## Detailed Design and Implementation

In this chapter we will present the reader with all the information necessary to understand the work done on the application, so it can be extended and improved in a future development. We will start by describing the reason behind the chosen language, tools, the top architecture, and then we will talk about the design of the application going into the most important details of the implementation. We will try to highlight the packages that our application in structured into, try to describe what functionality do packages organize and try to reconstruct the structure with class diagrams.

After this overall view of the entire project, we will start to explain the actual implementation in terms of source code. We will then try to present the implementation in a straight-forward way, such that anyone with basic understanding of the programming language in which it was written (C++, and small Ruby scripts) should be able to take a look over it and understand and maybe extend its functionality.

This section will also contain the code of the most important methods used along with their explanation.

## 5.1 Language, Technology, Framework

*C++ ox-11*. Our tool was created using C++ programming language using the new standard adopted last year in August. Areas of the core language that were significantly improved over the last version include multithreading support, generic programming support, uniform initialization, and performance enhancements, as stated in [21].

One should take a look at the improvements and new standards adopted for this version, and which the Visual Studio 2010 compiler include before their standardization, because we will use some of them, and these features are:

* New struct initializers
* Lambda expressions
* Type inference
* Foreach iterations over containers
* Smart pointers(unique\_ptr, shared\_ptr) which act as reference counters and which make memory management similar to garbage collected based languages

*OpenCv*. As pointed out in chapter 2 this is the main framework that we rely on to do the heavy lifting in terms of performing well established algorithms. We will rely on constructs that provide processing in the domain of:

* Background subtraction
* Classification
* Detection
* Lucas-Kanade Optical Flow
* Kalman Filter
* Morphological transforms

*Parallel Patterns Library from Microsoft (PPL)*. According to [22] the Parallel Patterns Library provides an imperative programming model that promotes scalability and ease-of-use for developing concurrent applications. The PPL builds on the scheduling and resource management components of the Concurrency Runtime in Windows. It raises the level of abstraction between the application code and the underlying threading mechanism by providing generic, type-safe algorithms and containers that act on data in parallel. The PPL also lets you develop applications that scale by providing alternatives to shared state.

The PPL provides the following features:

* *Task Parallelism*: a mechanism to execute several work items (tasks) in parallel
* *Parallel algorithms*: generic algorithms that act on collections of data in parallel
* *Parallel containers and objects*: generic container types that provide safe concurrent access to their elements

*SvmLight*. This is a small executable downloaded from [23] offering implementation of Support Vector Machines (SVMs) in C.

*Ruby language*. Ruby is a dynamic, reflective, general-purpose object-oriented scripting programming language. We will mainly use it in small scripts for file processing, particularly for interpreting the output from SvmLight to feed it back to the OpenCV Hog classifier and detector.

## 5.2 System packages

In this section we will try to show the current design and implementation pointing out the modularity of the architecture and the contracts that it provides for the plugability of other modules that implement equivalent algorithms. The next figure illustrates the packages that compose the entire solution.

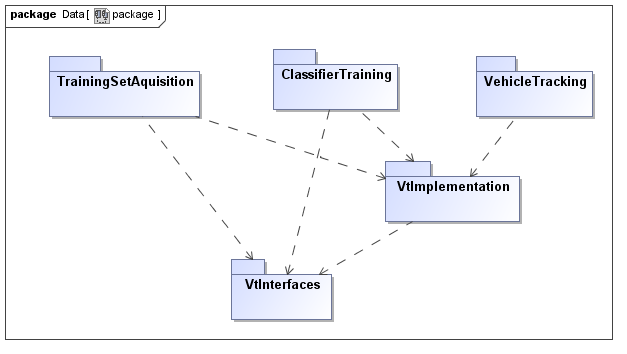


Figure 5… System package diagram

The packages and the main concerns are depicted in the following table.

|  |  |
| --- | --- |
| Packages | Concerns and Responsibilities |
| * VtInterfaces | * Provides the interfaces and contracts that modules must implement in order to be plugged in the system * Provides necessary data structures of input and output for the interfaces, which implementations must use to transfer data * Also provides basic tools for basic geometrical computations |
| * VtImplementations | * Provides interfaces implementations with the algorithms that we described * Also provides tools for processing member data structures that they auxiliary define |
| * TrainingSetAquisition | * Provides an architecture that is used for processing the video to collect the dataset * Uses the interfaces from the VtInterfaces package to define the architecture and instantiates it with specific implementations from VtImplementation |
| * ClassifierTraining | * Organizes the functionality to process the training set and extract training features * Performs classifier training * Collects output to feed it back into the system |
| * VehicleTracking | * Provides the architecture of the tracking functionality * Integrates the modules of classification, detection and tracking in terms of interfaces * Offers the possibility of configurability with new algorithms by instantiating the interfaces to modules that conform to these interfaces |