the following points we can explain the Bag of Word theory and how this algorithm works. Another advantage of using the Bag of Word representation is that it doesn't need to store the object images because the main information we need to use is in one vector, saving space in the memory system.

3.3.1 Implementation of Bag of Features Using a Bag of Key-Points

If we use a group of SIRF / SURF / AKAZE keypoints to create a descriptor of the object, we need to group this information using one technique to build a unique descriptor which introduces in the categorization structure. [9] [10]. Using the Bag of Features algorithm, we can build a system with this objective. Now, we will explain how the Bag of Features algorithm works.

Bag of Features is one of the popular visual descriptors used for visual data classification and it is inspired by a concept called Bag of Words, that is used in document classification. A bag of visual words of features is a sparse vector of occurrence counts of a vocabulary of local image features.

To achieve this, it usually includes the following four steps:

1. **Feature Representation:** Each image is represented by several keypoints and each keypoint is represented by numerical vectors called feature descriptors.

2. **Feature Description:** A good descriptor is one that has the ability to handle intensity, rotation and scale. For example, SIFT, SURF and AKAZE algorithms define a good features descriptor with the previous characteristics.

3. **Codebook Generation:** The last step is to convert the vector represented patches to "codebook". This codebook can be considered as a representative of several similar patches. Using this codebook, we can represent and classify different images, or, in our case, different objects in the scene.

4. **Object Representation:** Each patch in an image is mapped to a certain codeword through the clustering process and the image can be represented by the histogram of the codewords.

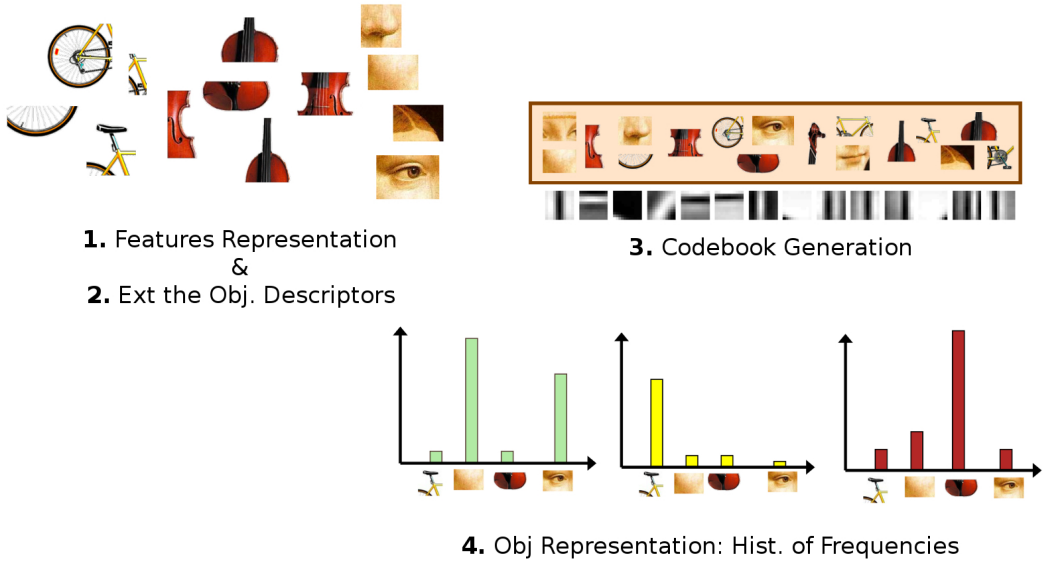


Figure 3.3: *Process of Bag of Features*

3.4 Categorization Methods

Once the image is represented in Bag of Features mode, the system needs a method to classify all images to create the recognition part of the project. To apply this in the project, we studied 2 categorization methods to create the main comparative structure and make the recognition of all objects present in the scene possible.

3.4.1 Categorization by Naïve Bayes

The classification using Naïve Bayes theorem is often used in text categorization. It is used for documents categorization according by prior probabilities. The accuracy obtained using this categorizer is typically high. To adapt this method in a visual categorization, we need to have a set of labeled images $I = \{I_i\}$ and vocabulary $V = \{v_i\}$ of representative keypoints. Each descriptor was extracted from the image using our keypoints and labeled, to which it lies closest in features space.

The classifier counts the number $N(t, i)$ of times keypoints V_i occur in the image I_i. To categorize a new image, the system applies Bayes's rule and takes the largest following score as the prediction:

$$P(C_j|I_i) \ \alpha \ P(C_j)P(I_i|C_j) = P(C_j)\prod_{t=1}^{|v|} P(v_t|C_j)^{N_{(t,i)}} \tag{3.10}$$

Using the Bayes's rule requires an estimation of the class-conditional probabilities of the key-points V_t given C_j. To compute the probabilities of zero, the system uses an estimation using Laplace smoothing:

$$P(v_t|C_j) = \frac{1 + \sum_{(I_i \in C_j)} N_{(t,i)}}{|V| + \sum_{s=1}^{|V|} \sum_{(I_i \in C_j)} N_{(s,i)}} \tag{3.11}$$

3.4.2 Categorization by Support Vector Machine

The support vector machine (SVM) is a supervised learning method that generates input-output mapping functions from a set of labeled training data. For the classification, the kernel functions transform the input data to a high-dimensional feature with the only objective of increasing the separability of the samples of the row data. The model produced only depends on the training data set used to build it.

The main objective of SVM is to find an hyperplane which separates class data with maximal margin (the distance of the closest training point to the separating hyperplane).

Given the X observations with the corresponding Y labels, the system adopts the functionality of this expression:

$$f(x) = sign(w^T x + b) \tag{3.12}$$

The parameters w and b represent the main parameters of the hyperplane. In most situations, the data set are not always linearly separable. In these cases, the SVM can use two different approaches to solve the problem.

- **Introducing an Error Weighting (C):** Penalizes classifications depending on the distance from the classification boundary.
- **Changing the Dimensions of Representative Data Space Modifying the Types of Kernels:** This new data changes can be explained with the new kernels generation 3.13. Finally, the decision function can be expressed in kernel formulation as 3.14.

$$K(u, v) = \Phi(u)\Phi(v) \tag{3.13}$$

$$f(x) = sign(\sum_i y_i \alpha_i K(x, x_i) + b) \tag{3.14}$$

But both parameters (kernel and the penalty C) are dependent and need to be determined depending on the final implementation. Finally, for our application, we need to distinguish different objects. For this reason, we need to apply a multi-class SVM classification. To make this possible, we have two options:

- **Multi-class SVM Problem:** Using a SVM structure can make multi-class classification taking different regions in the hyperplane into account. The advantage of using this option is that using only one SVM structure for all objects, it is easier to build the system and to add new objects to recognize it.
- **Two-class SVM Problem:** In this case, only two possibilities can be recognized. We need to build one SVM for each object we had in the data-base to distinguish each object from the rest of the objects in the system. The main problem of this method is that in all cases we need to compare the actual object with the rest of the objects in the system, increasing the computational load and the recognition time that the system needs to spend. However, using this option can increase the good results of the classifier because the samples only need to be separated in two groups.

OpenCV offers different configurations depending on our necessities and the difficulty to make the classifier obtain the results needed. We can configure all the Suport Vector Machine parameters using $CvSVMParams$ register.

In this register, we can configure important aspects of the behavior of the classifier in two main parameters:

1. The type of the classifier:
   - **CvSVM::C_SVC:** Builds a ($n > 2$) classifier. In this mode, the system has a penalty C taking the distance of the sample in reference to center of the class into account.
   - **CvSVM::NU_SVC:** Builds a n-class classification with possible imperfect separation. Parameter $\nu\{0, 1\}$
   - **CvSVM::ONE_CLASS:** . All the samples are from the same class. It builds a system that can separate the class from the rest of the features space.
   - **CvSVM::EPS_SVR:** The distance between the training set and the fitting hyper-plane must be less than p. Otherwise, the penalty multiplier C is applied.
   - **CvSVM::NU_SVR:** The same of the previous type, but using ν insted of p
2. The type of SVM kernel:
   - **CvSVM::LINEAR:** The system applies a linear discrimination using the original feature space. This option is the fastest one, but also the one with most problems of classification. This option is a good decision to classify separated samples.

$$K(x_i, x_j) = x_i^T x_j \tag{3.15}$$

   - **CvSVM::POLY:** When the difficulty of separating the samples increases and it is impossible to separate the samples with the linear kernel, a n polynomial kernel can be used to increase the efficiency of the classifier.

$$K(x_i, x_j) = (\gamma x_i^T x_j + coef_0)^n,\ \gamma > 0 \tag{3.16}$$

   - **CvSVM::RBF:** In this case, the kernel adds the radial basis function to the classification system to measure the influence of the samples taking the distance into account. The γ defines how far the influence of a single training example reaches. Low values meaning "far" and high values meaning "close".

$$K(x_i, x_j) = e^{-\gamma||x_i - x_j||^2},\ \gamma > 0 \tag{3.17}$$

   - **CvSVM::SIGMOID:** The kernel classifier uses the following equation:

$$K(x_i, x_j) = tanh(\gamma x_i^T x_j + coef_0) \tag{3.18}$$

3.5 Object Tracker

If now we take the initial scheme of figure 1.2, the object tracker system was considered to increase the quantity of the information that the system knows of the objects. Using this dynamic information, the system can know for example:

- If the user interacts with the object or not.
- The temporal studies of the interaction of the object.
- Monitory the movements of the objects.
- The study of the objects trajectories to detect the interaction between objects.
- The detection of user's task monitoring the movement of two objects at the same time.
- Classify the objects in different tasks to recognize more than one activity at the same time.

If the static information that the system has is completed with the dynamic information, the system can grow in complexity and will make the detections of more complicate activities in comparison of only using the static information possible.

Taking these advantages into account, we decided to implement an easy object tracker to evaluate if the information could be interesting for monitoring the user's activities.

To implement this first approximation, we used the CamShift algorithm [11] to implement this object tracker initial solution. This algorithm is a robust method of finding local extrema in the density distribution of a data set. The descriptor is used in its formal statistical sense. This means that it ignores the data points that are far from peaks in the data. It does so by processing the points within a local window of the data and then moving that window reference.

The CamShift algorithm can be summarized in the following steps:

Algorithm 3 CamShift algorithm

Input: Object Image

Output: The centroid of the object and the size of the windows.

1: **Set the region of interest** (ROI) of the probability distribution image to the entire image.
2: **Calculate a color probability distribution** of the region centered at the search window.

3: **while** (flag==true) **do**
4: **Calculate a Color Probability Distribution** in the region of the search window.
5: **find the centroid** of the probability image.
6: **Center the search window** at the location found in Step 4
7: **Set the window size** to a function of the zero moment.
8: **end while**

In the Figure 3.4, we can see one example of the CamShift tracker output.

Figure 3.4: *Ouput of CamShift algorithm*

3.6 Proposal

Before considering all the previous information to elaborate our proposal implementation, we begin this point by presenting the main structure blocks we built to extract all the information about the objects:

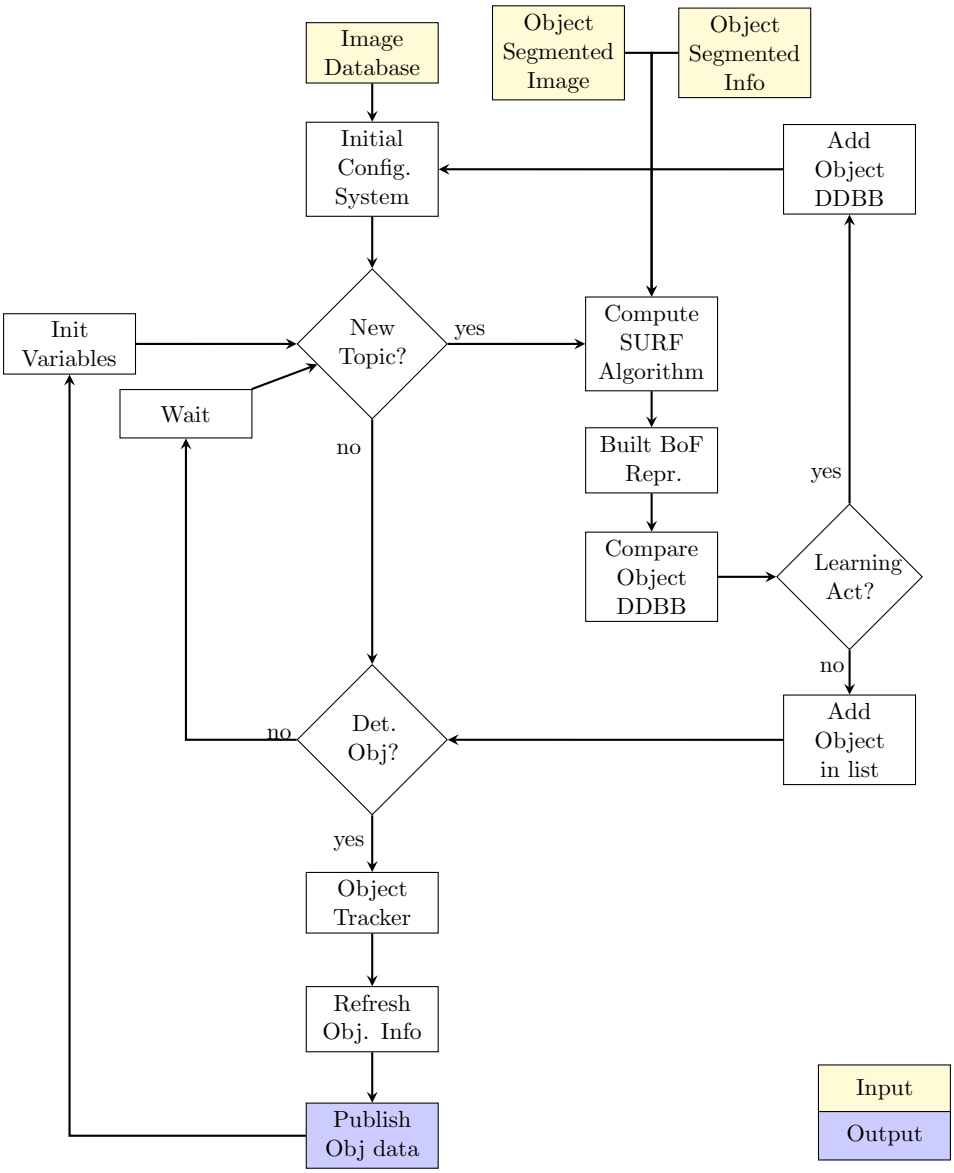


Figure 3.5: *Diagram of the Object Recognition*

In the previous section 2.3, we explained how the objects inside the scene can be detected and how the system manages the information to consider one region detected like an object. Then, the system publishes all the information of interest about the object using two different topics in ROS environment. Now, the following step is to process all this row data and convert it in interesting information about the objects. But, if we see the figure 3.5 again, we observe that the process needs to take different working modes into account to transform the raw data in data. In the following points, we will explain all the functionalities we implemented in the system and how it works step by step.

3.6.1 Reading the ROS Topics:

The first step we need to do is to read the ROS topics to start the process. In our case, we distinguish two main behaviors:

- **Compute the Characteristics of Static Objects:** It is necessary to compute the main features of the objects detected by the previous code. For this part of the code, we subscribe the topics created previously:
  - **/objects/segmentation object:** In this topic, the system shares the regions of the images detected like objects.
  - **/objects/segmentation object info:** In this topic, the system shares the main specification of the region. It is an extra information about the image that we can use in following steps. In this topic, it shares information such as: area of the region, position of the object in a camera frame, actual time or the frame camera ID.
- **Compute the Dynamic Characteristics of the Objects:** In case that the objects are in movement, we need to know the main characteristics that define the movement. This information is important for the followings steps to recognize the main activity. To make this part, the system needs to monitor the objects to determine if they are moving or not. We will talk about the object tracker in followings points.
  - **/cam2/hd/image color:** Image of the scene.

Only when the system detects that both topics are actualized, it starts to manage the information over all blocks of the system. But first we need to do one previous step, the initial configuration of all main parameters to allow the system the classification of the objects. To manage these different steps, we implement in the code one menu to allow the system to change the execution depending on the configuration. In the appendix C.2.5 we can see this menu.

3.6.2 Creating the Image Database:

An important first task is to create an initial database of all the objects we need our system to distinguish. To make this task easy, we used the image object segmented topic we had. The main process of this part is the following one:

1. **Waiting for a New Publish of an Object Segmented:** The system waits for new objects to appear.
2. **Check if the Object is Correct:** You can visualize the new region of the image caught by the system. The system asks if you would like to capture this image to add it on the database.
3. **Introduce the Label Number of the Object:** We need to add the label and the type of the object image to classify the data. This information needs to be entered by the user.
4. **Save Image to the Database:** Taking the label of the object and the number of samples into account, the system saves the image with a specific name.
5. **Repeat** all these steps until the user stops the process.

In the appendix D and C.2.6 we can see all the objects in the database and the code of this process.

3.6.3 Computing the Descriptors of the Image Database

When we have an initial image database created, the next step is to compute the main features of each image to define the characteristics of the objects. To make this, we take different solutions to know the main characteristics of the objects into account, as explained in previous points 3.2, the different techniques used to know the main features of the object.

In the first moment, we use 2 type of features to define the objects:

- **Global Features of the Object:** Using the histogram of the color, we can know the main color of the object.
- **Key-Points Features of the Object:** Using SIRF, SURF or AKAZE, we can know the main characteristics taking the form, corners and more interesting parts which define the object into account.

Using these two types of features, we built a vector using the Histogram Color as a Global Feature and SURF algorithm as an interest key-points feature. Using this vector, we could build a system that considers the general parts of the objects and the main little characteristics that define the object (form, labels, etc...).

The results we obtained with this structure were not the ones we expected. The system had a lot of problems with the object classification when there were similar objects, though there exist differential elements in both objects. This behavior shows that, for the system, the main color of the object is more important in comparison with SURF key-point.

For this reason we decided to use only the SURF key-point extractor to try to define the object because we had better results and speed of computation.

Once the SURF key-points were computed, we needed to adapt the results in Bag of Features description form to prepare the information to introduce in the comparative structure. In the appendix C.2.7 we can see the process to know the characteristics of the objects.

And, finally, to implement the classification structure we decided to use the Support Vector Machine. Considering all the configuration explained in the point 3.4.2, we started to test different configurations to find the best results of our application.

The main consideration that we had in building the classification structure was the necessity to build a SVM to try to classify all the objects we had in the database. This consideration makes the implementation of the SVM easy because it doesn't need a SVM for each class and the system can grow easily if we need to add more objects in the future. On the other hand, the existence of a multi-class SVM structure can make the system more complex increasing the errors during the classification process.

To evaluate the best configuration of our project, we implement different SVM configurations, trying to see what produced less errors using less computational time. The classification time is an important parameter to take into account, because the system must work in real-time. The results were the following ones with the next configurations:

- **Using Linear Kernel:** In this case, using a linear kernel is the easiest way to implement SVM structure, but the results we obtained were not so accurate. There were a lot of false classifications that worked wrong in the system. We tried to increase the number of samples in the learning process but the results didn't improve.
- **Using Polynomial Kernel:** In this case, the results were better than in a previous configuration depending on the number of degrees configured. Using little degrees, we had the same problems as in the first case, but this bad results decreased by increasing the degrees configuration. Degrees greater than 7th don't get changes in the classification process.
- **Using RBF Kernel:** It is the most general kernel. Using this type of kernel we observe that it produces accurate results, similar to the polynomial kernel but using less computational time to classify.

Once all the initial results we obtained in a primitive environment were valued, we decided to use RBF kernel because it obtained a good accuracy of detection using less computational time in comparison with the polynomial kernel, that uses more time. In any case, this time for our application it is not a constraint because both kernels work with enough velocity for our application. In the following code we can see the configuration of the main parameters of the SVM we used:

```
//SVM parameters gaussian kernel params

params.svm_type=CvSVM::C_SVC;
params.kernel_type=CvSVM::RBF;
params.gamma=0.50625;
params.C=312.5;
params.term_crit=cvTermCriteria(CV_TERMCRIT_ITER,100,0.000001);
```

Finally, using the training data, the labels of the objects and the parameter configuration, we start to make the learning process, preparing the system to start the classification part.

```
 printf("%s\n","Training SVM classifier");
 bool res=svm.train(trainingData,labels,cv::Mat(),cv::Mat(),params);
```

3.6.4 Working Mode:

When the system has all the information about the objects, the classification is trained and the other parameters are configured, the system can start to make the recognition of the objects. The main objective of this part of the code is to evaluate the new object samples that appear, trying to make a prediction of the new object in the scene and to manage this information in an efficient way to prepare it to publish. In the appendix C.2.8, we can see the code of this process.

3.6.5 Object Tracker

The object tracker was implemented using the CamShift algorithm explained in the chapter 3.5. The system remains waiting until it statically finds an object. From this moment, the object tracker uses the contextual information from the objects that it obtained on the previous steps. Using the centroid and the size of object window, the object tracker starts the process, calculating the color probabilities in the initial position, and finally, once this information has been calculated, it is stored in the object detection vector, preparing the process to allow the tracker of the objects. The system uses the waiting periods between ROS messages refreshments to monitor the objects using the object tracker, increasing the efficiency of the system taking advantage of the Idle times.

In the appendix C.2.9 we can see the main code we used to implement this object tracker.

3.6.6 Object Publisher List

Finally, when the system finishes the process of detection and classifies all the objects inside the scene, it sends the information captured using a specific ROS message to share it in the main Robot ROS environment.

The information that the system shares is the following one:

- **Header:** In this part, we share a general information about the message:

  - Seq: It is the number of topics that the system published in the execution.

  - Time Stamp: The actual time of the system.

  - Frame id: Here, the system publishes the origin of the message. In the case that there is more than one camera, we can recognize the origin monitoring this part of the message.

- **Name:** We publish the name of all the objects detected in the scene. We present this information using a vector to facilitate the representation of the objects in the followings steps.

- **Id reference:** It is the identification reference of the system. This number is related to the objects database folder.
- **Type:** The system recognized 3 types of objects:
  - **"1":** Containers
  - **"2":** Tools
  - **"3":** Raw materials
- **Validate:** The algorithm adds a validation parameter to increase the quality of the detections. It is a boolean value. When the system detects the object more than 4 times, the system defines it as validated. Using these parameters, the false detection can be detected and discarded, increasing the quality of the detections.
- **Status object state:** The system can distinguish 3 different states of the objects:
  - **"0":** The object is waiting to be used.
  - **"1":** The object is being used.
  - **"2":** The object was used.
- **Obj det counter:** It is the number of occasions that the system recognizes each object.
- **Position x and y:** It is the situation of the objects in the image coordinates frame.
- **Start movement time:** It is the moment when the system detects that one object starts being used.
- **Total used time:** It is the total of time that the system detects each object in movement.

3.7 Example Output

In the following picture we can see an example of the output of the process.

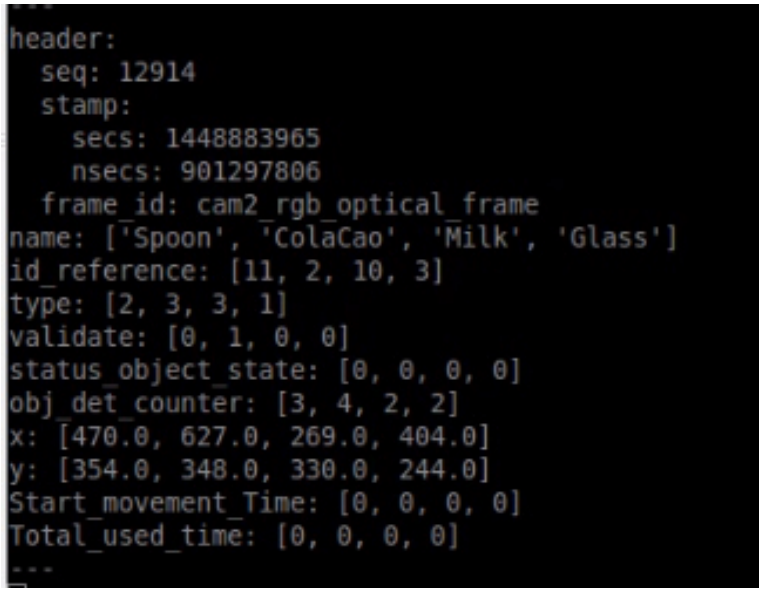


Figure 3.6: *Output Ros Topic Recognition Example*

Chapter 4

Activity Recognition

The Activities of Daily Living (ADLs) is a term used to refer to people's daily activities such as feeding themselves, bathing, dressing, grooming, work, leisure and homemaking. In our case, we are only centered in the homemaking in the kitchen environment. If we can model, monitor and predict these ADLs using different sensors installed in the environment, the robot system can use this information to condition the behavior of the robot automatically depending on the predictions that the system has made. In our case, we focus the attention only on the kitchen ADLs.

In the following chapter, we will introduce the implementation made to do the prediction of the activity using the contextual information. In our case, the system used the information provided of the objects to evaluate and predict the actual activity that the user is doing and introduce future information of the activity that the system could consider to change its behavior in an automatic way. Another important issue to consider is the evolution of the activities. The activities are in constant evolution, so the system needs to take this plasticity of the activities into account to maintain the parameters of the system always actualized.

In the following parts of this chapter, we will introduce how the "ADLs" can be recognized and, then, we will also explain the final implementation we made.

4.1 State of the Art

One of the methods of detecting the ADLs activities is by modeling the activity using the Activation Spreading Network(ASN)[13]. This method was inspired in hierarchical task networks, which is a way to represent the hierarchical relationships of a process or activity.

Using this method, the system can represent the activities distinguishing the high-level and low-level activities. High-levels represent complex activities and low-levels represent the minimal unit interaction that the user can make. So, the high-level activities are defined with "n" number of low-activities represented using a decision tree scheme. In most cases, the representation of the information used by Hidden Markov Models (HMMs) facilitates the task of classification of activities in a vision-based system. The nodes represent the different activities and the edges represent the relationships between this activities. The low-activities can represent the actions in the high-activities to make it.

Another important information to detect the activity is the use of the contextual information about the environment. The external conditions can be used as features to distinguish different activities. One example is that there are different activities that are only done in a determinate period of the day, or in determinate weather conditions or seasons of the year. So, the system can use this information to discard activities if the user doesn't usually do them in those conditions.

Another approach to find the activity of the user is by monitoring the hands of the users [14]. In this method, the system uses two types of features:

- Global features using PCA on the gradients of 3D hands trajectories.
- Local features using Bag of Words of snippets of trajectory gradients.

For the action recognition, the system uses the information of the hands trajectories, the events of doing an action, the pre-trained action models, the objects used in the actions and the pre-trained object models.

In another way, in [15] is explained the main activity of how the users can be monitored adding sensors in different places of the house and how the activity can be classified using a naive Bayesian classifier to detect the activities.

Taking the previous information into account, it is important not only to capture the objects to make a prediction of the task that the user is doing in this moment, but there are also other important informations that we need to know to make a good prediction. Including:

- **The Objects Inside the Scene:** It is important to know this information because it is a good method to define an activity. Normally, in all the activities there is one or more than one element that define the actual activity and we can use it to find it.
- **Order of Interaction of the Objects:** One activity is not only defined by the number or types of objects. We also need to know the order in which the user uses all the objects. It is possible that two activities have the same objects or that they are distinguished by one or two elements. In this case, the dynamic studies of the movements/interactions of the objects is an important feature to take into account because it is another element that defines the activity.
- **Contextual Information:** Another important factor to discard activities is the external conditions in which the users do the activity. Normally, the users do different activities in different periods of time throughout the day, or in determinate weather conditions (only when it is raining outside , it is a sunny day) or it is probable that one activity is more probably done depending on the season of the year. We can use all of this information about the environment to find the activity taking also the features explained before into account.
- **Wearable Sensors:** Another important information that we can use to find the most probable activity that the user is doing is the user's information. Nowadays, the miniaturization of the sensors allows the integration in different devices. For example: clothes, wrist band, intelligent clocks, smart-phones and integration of the sensors inside the body of the users are examples of these integrations that allow to monitor the most important parameters of the users. All these possibilities create a new environment to capture the personal data to use in different applications. This concept is known as **Internet of Things "IoT"**. The user activity is conditioned by the state and the activities of the users.

On the other hand, there are different characteristics that are accomplished to assure a good behavior of the activity recognition system. For example, we can highlight the following characteristics:

- **Database Generation:** The system needs to capture the actual state of the system and it needs to compare it with previous iterations of the same activity. To do that, we need to build a new database to store all the information about the activities that the system can detect.
- **User Learning:** It is important that the system can adapt to the necessities of each final user and change its behavior when the users also change.
- **Work in Real-Time:** The system needs to be able to make predictions during the time that the user is doing the activity.
- **Show the Future Information about the Activity:** If the system can work in real-time, it can show different information about the following steps of the activity detected.
- **Adapting the Process to New Features:** It is possible that the system needs to grow to add new features to increase the accuracy of the detection. The system needs to add new features in an easy way.

4.2 Proposal

In the following section, we will explain the final implementation we made to categorize and recognize the activities considering the previous information in previous points. The main scheme of our proposal are the following one:

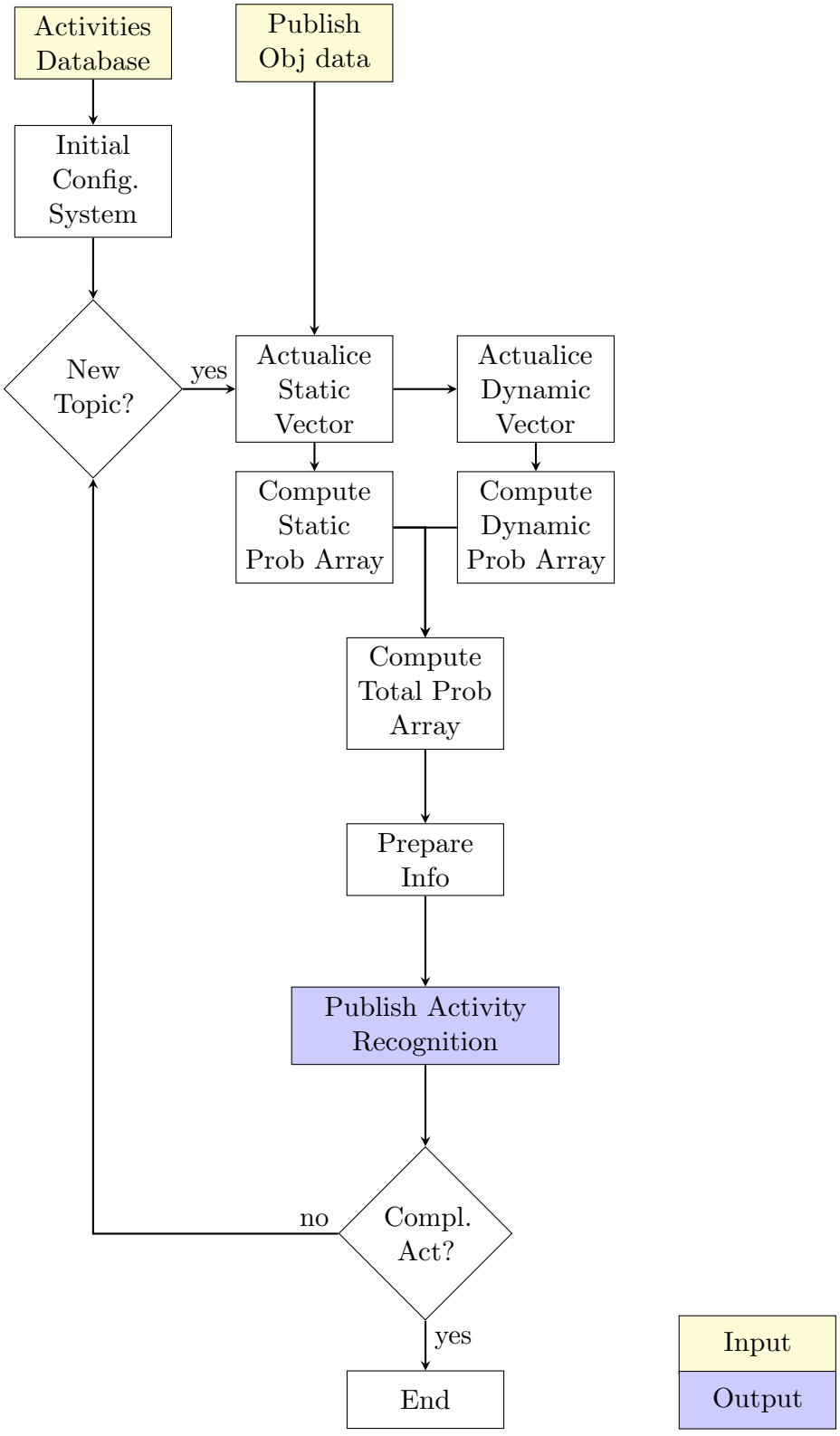


Figure 4.1: *Diagram of the Act. Recognition*

One characteristic of the system is to produce the estimation of the activity. First, the system needs to use information created in the previous steps of the process. That means that we need to assure the coherence of the data in all the steps of the process, avoiding the wrong behaviors when the final user changes the conditions of the system. For example, when the user adds new objects to recognize. To adapt to this changes, the system builds new data structures in each execution taking information of the different files created in the previous steps assuring that the information is actualized. Another important issue is the velocity of the system to recognize the activity. Although the application doesn't need high recognition velocities, the system needs to assure that the prediction works in real-time, doing more than one prediction during the execution time of the activity. The adaptation of the changes in the execution of the activities is another important point to consider in the development of our proposal.

Finally, the row data we used to recognize the activity was only the first two type of data. The reason is that in this moment we didn't have the main hardware structure to capture the information about the contextual and user's factors. But we prepared the system for the possibility that in the future this type of data may be added and increased the features used by the system.

Now, in the following points, we will explain all the process of the activity recognition parts step by step.

4.2.1 Reading the ROS Topics

First of all, the activity recognition code needs to subscribe and create topics to manage the main information about the process.

The list of the topics we need to use is the following one:

- **/objects/Object:** Subscriber to the topic of the list of objects that the system created in the part of object recognition 3.6.6.
- **/Activity Detection:** We created this new topic to store and publish all the information about the final detection activity of our system, sharing the information in the ROS environment for its use in another process.

The first task we need to do is to actualize the internal parameters where the list of the objects is stored. To do this task, we built a callback function to actualize the internal list of the objects when the system detects new publications of the object topics. In the appendix C.3.1 we can see the code of this process.

4.2.2 Selector Menu

To configure the system to start the activity recognition process, we need first to build a selector menu to change in the different steps of the process. The three processes we need to do are the following ones:

- **Load the Initial Data from Files:** This part captures all the information stored in different files to start to work.
- **Add New Activity:** The task of this part of the code is to actualize the activity files to add new characteristics to our system in an easy way. This part has not yet been implemented.
- **Activity Detection Mode:** When the previous step has finished, the system is ready to start to work. The function of this part of the code is the detection of the activity using the information loaded in the previous steps.

In the appendix C.3.2 we can see the menu selector.

4.2.3 Load Initial Data from Files

One of the tasks we need to do to start the process of the activity recognition is to load all the parameters that define the user's activity in the system. In this project, we define the activity with the following types of information.

- The objects used to do the activity
- The interaction of the objects when the user is doing the action.

In the point 4.2 we explained other types of data that we can use to define the activities: The external factors (weather, times, etc..) and the personal user's information, but we finally decided not to use it. The hardware resources of the group don't allow to integrate this type of information in the project in an easy way because we would need to implement it. But the solution we created has the possibility to integrate this information and to use it in the activity recognition code.

The database of the system uses two different files where all the information about the activities that the system can recognize is stored:

- **"activity file.csv":** In this file, all the information about what objects participate in the different activities is stored.
- **"activity file dynamic.csv":** In this file, we store how the users interact with the different objects, and how these interactions define each activity in the system.

Another important issue we need to take into account to load the information is the necessity to assure the structure of the data that was created in the previous codes. Probably, in the previous steps of the project, can be produced little changes that must be taken into account to load this information in the system. For example, we need to assure that the changes in the order of the objects or the addition of new objects in the system will not produce errors in the execution. For this reason, we used the information stored in the file like a guide to build the new variables to fusion all the information of the different steps. Now we will explain the process of each file in a depth way:

1. We need to load the "object data.yml". This file contains the name, id number and type of all the objects that the system can detect. This file was created during the extraction of features objects process 3.6.3. When the system adds new objects, it saves this new information in this file. So we always have to actualize the information of the objects that the system can detect.
2. The "activity file.csv" file stores the activities that the system can detect and how the objects interact in the activity.
3. All this information has to be saved in different variables in the same structure of data.

The following steps that the system needs to do is to load the parameters from the other file "activity file dynamic.csv". Here, the system can load the order of the previous interaction of the objects implicated in each activity. In this file, we can find different ways to do the same activity. We can use all these iterations of one activity to calculate the probabilities that define one activity and now, using this information, we can build a hierarchical representation of the activity represented by the probabilities.

The main steps we need to do in this process are the following ones:

1. Load all the iterations in the same matrix
2. Filter the information using the number id of the activity
3. Compute the probabilistic changes of the object used order.

4. Save the information in a vector of matrix.

The information in the output is the following one:

- **Vector** [**Matrix**]**:** Stores the information about all the activities.
- **Matrix:** Stores the information about one activity.
- **Columns:** The position of the column represents the id objects in the system.
- **Row:** The row position represents the objects' movements.
- **Values:** Stores the probability that one object intervenes in the activity in a determinate oder.

4.2.4 Activity Detection Mode

In this part of the code, we implemented in real-time the detection of the different activities that the user is doing. Taking the type of data that we can use explained in the point 4.1, we can divide the study of the detection of the activities in four big blocks:

1. **Static Study of the Activity:** The system does the evaluation of the objects inside the scene and tries to identify the activity using this "static" information.
2. **Dynamic Study of the Activity:** The system monitors the order in which the final user uses the objects and tries to identify the order taking the past experiences of the execution into account.
3. **External Factors:** We have not implemented it now, but we take it into account in this version to facilitate the integration task in case it could be implemented in the future.
4. **Users Factors:** We have not implemented it now, but we take it into account in this version to facilitate the integration task in case it could be implemented in the future.

The main strategy to compute the probability of activity that we applied, is to compute the probability layer by layer. In our case, we defined two different layers to compute the probability. But to resolve the possibility of adding more layers, we decided to compute each probability layer separately. Using this method, we can add new layers depending on the future necessities. Then, once each probability is calculated, we need to build an expression to relate all these independent probabilities in the same relationship. Now, we will explain how this part of the project was implemented.

Static Study of the Activity:

To do the static activity probability, we only used the information stored in the file called "activity file.csv". The steps we need to do are the following ones:

1. We need to build one vector using the size of the object database:

```
cv::Mat activity_detect = cv::Mat::zeros(1,objects_db.size(), CV_32F);
```

2. Now, we can put the objects that the system detected in this vector.

```
for (int i=0;i<Object_list.id_reference.size();i++)
{
 activity_detect.at<float>(0, Object_list.id_reference[i]-1, 0) = 1;
}
```

3. For all the activities that the system can detect, we need to compare the actual execution with the activity definitions in the database. We use it to do the comparison and operator and count the number of coincident objects:

```
cv::Mat temp;
bitwise_and(load_activity_db.values.row(i),activity_detect,temp);
float coincidence_object_model=countNonZero(temp);
```

4. We need to know how many objects are implicated in each activity. So, now we find this number:

```
cnt_model_object=countNonZero(load_activity_db.values.row(i));
```

5. We also need to know how many objects the system detected in the scene and weren't defined in the activity database, creating a penalty value used to discard options:

```
penalty_value=countNonZero(load_activity_db.values.row(i)-activity_detect);
```

6. In this moment, we need to build one expression to relate all this information we have of the activity. The penalization value was weighted to 0.5:

$$Stat_{Act\ Prob} = \frac{Coincidence_{Object\ Model}}{Cnt_{Model\ Object}} - 0.5 * \left(\frac{Penalty_{Value}}{Cnt_{Model\ Object}} \right) \tag{4.1}$$

7. Finally, all the information is stored in the memory. We store the name of the activity and the probability value in different vectors on the same structure.

```
activity_rate.activity.push_back(load_activity_db.activity[i]);
activity_rate.rate.push_back(((coincidence_object_model/cnt_model_object)-0.5*(
  penalty_value/cnt_model_object)));
```

Dynamic Study of the Activity

In this part of the code, we implemented the determination of the probability that one activity is being done by using the information from the file "activity file dynamic.csv". The steps we need to do are the following ones:

1. First, we need to create the vector used to monitor the order of the object iterations. When one object starts to move, the system only needs to apply one number depending on the order. In this first version of the object status monitoring, we only take the first use of the object into account.

```
for (int i=0;i<Object_list.status_object_state.size();i++)
{
  if ((Object_list.status_object_state[i]==1)&&(dynamic_activity_detect.at<float
  >(0,Object_list.id_reference[i]-1, 0) == 0))
  {
    dynamic_activity_detect.at<float>(0,Object_list.id_reference[i]-1, 0) =
  working_dynamic_cnt;
    working_dynamic_cnt++;
  }
}
```

2. Second, we use the probability data extracted from the information in the second file "activity file dynamic.csv" to compare how the actual activity seems with the database register. When the order is correct, the system adds it in the dynamic probability value positive factor. When the system detects that the user is doing the activity in a different order that the one stored in the database, the system assigns another penalty value. The dynamic equation we built to relate all this information is the following one:

$$Dyn_{Act\ Prob} = \sum_{i=1}^{N_{Goodmov}} \frac{1}{N_{ObjMoved}} - 0.25 * \sum_{i=1}^{N_{Badmov}} \frac{1}{N_{ObjMoved}} \tag{4.2}$$

```
for(int j=0;j<activity_prob_matrix.size();j++)
{
  float dynamic_prob=0;
  for(int i=1;i<=max; i++)
  {
    cv::Mat comp= cv::Mat::ones(1,objects_db.size(), CV_32F)*i;
    cv::Mat l_search=abs(dynamic_activity_detect-comp);
    double m, M;
    Point p_min, p_max;
    minMaxLoc(l_search, &m, &M, &p_min, &p_max);

    if(activity_prob_matrix[j].at<float>(i, p_min.x+1, 0)!=0)
    {
      dynamic_prob=dynamic_prob+1/max;
    }
    else
    {
      dynamic_prob=dynamic_prob-(0.25*(1/max));
    }
    movement_order++;
  }
  activity_values.Dynamic_rates.push_back(dynamic_prob);
}
```

Total Study of the Activity

When all the independent parameters are calculated, we can merge all the values in the same expression to find one unique value to use in the detection of the activity.

To calculate this unique value that relates all the activity detection layers, we used two different expressions taking into account if the activity started or not:

If the activity doesn't start to move, the dynamic activity detection part doesn't contribute in knowing the activity. For this reason, we didn't use it to calculate it:

$$Total_{Act\ Prob} = 1 * Stat_{Act\ Prob} + 0 * Dyn_{Act\ Prob} \tag{4.3}$$

Then, when the system detects that the user is moving the objects in the scene, the dynamic information is important to be known, because the final activity depends on the interaction that the user makes with the objects. For this reason, we used the following expression:

$$Total_{Act\ Prob} = 0.65 * Stat_{Act\ Prob} + 0.45 * Dyn_{Act\ Prob} \tag{4.4}$$

Using the method to calculate the main probability value allows to add new layers using different datasets in an easy way. For example, we can add the external information about the environment or the user information we explained in the point 4.1. Finally, we can change the expression 4.4 to take all layers we will have into account.

4.2.5 Activity Publisher

The final part of the activity recognition is to implement the best method to manage all the information we generated in an efficient way and publish it in the best possible way in the main system of the robot environment. Now, we will start to explain step by step how we implemented this part of the project.

The results we obtained in the execution of the point 4.2.4 are three different vectors:

- Output vector of static results.
- Output vector of dynamic results.
- Output vector of total results.

To determine the most probable activity that the user is doing, the system needs to use the highest value of the outputs vector explained before. Depending on these score values that the system finds, we can know not only the activity that the user is doing, but we can also know and evaluate the quality of the detections:

- $Value \geq 0.8$**:** We assume that the system can know the activity in execution with high guarantees because the action than the user is doing is similar to one activity defined in the database of activities.
- $Value \geq 0.5$**:** In this case, the system can recognize the activity, but it detects differences between the actual execution and the database. The possible differences between both are because:

  - There are more objects in the scene than the ones the activity needs.

  - The execution process of the activity is different.

  - Combination of both situations.

  So, we need to detect the origin of the difference and take one decision using the information that the system offers. The system publishes the information about the activity estimation in a separate way to facilitate this task.
- $Value \leq 0.2$**:** The system offers the most probable activity that the user is doing, but the system can't assure if the result of activity detection is good or not.

When we arrive at this point, we don't only know the prediction of the activity that the user is doing, but in this moment we have stored a lot of information about the activity that we can use to offer more interesting data at the possible subscribers of the activity detection. For example, we can know:

- **Detection Method:** Our system has the possibility to use different equations to calculate the total score of the probability depending if the activity has started or not. We explained this in the point 4.2.4.
- **Percentage Completed of the Activity:** If we know the number of objects that the user needs to do one activity and how many objects the user moves in the actual execution activity, we can publish the completed percentage of the user's activity.
- **Activity Status:** We can classify the status of the activity in three different status:

  - Waiting to Start: The system detects objects in the scene but they are not moving.

  - In Progress: The system detects movements of the objects. The system detects that the activity started.

  - Finished: When the percentage of completed parameter is equal to 100 %, the activity has finished.
- **Start, Finish and Total Execution Time:** If we know the status of the activity, we can also publish the time when the activity starts and finishes.
- **Next Movements:** If we use the database information about the activity, we can publish the next objects movements depending on the probability matrix values explained in the point 4.2.3.

4.2.6 Activity Ros Topic

To manage all the information we had about the activity recognition part, we built another Ros Topic Message to put all the important information about that in this new topic. The information that the system shares is the following one:

- **Header:** In this part, we share general information about the message:
  - **Seq:** It is the number of topics that the system published in the execution.
  - **Time Stamp:** The actual time of the system.
  - **Frame ID:** Here, the system publishes the origin of the message.
- **Static Activity:** Stores the main parameters of the activity recognition using only the static information:
  - **Static Activity Recognition:** Name of the activity recognized.
  - **Static Activity Recognition ID:** Identifier number of the activity recognized.
  - **Value Static Activity:** Result of the activity recognized.
- **Dynamic Activity:** Stores the main parameters of the activity recognition using only the dynamic information:
  - **Dynamic Activity Recognition:** Name of the activity recognized.
  - **Dynamic Activity Recognition ID:** Identifier number of the activity recognized.
  - **Value Dynamic Activity:** Result of the activity recognized.
- **Total Activity Recognition:** Stores the main parameters of the activity recognition using both methods implemented in this project:
  - **Total Activity Recognition:** Name of the activity recognized.
  - **Total Activity Recognition ID:** Identifier number of the activity recognized.
  - **Value Total Activity:** Result of the activity recognized.
- **Detection Method Indicator:** Indicates the method that the system uses to compute the activity probability:
  - **Detection Method:** Name of the method: Static or Dynamic.
  - **Detection method ID:** Identifier number of method.
- **Time Information:**
  - **Start Activity Time:** Time when one activity starts.
  - **Final Activity Time:** Time when one activity finishes.
  - **Total Activity Time:** Time that the user needs to spend to finish the activity.
- **Future Information:**

  - **Next Movements:** Stores the information about the next most probable object movements.

4.3 Example Output

In the following picture we can see an example of the output of the process.

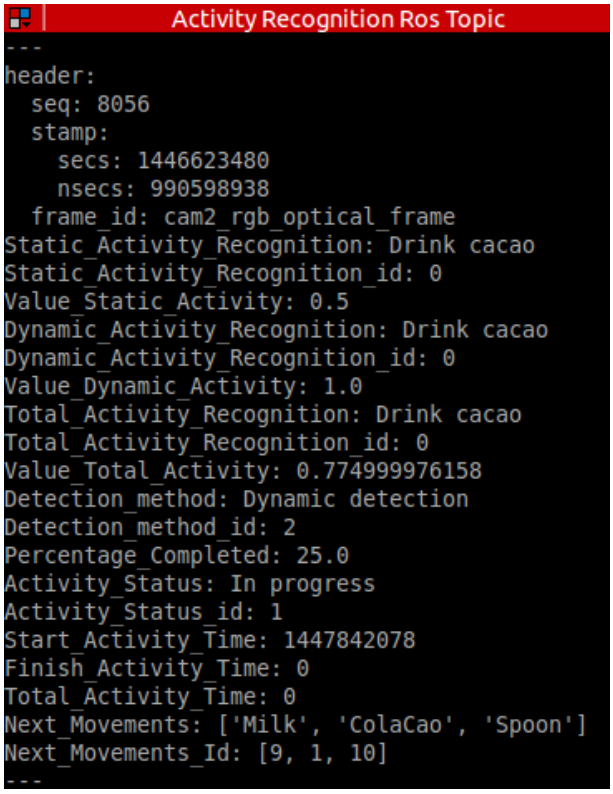


Figure 4.2: *Output Ros Topic Activity Recognition Example*

Chapter 5

Results

In the figure 5.1, we observe in a resumed way, the main blocks that our system had implemented and the flux of the information we created. In the following points, we will explain how we configured the environment and the results we obtained in the activity recognition process.

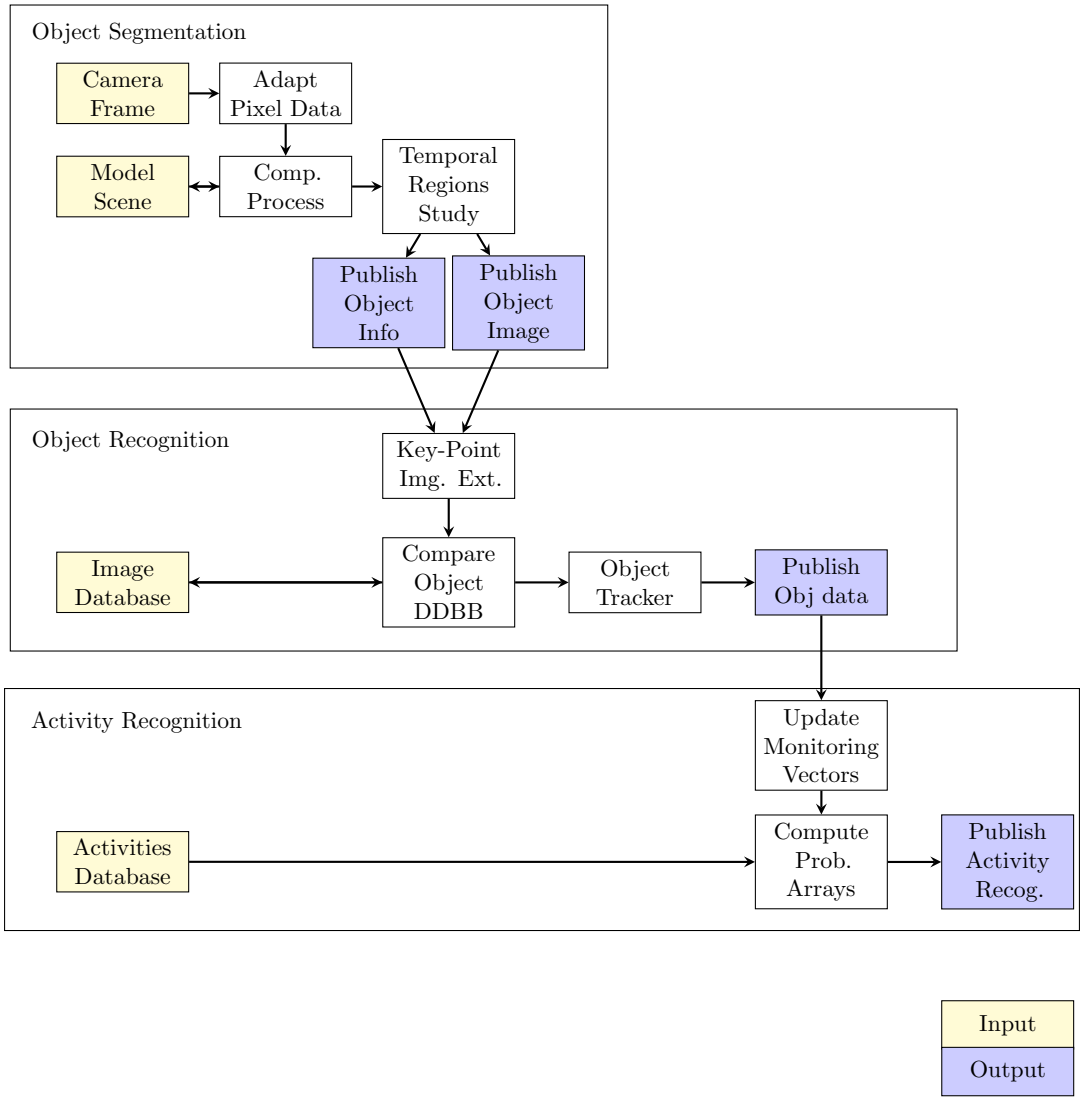


Figure 5.1: *G. Diagram of the Act. Recognition*

5.1 Experimental Setup

To carry out the experiment of the behavior of our system, we created the following test environment:

- **Number of Objects:** 16 different objects: Cut table, colacao, glass, orange, big fork, coffee, big spoon, sugar, juice, milk, spoon, tea, pan, dish, brush, heat shield. In the generation of the test database, we took 3 different types of objects which we can find in the scene of this project:
  - **Commercial Objects:** The system can obtain information using the trademarks in the container of the objects. For example: Milk, juice, coffee, etc..
  - **Tools:** Normally, the tools we used have only one dominant color and don't have any trademark to be identified. For example: Spoon, big fork, big spoon.
  - **Raw Materials without Layers:** In this case, the objects don't have any layer but the system can detect the texture of the object. For example: Orange.

  On the other hand, we made other different groups of all the objects in the database, depending if we used the objects to define the activities or not.
  - **Objects Used in Definition of the Activities:** In this case, the learning process done is more accurate and tries to capture more object samples to assure good ratios of detection.
  - **Objects Used to Build the Database:** These objects aren't used to define any activity. They have less number of samples and don't take all the possible cases into account.
- **Type of Objects:** 3 different types: Containers, tools and raw materials.
- **Number of Activities:** 6 different activities:
  - **5 Similar Activities:** Drink milk, drink cacao, drink Juice, drink tea, drink Coffee.
  - **1 Different Activity:** Peel Orange.

The external conditions we used to implement in the experiments were the following ones:

- **Status Scene:** Only the static objects remain on the scene.
- **Light:** Uniform light distributed.

5.2 Results of the Object Recognition

To evaluate the Object Recognition process, we built the confusion matrix between the different objects that the system can detect in the experimental setup. To build the confusion matrix, we captured 100 images of the objects in the scene and we captured the output of the object recognition system. The results are the following ones:

Table 5.1: *Object Confusion Matrix*

	C.Table	Cacao	Glass	Orange	B.Fork	Coffee	B.Spoon	Sugar	Juice	Milk	Spoon	Tea	Pan	Dish	Brush	H.shield	Accuracy
C.Table	98	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	98,00%
Cacao	0	93	0	0	0	3	0	4	0	0	0	0	0	0	0	0	93,00%
Glass	0	0	90	0	0	0	0	0	0	8	0	2	0	0	0	0	90,00%
Orange	0	2	3	92	0	0	0	0	0	1	0	0	0	0	0	2	92,00%
B.Fork	0	0	0	0	72	0	3	0	0	0	0	0	0	0	25	0	72,00%
Coffee	0	0	0	0	0	100	0	0	0	0	0	0	0	0	0	0	100,00%
B.Spoon	0	0	0	0	13	0	65	0	0	0	0	0	5	0	17	0	65,00%
Sugar	0	0	0	0	0	0	0	100	0	0	0	0	0	0	0	0	100,00%
Juice	0	0	2	0	0	2	0	0	91	5	0	0	0	0	0	0	91,00%
Milk	0	0	6	0	0	0	0	0	0	94	0	0	0	0	0	0	94,00%
Spoon	0	0	0	0	0	0	0	2	0	0	98	0	0	0	0	0	98,00%
Tea	0	0	10	0	0	0	0	0	0	5	0	85	0	0	0	0	85,00%
Pan	0	0	0	0	0	0	0	0	0	0	0	0	68	0	32	0	68,00%
Dish	0	0	0	0	0	0	0	0	0	0	0	0	0	93	0	7	93,00%
Brush	0	0	0	0	8	0	0	0	0	0	0	0	0	0	92	0	92,00%
H.shield	0	0	0	0	0	0	0	0	0	0	0	0	0	7	0	93	93,00%
Accuracy	100,00%	95,87%	81,00%	100,00%	77,42%	95,24%	95,59%	94,34%	100,00%	83,19%	100,00%	97,70%	93,15%	93,00%	55,42%	91,18%	89,00%

Taking the object grouping explained in point 5.1 into account, we can explain the results in a more efficient way. First of all, The results we obtained with commercial objects with trademarks have good detection scores, in all the cases over 90%. The topology of the system allows to capture the most interesting points of the objects inside the layers to make a good identification of the different objects in this classification. The number of samples of the objects to obtain this score are around to 15 samples. Only one of the objects had more problems in doing the detection. It is the tea case that the system confuses with the glass. The possible reason of that is that the database doesn't have enough samples (it only has 9 samples) or that the object image doesn't show the main characteristics of the tea case to model the object. But in any case, it has a high recognition score with an 85% of accuracy.

On the second place, the results we obtained with the tools that only have one dominant color are worse than in the first group of objects. The possible reason of that is that the system can find a big number of different key-points to define the objects. In this case, we saw different behaviors of this type of objects:

- The c.table has a good score with a 98% of accuracy using only 4 samples to make the learning object process and can maintain the good results when the system has a partial vision of the object. The reason of that is that the object only has one dominant color of wood but the wood has streaks that the system detects to model the object. And these characteristics don't exist in any other object.

- The tools that only have one dominant color and have less samples (around 4/5 samples) don't have high values, around 70% of accuracy. In the case that the object has more samples (around 12 samples) the values of accuracy increase until 90% if there are differences between objects.

And finally, the last group of objects we had in the database are the raw materials without layers. It is possible that the system needs to detect natural objects like fruits or vegetables. In this case, the system needs to detect the texture of the objects to make a good model of this type of objects. We obtained scores around 92% of accuracy of detection in this objects using around 10 samples.

But, taking the other classification groups into account, we saw that the objects that had better score results are the objects used in the definition of the activities. The reason of that is that we used more samples and more cases to define the objects in all the possible cases and this increases the quality of the detection object process.

If we see the total accuracy of the system, it is situated around the 89% taking the specifications of our object database into account.

5.3 Results of the Activity Recognition

To evaluate the activity recognition process, the confusion matrix between the different activities that the system can detect in the experimental setup was built. To do it, we repeated each activity 30 times to compute the different probabilities of the detection.

Now, to do this study of the behavior of our system, the detection of the activities was repeated 30 times changing the situations of the objects and storing the results obtained in the detection activity process.

As shown in the static study of the activity detection table 5.2, the system obtains very stable results and has a good score on the activity detection. By studying the different scores we obtained in the detections, may be seen that the system has high score values in all the cases. In the study done, was considered the possibility there are in the environment objects that don't participate in the activity or that the system detects in a wrong way into account. In this cases, it can also be observed that the quality in detections decreases, but the system detects correctly the activity that the user is doing. In the figure 5.2 all the results we obtained in the study are shown.

Table 5.2: *Static Object Confusion Matrix*

	Drink cacao	Drink juice	Drink Coffee	Drink tea	Drink milk	Peel orange	Accuracy
Drink cacao	30	0	0	0	0	0	100,00%
Drink juice	0	30	0	0	0	0	100,00%
Drink Coffee	0	0	30	0	0	0	100,00%
Drink tea	0	0	1	29	0	0	96,67%
Drink milk	0	0	0	0	30	0	100,00%
Peel orange	0	0	0	0	0	30	100,00%

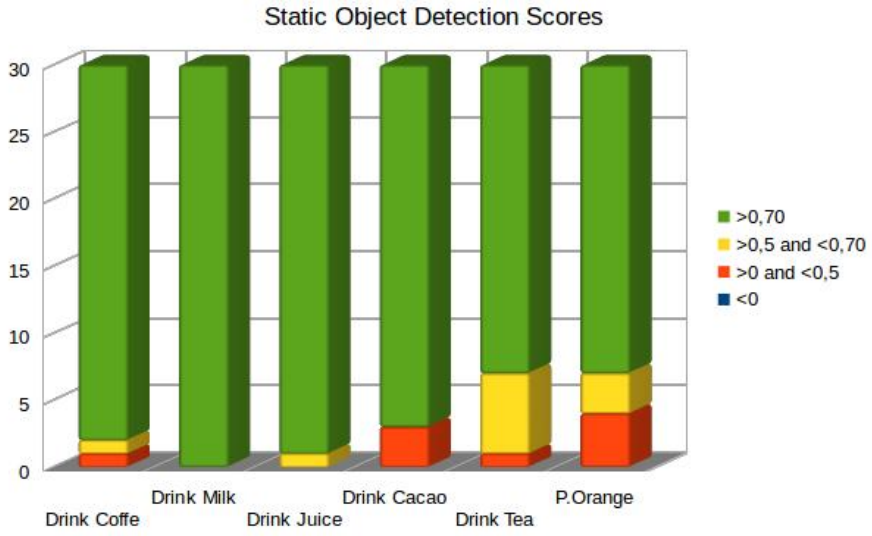


Figure 5.2: *Static Object Detection Scores*

If we take now only the results we obtained using the dynamic information 5.3, we can see that there are more discrepancies in the detection because there are more false detections. The general accuracy of the system decreases to 80.1%. The reason of that is situated in the object tracker system. We built an initial solution to track the object, and this initial solution doesn't take the situations that

produce errors into account, and this errors affect in the final score of the dynamic activity recognition part. In the figure 5.3 we can see the different values of the scores we obtained. If the user is doing one activity in which the definition is different from the rest of the activities, we can see that in most cases it has good score detections.

Table 5.3: *Dynamic Object Confusion Matrix*

	Drink cacao	Drink juice	Drink coffee	Drink tea	Drink milk	Peel orange	Accuracy
Drink cacao	23	1	3	0	3	0	76.667%
Drink juice	2	23	0	2	3	0	76.667%
Drink coffee	4	0	26	0	0	0	86.667%
Drink tea	2	0	8	20	0	0	66.667%
Drink milk	2	0	0	3	25	0	83.333%
Peel orange	3	0	0	0	0	27	90%

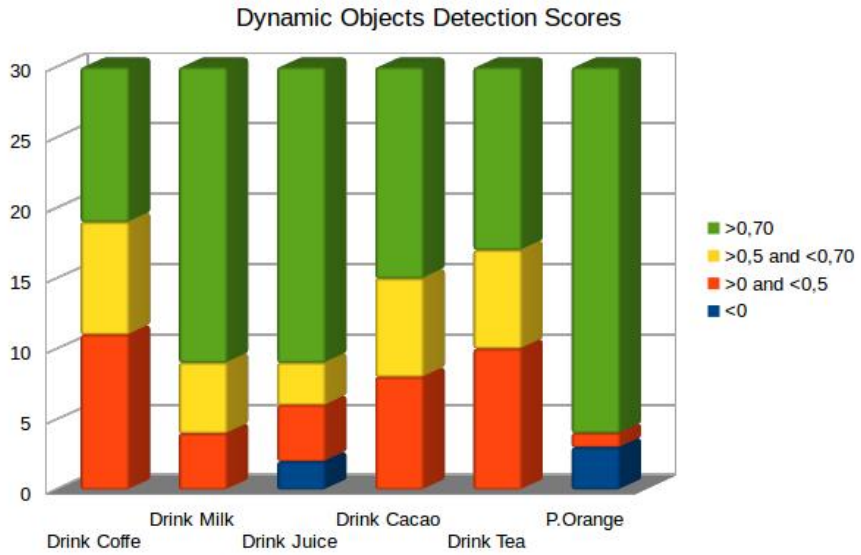


Figure 5.3: *Dynamic Object Detection Scores*

When the user is doing an activity with similar parts to other activities, the scores decrease. But the dynamic information can help the system to detect activities in more complex situations. For example, if in the scene there are at the same time the objects needed for two different actions, the system can use this information to discriminate the activity that the user is really doing.

Evaluation of the dynamic information

To evaluate the behavior of the system in these conditions, another experiment was created. We introduced the objects needed to do two activities with the same number of objects, drink milk and drink juice, in the scene. In the picture 5.4 we can observe the static, dynamic and total values of both activities that the system detects.

Taking only the static score of the system, we observe that both activities have the same score, because they have the same conditions (good and wrong objects). The system publishes the information about the most probable activity that the user is doing, but both values are the same. However, the evaluation of the dynamic execution of the activities presents differences in both activities. For one of them, the actual execution, perfectly follows the model of the activity,thus the score is the highest value "1". And we observe in the score of the other activity, that there is one part of the execution that follows the model and another part that doesn't, penalizing the differences with respect of the model

in the final dynamic score. Then, in the total score, we can observe that these differences produce the good detection of the activities.

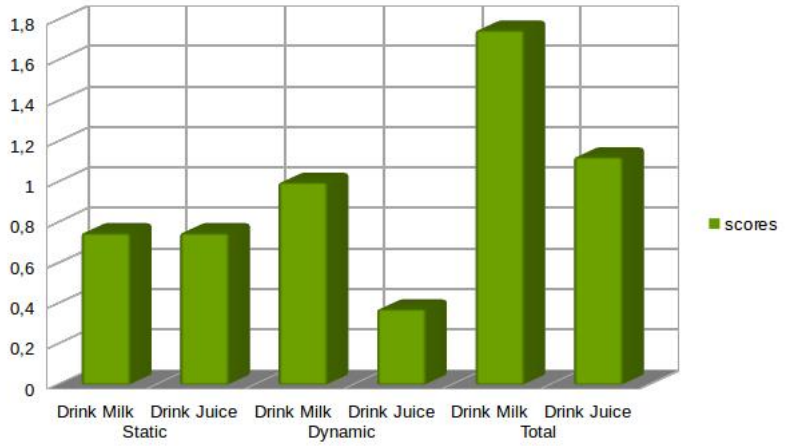


Figure 5.4: *Experiment to detect the activity in bad conditions*

Finally, in the table 5.4 the final result of the activity detection process is shown. We can see that the system remains with stable results and has good accuracy values. The figure 5.5 shows the final results score of the detection of the activity.

Table 5.4: *Total Object Confusion Matrix*

	Drink cacao	Drink juice	Drink Coffee	Drink tea	Drink milk	Peel orange	Accuracy
Drink cacao	28	0	2	0	0	0	93,33%
Drink juice	0	30	0	0	0	0	100,00%
Drink Coffee	0	0	30	0	0	0	100,00%
Drink tea	2	0	0	28	0	0	93,33%
Drink milk	0	0	0	0	30	0	100,00%
Peel orange	0	0	0	0	0	30	100,00%

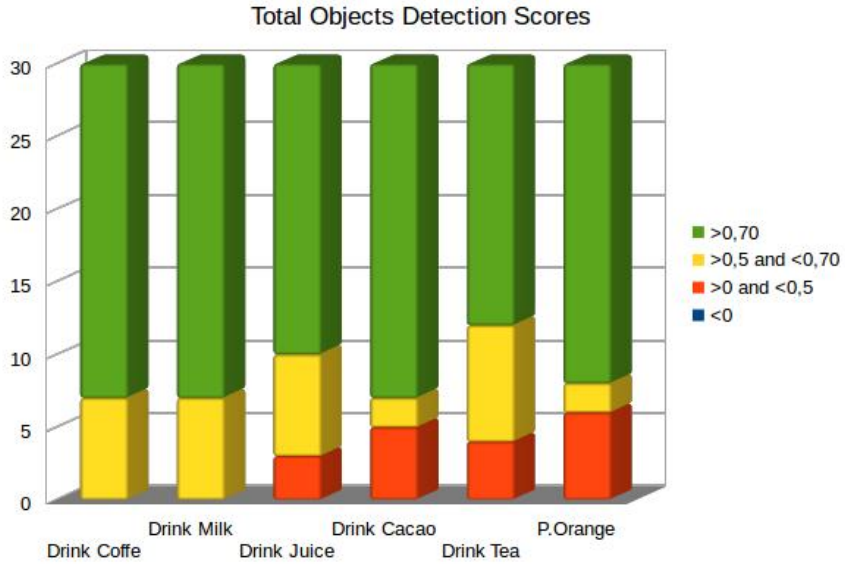


Figure 5.5: *Total Object Detection Scores*

Chapter 6

Conclusions

Finally, with the implementation of this project, we had a minimal viable product "MVP" to recognize different activities that the user is doing. This "MVP" can recognize the activities previously modeled by using the recognition of the objects in the scene and the moment that the user interacts with them.

6.1 Achieved Improvements

This project started by evaluating the results of another project implemented in the past and took the results it obtained into account in its elaboration as an evolution of the first one.

In a resumed way, the improvements we have added to the system with respect of the previous version are the following ones:

- **General Improvements:**
  - Adapt the activity recognition layer into the new environment.
  - Project implemented using the new ROS version: Indigo version.
  - Using a distributed ROS configuration.
  - Segmentation of the code to allow the integration of improvements in parallel.
  - Possibility to use intermediate steps information to use it in other projects in the robot environment.
- **Object Segmentation Process:**
  - Increase the area of the object detection.
  - Remove the user when he appears in the scene, removing the unstable regions.
  - The possibility of using different cameras in the environment at the same time with different characteristics. The system only needs that the camera can publish its information in the ROS environment.
- **Object Recognition Process:**
  - Use more sophisticated algorithms to know the key-points of the objects.
  - Integrate in SVM structure to do the classification process.
  - Add the Supervised Learning Process.
  - Modify the objects database, adding images using the supervised learning mode.
  - Add an initial object tracker solution.
  - Publish a list of objects and information about the state of these objects.
- **Activity Recognition Process:**

- Create a database file to allow the system to modify or add new activities.
- Add the dynamic executions of the activity.
- Add information about the next most probable movements of the user's of the activity recognition.
- Add information about the state of completion of the activity.
- Publish the score of the recognized activity.

6.2 Results Evaluation

6.2.1 Results of the Object Segmentation

The main objective of this part of the process explained in the chapter 2 is the extraction of the objects that are present in the workspace of the activity to use it in next steps of the process. To do this task, the system attacks the problem dividing it in various subproblems that the system can resolve more effectively, taking the strength and weakness of the different methods we studied into account. This division of the problem generates 4 independent masks using two different expressions. One of these expressions is built using previous electronics knowledge to adapt it in the computer vision area. Once that all independent masks are created, the system builds the general mask using this 4 partial mask. To facilitate the process, before the comparisons process, the system applies first a Gaussian filter to reduce the edges of the objects and, then, adapts the pixel representation pondering the values of each channel with the total energy value that the sensor can detect. All these steps minimize the perturbations produced by the changes of the illumination and the quantification problems of the sensors, producing more stable results.

The results of applying this process is the accomplishment of the segmentation of the object, cropping the raw size of the image in little parts where they appear inside the object. One of the advantages of the system is that it doesn't consider the shadows generated in the scene as objects when they appear in the scene, facilitating the following process to compute the SURF key-points. In the appendix D, we can see all the objects in the database and how the system segments the image descanting the shadows. On the other hand, in the main mask appear unstable regions that the system detects wrong as objects. To reduce this problem, the system applies a temporal study to discard these unstable regions considering the information of the centroid and the area. These unstable regions appear and disappear, changing the position and the area values all the time, so by applying this temporal study, we discard all these regions selecting only the region of the objects.

But applying this solutions has a negative effect, the system needs to wait time to prepare all the informations to consider in the temporal study, wasting time in the process. Now, this process is the slowest part of all the project.

On the development process we considered not to generate an individual solution for our environment, but we tried to find a generic solution that allowed the system to adapt itself to different environments and to the various sensors that the system may have if the images are published in ROS.

6.2.2 Results of the Object Classifier

It is the most important and extensive part of the project because we need to develop different steps in the classification of objects to make the process possible.

First, in chapter 3 we evaluated the use of two types of features to be used as descriptor of the objects. On one hand, the use of global descriptor using the histogram of color and, on the other hand, the use of local characteristics of the objects using SURF algorithm. But the result of mixing both features produced errors in the recognition when two objects were similar. After evaluating this behavior and finding the reason of that, we discarded it and only used the SURF algorithm to build the descriptor of the objects.

The results we obtained were the expected ones, the system has good accuracy in the objects that have more unique characteristics. One of the best classification cases is the cut table, that the system can detect using 4 samples in the database. The reason of that is the streak of the wood, because it is an unique characteristic that defines this object. In the other case, the system has more problems to recognize the objects that don't have a characteristic surface and only have one dominant color.

In the classification process the system uses a Support Vector Machine due to its facility in managing the configuration, making it possible to adapt to our necessities. If we see again the results we presented in the table 5.1 the behavior of the classification is situated over 89 % of accuracy. Using the supervised learning mode, the system can add new objects and more samples using the segmented object image in a comfortable method for the users.

But, in the development process we didn't take the possibility that two objects appeared together into account. In this case, in the area segmented appeared both objects in the same region. When the system tried to classify the object images, depending on the influence of both objects, the results change. When the influence of perturbation isn't important, the system recognizes the object, but when there is an important perturbation, the system may recognize it in a wrong way.

6.2.3 Results of the Object Tracker

The main objective explained in chapter 3.5 of the object tracker is to find the moment when the user interacts with the objects to develop a hierarchical descriptor of the activity. It was an initial solution to evaluate the importance of these relationships in the detection of the activities by the system. The object tracker was implemented using the CamShift algorithm using the object information computed in previous steps of the process.

The results we obtained in this first solution is the possibility to detect if the objects are moving or not. Remember that the system can detect the objects in static mode, so it is important to monitor the objects when they don't remain still. Now, when a new object appears in the scene, the system stores the information, but one part of this information was actualized by the object tracker. So, the system can follow all the objects if it remains inside the scene. Now, the main problem we had using the object tracker is when there appear other characteristic points, for example, when one object appears near another, the computation of the features takes both characteristics into account to follow the objects, appearing problems because the solution doesn't consider all the situations that can appear in a real scenario.

The object tracker was implemented in the object recognition code because it uses an important information produced in this part of the process to use it in the tracking task, using the dead time that the process has, waiting the actualization of the object ros topic.

6.2.4 Results of the Activity Classifier

The objective explained in the chapter 4 is the realization of the on-line prediction of the activity in early steps of the activity to bring this information to the main system to use it in other process.

This prediction is computed using the information of the objects inside the scene and how it is used in the execution of the activity to build the different scores that the system has:

- **Static Value:** The system evaluates the objects in the scene and tries to use this information to make the prediction of the activity. These values don't need that the activity is in execution and the system starts to make predictions at the same moment that the system detects the first object in the scene.

- **Dynamic Value:** The system creates a hierarchical definition of the activity and uses it to compare this relationship with the model.

- **Total Value:** Is the value computed using both previous scores with an expression.

The necessity of bringing to the subscriber an early prediction in on-line mode makes the system the necessity to use all the informations that the system knows at the moment to use it in the elaboration of the predictions. Now, to accomplish these specifications, the system starts to compute the predictions at the same time that the first object appears in the scene. Another important issue that the system takes into account is to detect if the activity is in execution or if the user is only preparing the objects to do the activity. Depending on these two status, the method that the system uses to compute the total score changes from using only the static information or also taking the dynamic score into account. Moreover, using an internal information about the activity, the system offers other important data, for example:

- The percentage of the execution of the activity that the system has predicted.
- The status of the activity: waiting, in execution or finished
- Time information of the execution.
- The method that the system uses to compute the total value score.
- The most probable movements that the user will do to accomplish the activity that the system has predicted.

If now we take the results of the classifier of the activity in the point 5.3 into account, the system has accurate results in classifying the different activities. But this accurate values are produced by the static component. If we only consider the static values, we can observe that the system only needs these values to classify the most probable activity that the user is doing. The dynamic information may bring more information in determinate cases to help the static information to predict the action correctly. For example, when inside the scene there are the objects needed to do two or more activities at the same time. In this case, the static process produces similar scores in different activities, avoiding a clear detection of the activity. The system publishes the value of the different scores in a separated way: static, dynamic and total, bringing the possibility to adapt better to the necessities of the final subscribers. They can use the different scores and decide what scores are more important in their applications.

Chapter 7

Future Works

Taking the results in chapter 5 and the conclusion in chapter 6 into account and how is the behavior of the system in the different conditions in which we tried it during all this time, the final results can be improved increasing the quality of the different parts of the project.

The changes proposed in this chapter may facilitate the main task of evaluation of the user's activity, increasing the results and the stability of our system. These improvements are presented in the four main parts in which this project is divided.

7.1 Object Segmentation

Here, in this part of the process, the improvements are focused in two main areas. First, the system can recognize these stable different areas but, in the mask generation process, there appear other unstable regions that the system detects as objects when they don't exist. To improve the results we obtained, we propose to add new layers in the process to take the depth and infrared data into account. The system only needs to add the two processes more, to build each mask and add the results in the main process mask. Taking this part into account, we can build one expression to relate all these layers making possible to discard areas depending on the results of the different channel layers.

Second, another process we need to improve to increase the quality of the system is the velocity of the publication of the objects. The actual process needs to wait a determinate time to evaluate which regions remain stable and which don't. We can increase the results by modifying the method we use to manage the temporal images segmentation vector to increase the publication velocity.

7.2 Object Recognition

To improve this part of the project, we first have to substitute the objects without texture for other textured objects to facilitate the recognition task process.

The second improvement we can propose is the integration of the new key-point extractor algorithm, explained in the chapter 3.1.5. We need to evaluate the integration of this new key-point extractor because this change afects in the actual version of the OpenCV. In theory, the Akaze descriptor increases the efficiency of the system in the computation of the key-points in comparison of SURF. But before the integration, we must evaluate if it really has advantages to use it in this project because it implies that the OpenCV v3.0 must be installed.

If now we center our attention in the classification part, the first change we may do to improve the system is the possibility to store the information extracted of the image database in one file. This change makes it possible that we don't need to store the image database and compute the main characteristics of the objects in each execution of the code, saving this part of the process, so the time that the system needs to use in the configuration process will be less. This change has another important advantage, the free space that the system needs to use to store the information will be reduced, because the system only needs to store one matrix to represent all the image database information.

Another important improvement that we can do in this project is about how to manage the output of our SVM. Now the system tries to classify any object in the most probable object cases taking the information in the database into account, so if we put one untrained object, the system will try to classify it too. To solve this bad behavior of the system, we would use a nested boolean SVM structure. Using an SVM structure for each object, we can discard objects even though they appear together in the scene.

7.3 Object Tracker

The object tracker detector is another part of the project which can improve our results. We can use it for example to detect trajectories of the objects to detect the main area where the action is occurring or if one object is implicated in one activity or in multiple activities recognition. We can also know the main activities of the users, if we know that two objects are in movement together in the same area, the most probable case is that the user is using them both to make an action, so we use it to define actions that are in the activity detections.

If we would like to grow the detection of more complex activities, this module must grow. Now, in the actual version of this project, the object tracker is situated in the same code than the object recognition, but in future versions of the code, it should be situated in an independent block.

7.4 Activity Recognition

In this part of the project we can improve the results, first by adding the two different layers explained in the point 4.2.4. When the system needs to discriminate in more activities, it is possible that we can add new probabilistic parameters that penalize or increase the probability that one activity appears. We can reduce the cases in which the system needs to classify, increasing the best scores of our activity recognition system.

By adding information about fitness bands in other improvements we can implement to know the activity that the user is doing better. But it has not only this application. If we know the fitness activity of the user and the activity that the user is doing, we can evaluate if the activity in the kitchen is healthy for him. We can introduce another type of project to evaluate the users healthy habit to try to improve the quality of the user's life.

7.5 Discussion

During the development of this project, we increased the knowledge of the necessities to implement an activity recognition layer in the robotics system. Even thought the actual version of the activity recognition layer increases the complexity of the previous project, it is still necessary to improve the system to allow the possibility to detect more complex situations. The detection of a multi activity or the user's task when it detects the interaction with two objects at the same time are two examples of these possible improvements. They can be developed if more time is invested in the accuracy of the object tracker, improving the general behavior of the system.

Nowadays, when working with robotics in continuously changing environments, it is difficult to take all the possible cases into account and there will be always situations that can't be considered. Of course, by increasing the complexity of the system, we can have better results in more different cases, but it is important to work taking the past experiences into account to apply them in future cases to increase the trustworthiness of the project using a standard robotic environment.

Chapter 8

Scheduling and Economical Analysis

In this part, we will talk about the scheduling developed and the economical part of the project. In the appendix A we can see the Gantt diagram with the scheduling of the implementation of the project. This project was developed during 6 months, it started in June '15 and finished in November '15. If we see the diagram, the most important part of the integration was implemented in a serial method because we needed results from the previous step to implement them in the next part of the project, so we couldn't reduce the implementation parallelizing task. But this development limitation increased the test time we used to evaluate the results of the different parts of the project. So, this is an advantage because we tested our project in different days and it allowed us to optimize the code to improve the results.

Now we will talk about the economical analysis of the implementation. In the appendix E we can see two tables where there are the most general the development and implementation costs of the project detailed. To develop the project we don't need a very specialized hardware, because we only need one powerful computer and one kinect v2 to develop the solution. If we talk about the software we used to build the solution, we don't have supplement costs because all the software we used is distributed with free license.

Talking about the objective market of this project, we observe that it is oriented to a specialized market, because this project was implemented to work together with other robotic parts. This means that nowadays only big companies or researcher groups have enough money to invest in this type of robotic environment. So, the number of potential clients decreases, but, on the other hand, the client prototype is a specialized client that can invest enough money in these robotic solutions.

Taking the client prototype into account, it is necessary to think if this type of client is good or not for our project or if we need to orientate this project to another type of client.

If now we evaluate the integration of this project in the domestic environment, the prototype client changes because the project is addressed to customers with a high economic level. This change affects in the number of potential clients that are possibly interested in this system. But this change affects directly on the main objectives of the project because we need to adapt the system to offer the final user interesting information of the scene.

Generally, the market in the actuality is working on the integration of different sensors and actuators in the domestic environment, so this project can offer new types of sensors that we will use to integrate new characteristics in this intelligent domotic environments. The handicapped users, blinded people or general users working in the assistance environment are examples of the clients that would be interested in the integration of this project.

What is the main strategy to integrate this project in the user's life? Nowadays, it doesn't exist any system integrated in the domotic environments to being used to capture the user's activities information and use it for a specific task. For this reason, we suppose that an effort in investment in a good marketing study needs to be done to make it possible that the final users know our system. So, this situation produces the necessity to catch the attention of the final users and to create in them the necessity to obtain this new system.

If we see the picture 8.1, we can see the classical product life cycle and the different steps of the process. So, the marketing road route needs to take this graph into account to adapt the different strategy markets to take always advantage of the market situation.

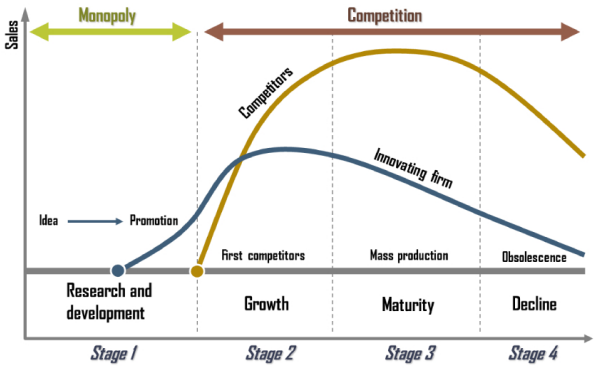


Figure 8.1: *Classical Product Life Cycle*

The different marketing strategies we would apply to return the investment would be the following ones:

1. **The product enters in the market:** A priori, we don't have many competitors, but we also don't have a real demand. The possible clients we would be interested on are specific clients with a high economical capacity. The final cost of our product can remain high because the type of clients runs. In this part of the process, we don't have a big costumer base, so we must be interested in making this costumer base grow. The easiest way to do it is by using our clients like a marketing stuff, offering discounts if another client buys another system. This discount would be the 50% of the development costs.

2. **The product generates expectation:** In this point, our costumer base increases but the product generates expectation. In this state point of the life cycle, it is possible that the competitors work to develop their solutions. So we need to take a decision to become more aggressive in the marketing campaign, because we need to decrease the cost of the product to increase its demand. The discount we can offer would be a 25 % of the development costs for our clients to get new clients, and a 25 % of discount for the new costumers.

3. **The product is consolidated in the market:** In this point we can reduce the development costs in the price of the final user because the return of the investment is about to be produced. If we reduce the price, more customers will be interested in the system and the benefits of our project will increase.

4. **Following steps:** Using the 20 % of benefits of this project and the 50 €/Year (development tax) for each client, we have the possibility to reinvest this money to improve the system by adding more characteristics, increasing the quality by using the previous experience of our costumers.

Appendix A

Schedule of the Project

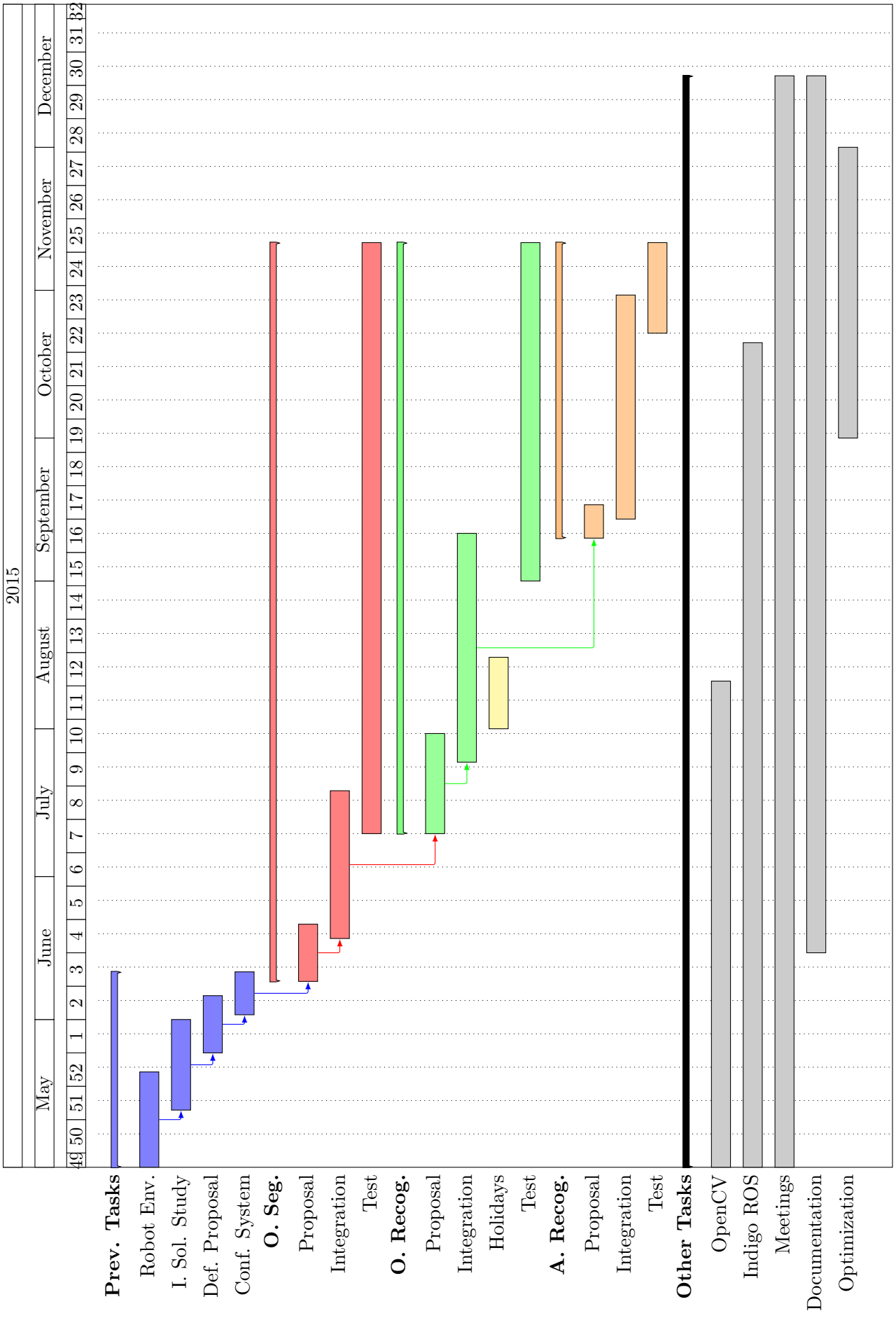


Figure A.1: *Project Planning Schedule*

Appendix B

General Diagram

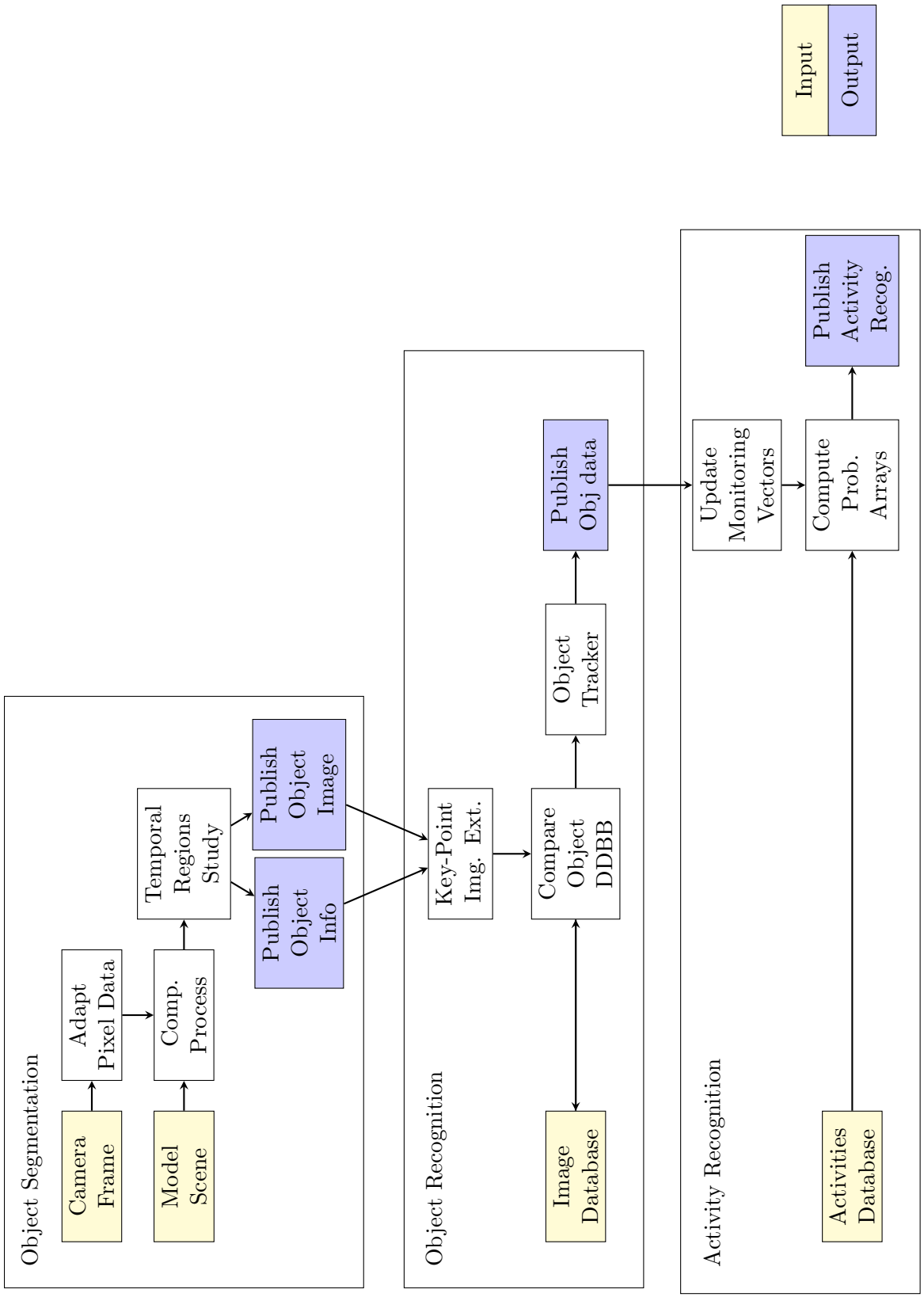


Figure B.1: *General Diagram of the Act. Recognition Project*

Appendix C

Code Section

C.1 Object Segmentation Codes

C.1.1 Mixture of Gaussian Object

```
// The Mixture of Gaussian object used with all default parameters
cv::BackgroundSubtractorMOG mog;
// update the background and return the foreground
mog(frame,foreground,0.01);
// Complement the image
cv::threshold(foreground,foreground,128,255,cv::THRESH_BINARY_INV);
```

OpenCV allows to configure the MOG implementation adding commands to control it. It allows to control, for example, the number of Gaussian mixtures used by the system or if the user would like to capture the shadows of the objects or not.

```
//create Background Subtractor objects
cv::BackgroundSubtractorMOG2 bg;
bg.nmixtures = 3; // set number of gaussian mixtures
bg.bShadowDetection = false; // turn the shadow detection off
bg.operator ()(frame,fore_ground);
bg.getBackgroundImage(back_ground);
```

C.1.2 Image Callback

```
//--------------------------------imageCallback------------------------------------------
void imageCallback (const sensor_msgs::ImageConstPtr& msg)
{
  try
  {
    frame =(cv_bridge::toCvShare(msg, "bgr8")->image);
    frame.convertTo(frame,CV_32F, 1.0/255.0);
    frame_flag=true;
  }
  catch (cv_bridge::Exception& e)
  {
    ROS_ERROR("Could not convert from '%s' to 'BGR8'.", msg->encoding.c_str());
  }
}
```

C.1.3 Infrared Callback

```
//--------------------------------Ir_Callback------------------------------------------
void ir_image(const sensor_msgs::ImageConstPtr& msg)
{
   try
   {
      ir_frame =(cv_bridge::toCvShare(msg, "16UC1")->image);
```

```
        double min, max;
        minMaxIdx(ir_frame, &min, &max);
        ir_frame.convertTo(ir_frame,CV_32F, 1.0/max);
        equalizeHist( ir_frame, ir_frame );
        ir_frame_flag=true;
    }
    catch (cv_bridge::Exception& e)
    {
       ROS_ERROR("Could not convert from '%s' to '16UC1'.", msg->encoding.c_str());
    }

}
```

C.1.4 Comparative Process

```
Image_b=Image_Energy_b/(Image_Energy_b+background_b);
Image_b=(abs(Image_b-0.5))/0.5;
val_max=mean(Image_b)[0]+0.05;
Image_b=(Image_b > val_max);

Image_g=Image_Energy_g/(Image_Energy_g+background_g);
Image_g=(abs(Image_g-0.5))/0.5;
val_max=mean(Image_g)[0]+0.05;
Image_g=(Image_g > val_max);

Image_r=Image_Energy_r/(Image_Energy_r+background_r);
Image_r=(abs(Image_r-0.5))/0.5;
val_max=mean(Image_r)[0]+0.05;
Image_r=(Image_r > val_max);

absdiff(Image_Energy,background_low, Image_low);
threshold(Image_low,Image_low,0.8,1,0);
Image_low.convertTo(Image_low,CV_8U, 255);
```

C.1.5 Study of Regions

```
int comparator;

  if (count_mc_previus < 5)
  {
    for(int i =0; i<contours.size();i++)
    {
      if (contourArea(contours[i])>800)
      {
        mc_ant.push_back(mc[i]);
      }

    }
    count_mc_previus++;
  }
    else
    {
      for(int i =0; i<contours.size();i++)
      {
        mc_actual=mc[i];
        comparator=count_if (mc_ant.begin(), mc_ant.end(), comparation_funtion);
        Mat drawing = Mat::zeros( canny_output.size(), CV_8UC3 );
        if (comparator > 9)
        {
          Mat img_temp = Mat::zeros( canny_output.size(), CV_8UC3 );
          ROS_INFO("Plot new object detection");
          drawContours( drawing, contours, i, Scalar( 255, 255, 255 ), 2, 0,
  hierarchy,0, Point() );
          cv::floodFill(drawing, mc_actual, cv::Scalar(255.0, 255.0, 255.0));
```

```
28            circle( drawing, mc[i],4, Scalar( 255, 0, 0 ),-1, 8, 0);
29            drawing.convertTo(drawing,CV_32F, 1.0/255);
30            multiply(frame_w_out_blur,drawing,img_temp);
31            imshow("frame",frame_w_out_blur);
32
33            //Detect the keypoint of the object and the features thats define it.
34          }
35
36        }
37        count_mc_previus=0;
38        comparator=0;
39        mc_ant.clear();
40      }
```

C.2 Object Recognition Codes

C.2.1 Color Histogram Code

```
1  //--------------------------------color histogram descriptor-------------------------------
2  void color_hist_descriptor(cv::Mat img)
3  {
4     img.convertTo(img,CV_8U, 255);
5
6     /// Separate the image in 3 places ( B, G and R )
7     vector<Mat> bgr_planes;
8     split( img, bgr_planes );
9
10    /// Establish the number of bins
11    int histSize = 10;
12
13    /// Compute the histograms:
14    calcHist( &bgr_planes[0], 1, 0, Mat(), b_hist, 1, &histSize, &histRange, uniform,
      accumulate );
15    calcHist( &bgr_planes[1], 1, 0, Mat(), g_hist, 1, &histSize, &histRange, uniform,
      accumulate );
16    calcHist( &bgr_planes[2], 1, 0, Mat(), r_hist, 1, &histSize, &histRange, uniform,
      accumulate );
17
18    /// Normalize the result to [ 0, histImage.rows ]
19    normalize(b_hist, b_hist, 0, histImage.rows, NORM_MINMAX, -1, Mat() );
20    normalize(g_hist, g_hist, 0, histImage.rows, NORM_MINMAX, -1, Mat() );
21    normalize(r_hist, r_hist, 0, histImage.rows, NORM_MINMAX, -1, Mat() );
22
23    for(int j = 0; j < b_hist.rows; j++) hist.push_back(b_hist.at<float>(j, 0));
24    for(int j = 0; j < g_hist.rows; j++) hist.push_back(g_hist.at<float>(j, 0));
25    for(int j = 0; j < r_hist.rows; j++) hist.push_back(r_hist.at<float>(j, 0));
26
27 }
```

C.2.2 SIFT Code

```
1  //--------------------------------sift_descriptor-------------------------------------------
2  void sift_descriptor(cv::Mat img)
3  {
4     img.convertTo(img,CV_8U, 255.0);
5     cv::SiftFeatureDetector detector;
6     std::vector<cv::KeyPoint> keypoints_sift;
7     detector.detect(img, keypoints_sift);
8  }
```

C.2.3 SURF Codes

```
//--------------------------------surf descriptor-------------------------------------
void surf_descriptor(cv::Mat img)
{
   img.convertTo(img,CV_8U, 255.0);
   //-- Step 1: Detect the keypoints using SURF Detector
   int minHessian = 400;
   SurfFeatureDetector detector(minHessian);
   std::vector<KeyPoint> keypoints_object;
   detector.detect( img, keypoints_object);
   if (keypoints_object.size() > 39)
   {
      RetainBestKeypoints(keypoints_object,40);
      for (int i=0; i<keypoints_object.size(); i++) ROS_INFO_STREAM(keypoints_object
 [i].size);

      // Construction of the SURF descriptor extractor
      cv::SurfDescriptorExtractor surfDesc;
      // Extraction of the SURF descriptors
      cv::Mat descriptors;
      surfDesc.compute(img,keypoints_object,descriptors);
   }
}
```

```
//--------------------------------strongest M keypoints-------------------------------
bool compareFunction(KeyPoint p1, KeyPoint p2) {return p1.response>p2.response;}
void RetainBestKeypoints(vector<KeyPoint> &kp, int M)
{
   vector<KeyPoint> sortedkp;
   sort(kp.begin(),kp.end(),compareFunction);
   if (kp.size()>M)
   kp.erase(kp.begin()+M,kp.end());
}
```

C.2.4 AKAZE Code

```
//--------------------------------akaze_descriptor-------------------------------------
void akaze_descriptor(cv::Mat img)
{
   img.convertTo(img,CV_8U, 255.0);
   vector<KeyPoint> keypoints_akaze;
   Mat desc1;
   AKAZE akaze;
   akaze(img, noArray(), keypoints_akaze, desc1);
}
```

C.2.5 Selector Menu

```
//--------------------------------Select_option-----------------------------------------
void Select_option(int sel_opt)
{
  switch(sel_opt)
  {
    case 1: ROS_INFO("-------------------------------");
            ROS_INFO("| Load Inicial Data from file |");
            ROS_INFO("-------------------------------");
            load_svm_params();
            break;

    case 2: ROS_INFO("-------------------------------");
            ROS_INFO("| Add element to Data Base |");
            ROS_INFO("-------------------------------");
            Add_element_database();
```

```
            ROS_INFO("Add new element to database? (Y/N):");
            getline(cin, str);
            if ((str == "n") || (str == "N")) select_option=3;
            break;

    case 3: ROS_INFO("-----------------------------------------");
            ROS_INFO("| Compute the Features of all Database |");
            ROS_INFO("-----------------------------------------");
            compute_data_base();
            break;

    case 4: ROS_INFO("-------------------------");
            ROS_INFO("| Working scene mode |");
            ROS_INFO("-------------------------");
            working_mode();
            break;
  }

//Reset the main flags
frame_flag=false;
info_flag=false;
}
```

C.2.6 Add Object to Database

```
//---------------------------------Add_element_database-----------------------------------
void Add_element_database()
{
 ROS_INFO("This region belongs to the object to be introduced into the Data Base? y/n
  :");
 imshow("output",frame);
 getline(cin, str);

 if ((str == "y") || (str == "Y"))
 {
    ROS_INFO("Save new image in data Base to use it in the training dataset:");
    frame.convertTo(frame,CV_8U, 255,0);
    ROS_INFO("introduce the label of the object(1,2,3,...):");
    getline(cin, str);
    cnt_samples=atoi(str.c_str());
    ROS_INFO("Is a new object?");
    getline(cin, str);
    if ((str == "y") || (str == "Y")) cnt_image=0;

 sprintf(ch,"%s%s%d%s%d%s","/home/eajenjo/PFM_ws/objects_data_base/","Object",
  cnt_samples,"(",cnt_image,").jpg");
 imwrite(ch,frame, compression_params); //write the image to file
 cnt_image++;
 }
 else ROS_INFO("Descart image to add at Data Base");

}
```

C.2.7 Compute the Descriptors of the Image Database Code

```
//---------------------------------compute_data_base();-----------------------------------
void compute_data_base()
{
 for(int j=1;j<=cnt_samples;j++)
 for(int i=1;i<=cnt_image;i++)
 {
    sprintf(ch,"%s%d%s%d%s","/home/eajenjo/PFM_ws/objects_data_base/Object",j,"(",i,"
  ).jpg");
    const char* imageName = ch;
```

```
    Mat img = cvLoadImage(imageName);
    vector<KeyPoint> keypoint;
    detector.detect(img, keypoint);
    Mat features;
    extractor->compute(img, keypoint, features);
    bowTrainer.add(features);
}
vector<Mat> descriptors = bowTrainer.getDescriptors();
Mat dictionary=bowTrainer.cluster();
bowDE.setVocabulary(dictionary);

//Extracting histograms in the form of BOW for each image

Mat labels(0, 1, CV_32FC1);
Mat trainingData(0, dictionarySize, CV_32FC1);
int k=0;
vector<KeyPoint> keypoint1;
Mat bowDescriptor1;

for(int j=1;j<=cnt_samples;j++)
for(int i=1;i<=cnt_image;i++)
{
    sprintf(ch,"%s%d%s%d%s","/home/eajenjo/PFM_ws/objects_data_base/Object",j,"(",i,"
).jpg");
    Mat img2 = cvLoadImage(ch);
    detector.detect(img2, keypoint1);
    bowDE.compute(img2, keypoint1, bowDescriptor1);
    trainingData.push_back(bowDescriptor1);
    labels.push_back((float) j);
}

}
```

C.2.8 Working Mode Code

```
void working_mode()
{
    cout<<"Processing evaluation data..."<<endl;
    vector<KeyPoint> keypoint2;
    Mat bowDescriptor2;
    frame.convertTo(frame,CV_8U, 255);
    imshow("frame",frame);
    detector.detect(frame, keypoint2);
    RetainBestKeypoints(keypoint2,30);
    bowDE.compute(frame, keypoint2, bowDescriptor2);
    float response = svm.predict(bowDescriptor2);
}
```

C.2.9 Object Tracker Code

```
    for(int i=0;i<objects_detected_list.size();i++)
    {
      if(objects_detected_list[i].status_object_state !=2)
      {
        if(objects_detected_list[i].histogram.empty())
        {
          selection.x =objects_detected_list[i].x-(objects_detected_list[i].
image_size.y/2);
          selection.y =objects_detected_list[i].y - (objects_detected_list[i].
image_size.x/2);
          selection.width = objects_detected_list[i].image_size.y;
          selection.height = objects_detected_list[i].image_size.x;


```

```
13          if((0 <= selection.x && 0 <= selection.width && selection.x + selection.
    width <= hue.cols && 0 <= selection.y && 0 <= selection.height && selection.y +
    selection.height <= hue.rows)==1)
14          {
15            Mat roi(hue,selection),maskroi(mask,selection);
16            calcHist(&roi, 1, 0, maskroi, objects_detected_list[i].histogram, 1, &
    hsize, &phranges);
17            normalize(objects_detected_list[i].histogram, objects_detected_list[i].
    histogram, 0, 255, NORM_MINMAX);
18
19            Mat backproj;
20            calcBackProject(&hue, 1, 0, objects_detected_list[i].histogram, backproj,
     &phranges);
21            backproj &= mask;
22            objects_detected_list[i].init_trackBox = CamShift(backproj, selection,
23            TermCriteria( TermCriteria::EPS | TermCriteria::COUNT, 10, 1 ));
24            objects_detected_list[i].trackBox=objects_detected_list[i].init_trackBox;
25            objects_detected_list[i].init_histogram=objects_detected_list[i].
    histogram;
26
27            ellipse(scene,objects_detected_list[i].trackBox, objects_detected_list[i
    ].color, 0.1, CV_AA );
28            cv::imshow("output", scene);
29          }
30        }
31        else
32        {
33          selection.x = objects_detected_list[i].trackBox.center.x - (
    objects_detected_list[i].trackBox.size.width/2);
34          selection.y = objects_detected_list[i].trackBox.center.y - (
    objects_detected_list[i].trackBox.size.height/2);
35          selection.width = objects_detected_list[i].trackBox.size.width;
36          selection.height = objects_detected_list[i].trackBox.size.height;
37
38          if((0 <= selection.x && 0 <= selection.width && selection.x + selection.
    width <= hue.cols && 0 <= selection.y && 0 <= selection.height && selection.y +
    selection.height <= hue.rows)==1)
39          {
40            Mat backproj;
41            calcBackProject(&hue, 1, 0, objects_detected_list[i].init_histogram,
    backproj, &phranges);
42            backproj &= mask;
43            RotatedRect trackBox= CamShift(backproj, selection,TermCriteria(
    TermCriteria::EPS | TermCriteria::COUNT, 10, 1 ));
44
45
46
47            if (trackBox.center!=Point2f(0,0))
48            {
49              objects_detected_list[i].trackBox=trackBox;
50              objects_detected_list[i].trackBox.size=objects_detected_list[i].
    init_trackBox.size;
51              ellipse(scene,objects_detected_list[i].trackBox, objects_detected_list[
    i].color, 0.1, CV_AA );
52              cv::imshow("output", scene);
53
54              objects_detected_list[i].distance=objects_detected_list[i].trackBox.
    center - objects_detected_list[i].init_trackBox.center;
55              objects_detected_list[i].distance.x=abs(objects_detected_list[i].
    distance.x);
56              objects_detected_list[i].distance.y=abs(objects_detected_list[i].
    distance.y);
57              objects_detected_list[i].x=selection.x;
```

```
58                objects_detected_list[i].y=selection.y;
59              }
60
61              if((objects_detected_list[i].distance.x >50 || objects_detected_list[i].
    distance.y >50) && objects_detected_list[i].status_object_state<2)
62              {
63                objects_detected_list[i].status_object_state=1 ;
64                if (objects_detected_list[i].Start_movement_Time == 0) time(&
    objects_detected_list[i].Start_movement_Time);
65                else
66                {
67                  time(&objects_detected_list[i].currentTime);
68                  objects_detected_list[i].Total_used_time=objects_detected_list[i].
    currentTime - objects_detected_list[i].Start_movement_Time;
69                }
70              }
71              if(objects_detected_list[i].status_object_state==1 && trackBox.center==
    Point2f(0,0))
72              {
73                objects_detected_list[i].status_object_state=2;
74              }
75
76
77
78            }
79
80          }
81        }
82      }
83 }
```

C.3 Activity Recognition Codes

C.3.1 Object Callback Code

```
1//-----------------------------------Object_Callback-------------------------------------------
2void Object_Callback (const project_package::Object& msg)
3{
4 try
5 {
6  Object_list=msg;
7  Activity_msg.header=Object_list.header;
8  flag_Object_list=true;
9 }
10 catch (cv_bridge::Exception& e)
11 {
12   ROS_ERROR("Can't obtain the topic message");
13 }
14}
```

C.3.2 Mode Selector Menu

```
1
2//-----------------------------------Mode_selector---------------------------------------------
3void mode_selector(int sel_opt)
4{
5  switch(sel_opt)
6  {
7    case 1 :  ROS_INFO("-------------------------------------");
8              ROS_INFO("| Load Inicial Data from files |");
9              ROS_INFO("-------------------------------------");
10             load_init_params();
11             break;
```

```
12
13      case 2 :  ROS_INFO("-----------------------------------");
14                ROS_INFO("| Add new Activity                |");
15                ROS_INFO("-----------------------------------");
16                Add_activity_database();
17                break;
18
19      case 3 :  ROS_INFO("-----------------------------------");
20                ROS_INFO("| Activity detection mode |");
21                ROS_INFO("-----------------------------------");
22                activity_values.Dynamic_rates.release();
23                activity_values.Static_rates.release();
24                working_mode();
25                working_dynamic_mode();
26                publish_info_mode();
27                break;
28    }
29}
```

Appendix D

Object Element Database

(a) *Cut Table* **(b)** *Colacao* **(c)** *Glass* **(d)** *Orange*

(e) *Big Fork* **(f)** *Coffee* **(g)** *Big Spoon* **(h)** *Sugar*

(i) *Juice* **(j)** *Milk* **(k)** *Spoon* **(l)** *Tea case*

(m) *Pan* **(n)** *Dish* **(o)** *Brush* **(p)** *Heat Shield*

Figure D.1: *Object Data Base*

Appendix E

Economical Analysis

Table E.1: *Economical Development Costs*

Product	Quatity	Price	Repayment	Month Dev.	Cost
Developer PC Dev	1	1200	24	6	300
Kinect Camera	1	200	12	6	100
ROS	1	0	0	6	0
OpenCV	1	0	0	6	0
Developer	1	35000	12	8	23333

Dev. Costs	23733 €
20% Benefit	4747 €
Total	28480 €

Table E.2: *Market Value*

Product	Quatity	Unitary Price	Price 100u	Price 1000u
NVIDIA Jetson TK1	1	204	18360	173400
IP Camera wireless	1	69,83	6284,7	59355,5
ROS	1	0	0	0
OpenCV	1	0	0	0
HDD storage Dev	1	40	3600	34000
Dev. Annual Tax	1	50	50	50

Product Cost	363,83 €	282 €	266 €
Dev. Cost	23733 €	237 €	24 €
Dev. Cost +20%	28480 €	284,8 €	28,48 €
Total price + dev.	24097 €	520 €	291 €
Total price + dev. +20%	28843 €	567 €	295 €

Bibliography

[1] Robert Laganière, *OpenCV 2 Computer Vision Application Programming Cookbook*

[2] Release 2.4.9.0, *The OpenCV Reference Manual*

[3] Marcin Blachnik,Jorma Laaksonen, *Image Classification by Histogram Features Created with Learning Vector Quantization*

[4] D.Lowe, *Distinctive Image Features from Scale-Invariant Keypoints*

[5] Herbert Bay,Tinne Tuytelaars,Luc Van Gool, *SURF: Speeded Up Robust Features*

[6] Pablo F. Alcantarilla, Adrien Bartoli, and Andrew J. Davison, *KAZE Features*

[7] Pablo F. Alcantarilla,Jesus Nuevo,Adrien Bartoli, *Fast Explicit Diffusion for Accelerated Features in Nonlinear Scale Spaces*

[8] Kristen Grauman,Bastian Leibe, *Visual Object Recognition*

[9] Gabriella Csurka, Christopher R. Dance, Lixin Fan, Jutta Willamowski, Cédric Bray, *Visual Categorization with Bags of Keypoints*

[10] Stephen O'hara, Bruce A. Draper, *Introduction to the bag of features paradigm for image classification and retrieval*

[11] Gary Bradski, Adrian Kaehler, *Learning OpenCV Computer Vision with the OpenCV Library*

[12] Wikipedia, *Activities of Daily Living*

[13] Mohammad Taghi Saffar *, Mircea Nicolescu, Monica Nicolescu and Banafsheh Rekabdar, *Intent Understanding Using an Activation Spreading Architecture*

[14] Jinna Lei,Xiaofeng Ren,Dieter Fox, *Fine-Grained Kitchen Activity Recognition using RGB-D*

[15] Emmanuel Munguia Tapia, Stephen S. Intille, Kent Larson, *Activity Recognition in the Home Using Simple and Ubiquitous Sensors*

[16] Gita Sukthankar, Christopher Geib, Hung Bui, David Pynadath and Robert P. Goldman, *Plan, Activity, and Intent Recognition*