ARAM User Guide

Version	1.0	Application date	23.04.2014
Code	REA-PUB-MOP-0001	Revision date	29.04.2015
Applied to		ARAM 1.2 beta	

	Written by	Approved by
Name	Alexandre KORNMANN	Abdelkrim BELHAOUA
Function	C++ Developer	Scientist coordinator

Document description
This document provides an installation and user guide of the library ARAM (Augmented Reality for Application on Medical field library).

Summary

1. ARAM User Guide

I. Prerequisite

a. CMake

CMake is a cross-platform free software program for managing the build process of software using a compiler-independent method. It is designed to support directory hierarchies and applications that depend on multiple libraries, and for use in conjunction with native build environments such as make, Apple's Xcode, and Microsoft Visual Studio.

For ARAM library, **CMake >=2.8** is required.

b. OpenCV

OpenCV (Open Source Computer Vision) is a library of programming functions mainly aimed at real-time computer vision.

For ARAM library, **OpenCV >=2.4.8** is required.

II. Setting up

This guide was tested on Windows 7 Enterprise SP1 32bits and 64bits.

In this document, the default workspace is the directory **C:\ARAM\.**
In this directory, we find (see ARAM WorkspaceFigure 1):

- **include**: a subdirectory for ARAM headers files (**.hpp**)
- **src** : a subdirectory for sources files (**.cpp**)
- **build** : an empty subdirectory, for builed files (**.sln**, **.obj**, …)
- **CMakeLists.txt** : CMake configuration file, which help us to generate project file like **makefile** (gcc, clang, …) or **Visual studio solution** (Visual studio).
-

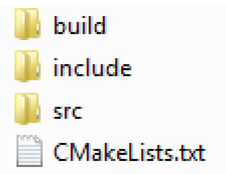


Figure 1 ARAM Workspace

First, we generate our project file (Visual Studio solution, **.sln**), using **CMake GUI (**Figure 2**).**

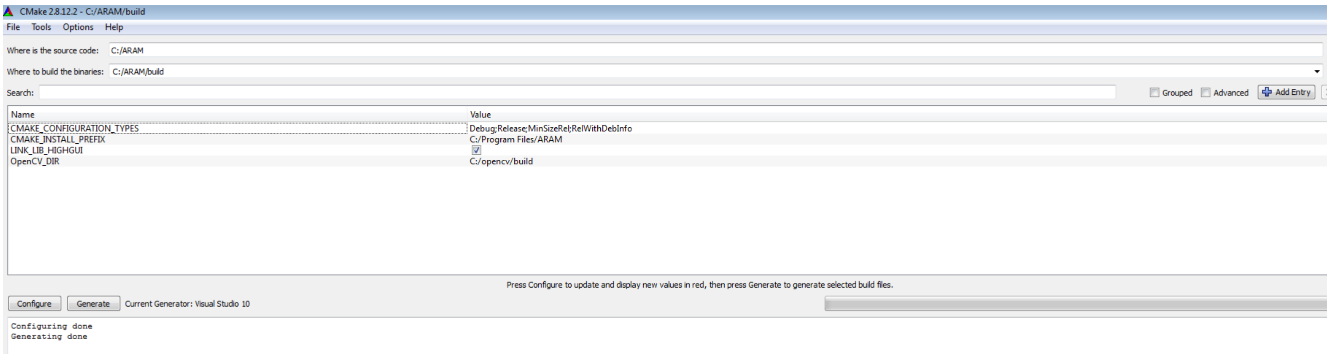


Figure 2 CMake GUI configuration

If CMake fails in OpenCV linking, you have to indicate where OpenCV is in OpenCV_DIR variable.

After click on **Generate**, your directory should contain many files. Open **ARAM.sln**, and start the program!

III. First application

In this section, our goal is to develop a simple application using ARAM library (Figure 3).

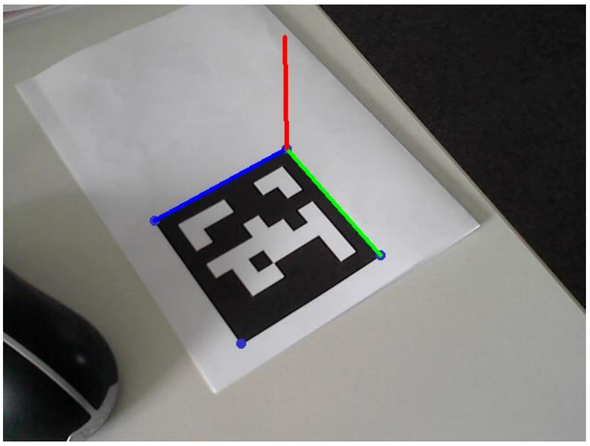


Figure 3 First application output

c. CMakeLists.txt

First, build your workspace for your application (as previously, **src**, **include**, **build**, and **CMakeLists.txt**.)
Typical, **CMakeLists.txt** looks as following:

```
cmake_minimum_required(VERSION 2.8)

project(ARAMProject)
set(EXECUTABLE_OUTPUT_PATH bin)

# ARAM include
include_directories("C:/ARAM/include")
link_directories("C:/ARAM/build/lib/Debug")

# OpenCV include
find_package(OpenCV REQUIRED)
include_directories(${OpenCV_INCLUDE_DIRS})

# Sources files
file(GLOB_RECURSE source_files src/* include/*)

add_executable(${CMAKE_PROJECT_NAME} ${source_files})


# linker
target_link_libraries(${CMAKE_PROJECT_NAME} ARAM)
target_link_libraries(${CMAKE_PROJECT_NAME} ${OpenCV_LIBS})
```

d. *Tags detection*

This is a typical structure for ARAM application:

```
#include <ARAM/TagDetector.hpp> // Main ARAM class
#include <ARAM/tag/StandardTag.hpp> // Tag validator
#include <ARAM/ROIDetector/EdgeDetector.hpp> // Region of interest detection

#include <opencv2/opencv.hpp> // OpenCV data structure

#include <exception> // std::exception

int main(int argc, char** argv)
{
	try
	{
		// Detection parameters:
		// -> Region of interest detection
		// -> Tag validator
		typedef aram::TagDetector<aram::EdgeDetector,aram::StandardTag> myDetector;

		// Tag detector instantiation
		myDetector *detector = new myDetector();

		// Intrinsic parameters
		aram::Intrinsics intr("C:\\camera_data.xml");

		// Video input (see openCV doc)
		cv::VideoCapture cap(0); // use default video
		if(!cap.isOpened()) throw std::exception();

		cv::Mat frame;

		// Main loop
		while(true)
		{
			// next frame from video input
			cap >> frame;

			// Tag detection
			detector->detect(frame);

			// Tag list iterator
			aram::iteratorTag it;
			// Loop over valid tag in current frame
			for(it=detector->begin();it!=detector->end();++it)
			{
				// Some operations here!
			}

			// render
			cv::imshow("render", frame);
			// GUI refresh (see openCV doc)
			if(cv::waitKey(10)>=0) break;
		}
	}
```

```
	catch(std::exception &)
	{
	}

	return 0;
}
```

This program allows us to launch the camera.

Now, we add some operations using detected tag. First, we circle tags corners.
In this loop:

```
// Loop over valid tag in current frame
for(it=detector->begin();it!=detector->end();++it)
{
	// Some operations here!
}
```

As an example:

```
aram::vecPoint2D imgPoint = (*it)->corners();
for(unsigned int i=0;i<imgPoint.size();++i)
{
	cv::circle(frame,imgPoint[i],3,cv::Scalar(200,50,50),3);
}
```

Now, your output looks as following:

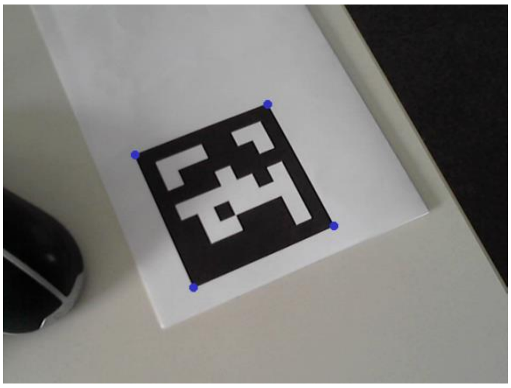


Figure 4 First application output (step 1)

e. Pose estimation

For augmented reality, we need to compute the camera pose estimation. You can perform this with ARAM.

First, you have to calibrate your camera using OpenCV procedure. This procedure produces an **.xml** file, as the following:
I stored my **.xml** file at **C:\camera_data.xml**

```
<?xml version="1.0"?>
<opencv_storage>
<calibration_Time>"03/12/14 13:56:10"</calibration_Time>
<nrOfFrames>5</nrOfFrames>
<image_Width>1280</image_Width>
<image_Height>720</image_Height>
<board_Width>13</board_Width>
<board_Height>13</board_Height>
<square_Size>3.</square_Size>
<FixAspectRatio>1.</FixAspectRatio>
<!-- flags:  +fix_aspectRatio +fix_principal_point +zero_tangent_dist -->
<flagValue>14</flagValue>
<Camera_Matrix type_id="opencv-matrix">
  <rows>3</rows>
  <cols>3</cols>
  <dt>d</dt>
  <data>
    1.2945600566643177e+003 0. 6.3950000000000000e+002 0.
    1.2945600566643177e+003 3.5950000000000000e+002 0. 0. 1.</data></Camera_Matrix>
<Distortion_Coefficients type_id="opencv-matrix">
  <rows>5</rows>
  <cols>1</cols>
  <dt>d</dt>
  <data>
    2.0026392183907174e-001 -2.7114152188654712e+000 0. 0.
    8.1223710401493587e+000</data></Distortion_Coefficients>
<Avg_Reprojection_Error>2.2073910408958071e+000</Avg_Reprojection_Error>
<Per_View_Reprojection_Errors type_id="opencv-matrix">
  <rows>5</rows>
  <cols>1</cols>
  <dt>f</dt>
  <data>
    1.11958861e+000 2.25522900e+000 3.74346900e+000 1.43035901e+000
    1.40137517e+000</data></Per_View_Reprojection_Errors>
</opencv_storage>
```

Based on previous code, add before the main loop this instruction:

```
// Intrinsics parameters
aram::Intrinsics intr("C:\\camera_data.xml");
```

And in the tag iteration loop:

```
// Loop over valid tag in current frame
for(it=detector->begin();it!=detector->end();++it)
{
	aram::vecPoint2D imgPoint = (*it)->corners();
	for(unsigned int i=0;i<imgPoint.size();++i)
	{
		cv::circle(frame,imgPoint[i],3,cv::Scalar(200,50,50),3);
	}

	// 3D points corresponding to corners
	aram::vecPoint3D objPoints;
	objPoints.push_back(aram::Point3D(0.0,0.0,0.0));
	objPoints.push_back(aram::Point3D(1.0,0.0,0.0));
	objPoints.push_back(aram::Point3D(1.0,1.0,0.0));
	objPoints.push_back(aram::Point3D(0.0,1.0,0.0));

	aram::Extrinsics e(intr,imgPoint,objPoints);

	aram::Point2D o = e.project(aram::Point3D(0.0,0.0,0.0));
	aram::Point2D x = e.project(aram::Point3D(1.0,0.0,0.0));
	aram::Point2D y = e.project(aram::Point3D(0.0,1.0,0.0));
	aram::Point2D z = e.project(aram::Point3D(0.0,0.0,1.0));

	cv::line(frame,o,x,cv::Scalar(255,0,0),3);
	cv::line(frame,o,y,cv::Scalar(0,255,0),3);
	cv::line(frame,o,z,cv::Scalar(0,0,255),3);
}
```

aram::Extrinsics compute the rotation matrix and translation vector between camera and tag, using openCV cv::solvePnP method. It allow you to project a 3D point (in world coordinate) to a 2D points (in image coordinate) using aram::Extrinsics::project method.

Be careful to the Point3D order!
Standard tag class order Point2D corners in “OpenCV order”:

- Top left
- Bottom left
- Bottom right
- Top right

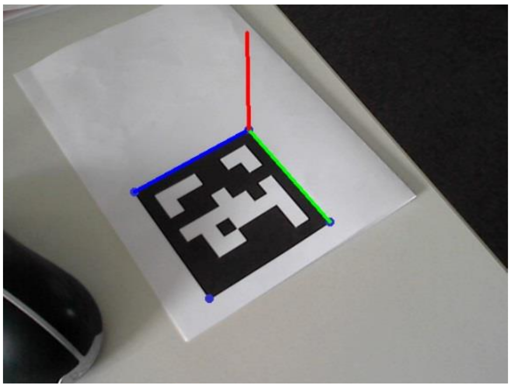
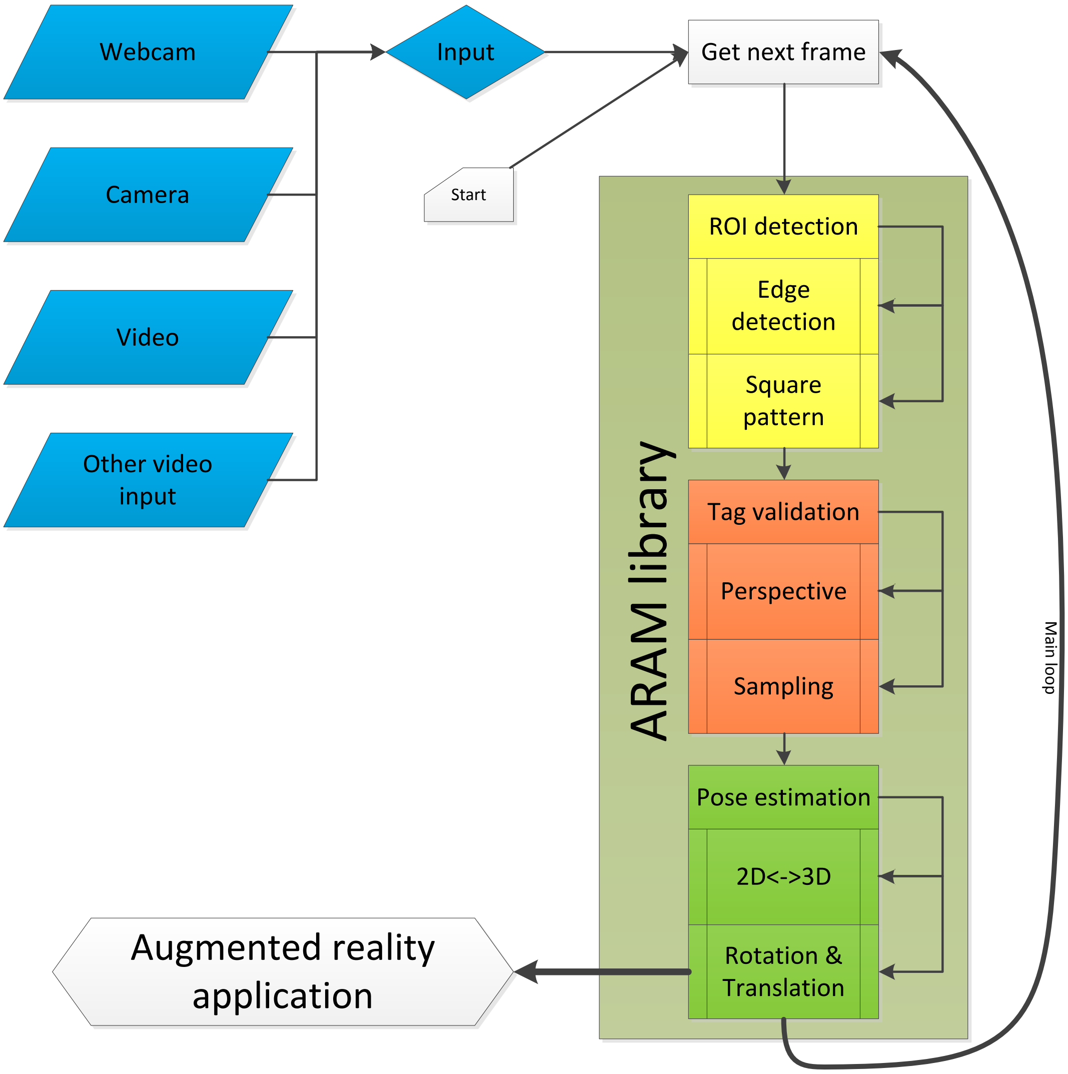


Figure 5 First application output (step 2)

IV. How does ARAM work?

f. Basics



g. ARAM tag dictionary

ARAM is provided with a dictionary of 50 tags. tag[id].png, with id=0..49

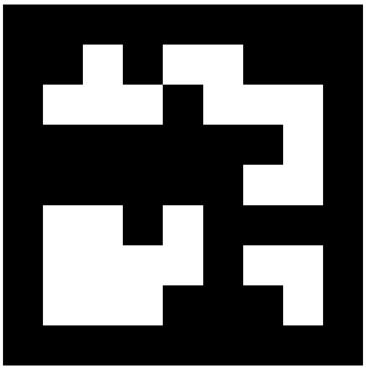


Figure 6 tag0.png

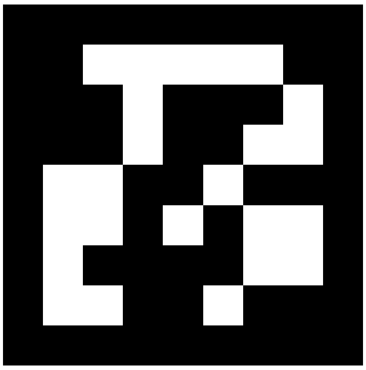


Figure 7 tag1.png

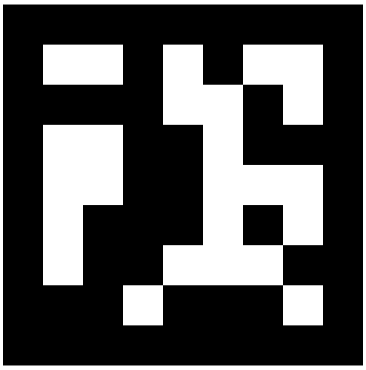


Figure 8 tag3.png

V. Multi tracking

ARAM can use a set of markers that define only one position! Multi tracking is a way to deal with occlusions.

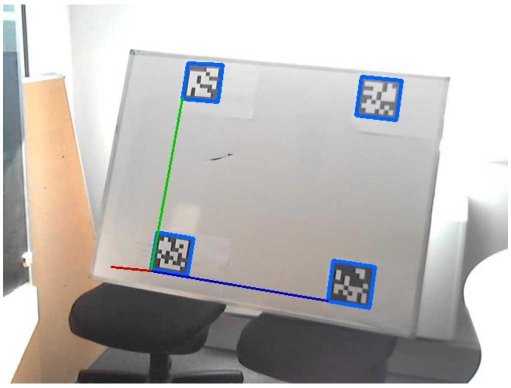


Figure 9 Multi tracking output

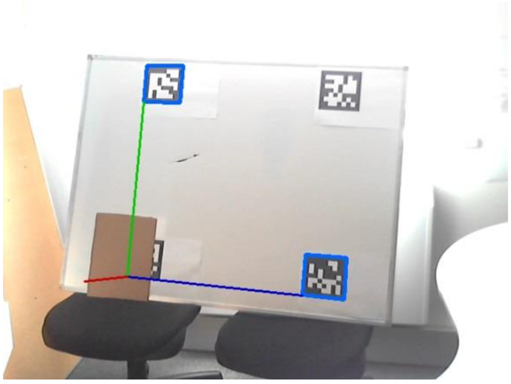
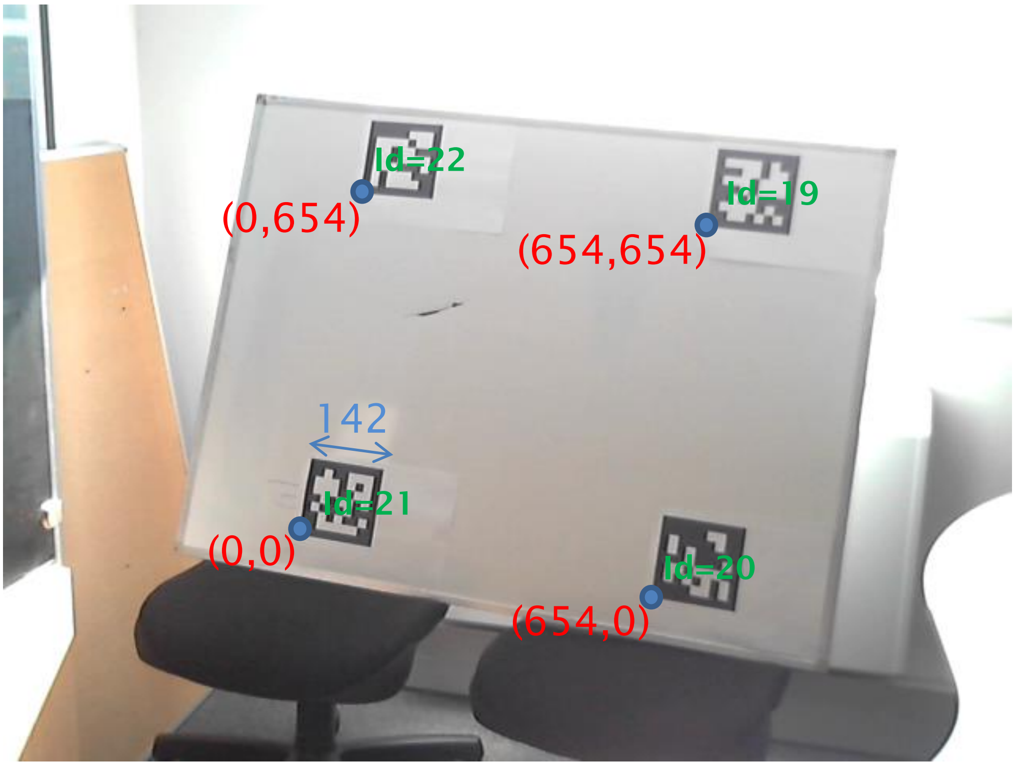


Figure 10 Deal with occlusion

h. Build a Grid

ARAM supports only coplanar tags detection.



Be consistent with your units! In this case, we use millimetres.

In your code:

```
float size = 142.0;
float delta = 654.0;

aram::Grid g;
aram::TagInfo t1(21,aram::Point2D(0.0,0.0),size);
aram::TagInfo t2(22, aram::Point2D(0.0,delta),size);
aram::TagInfo t3(19, aram::Point2D(delta,delta),size);
aram::TagInfo t4(20, aram::Point2D(delta,0.0),size);

g.addTagInfo(t1);
g.addTagInfo(t2);
g.addTagInfo(t3);
g.addTagInfo(t4);
```

i. *Multi tracking application*

For multi tracking operations, we need to know tag's id. aram::HammingTag give an unique id, so we use aram::HammingTag instead of aram::StandardTag in our detector.

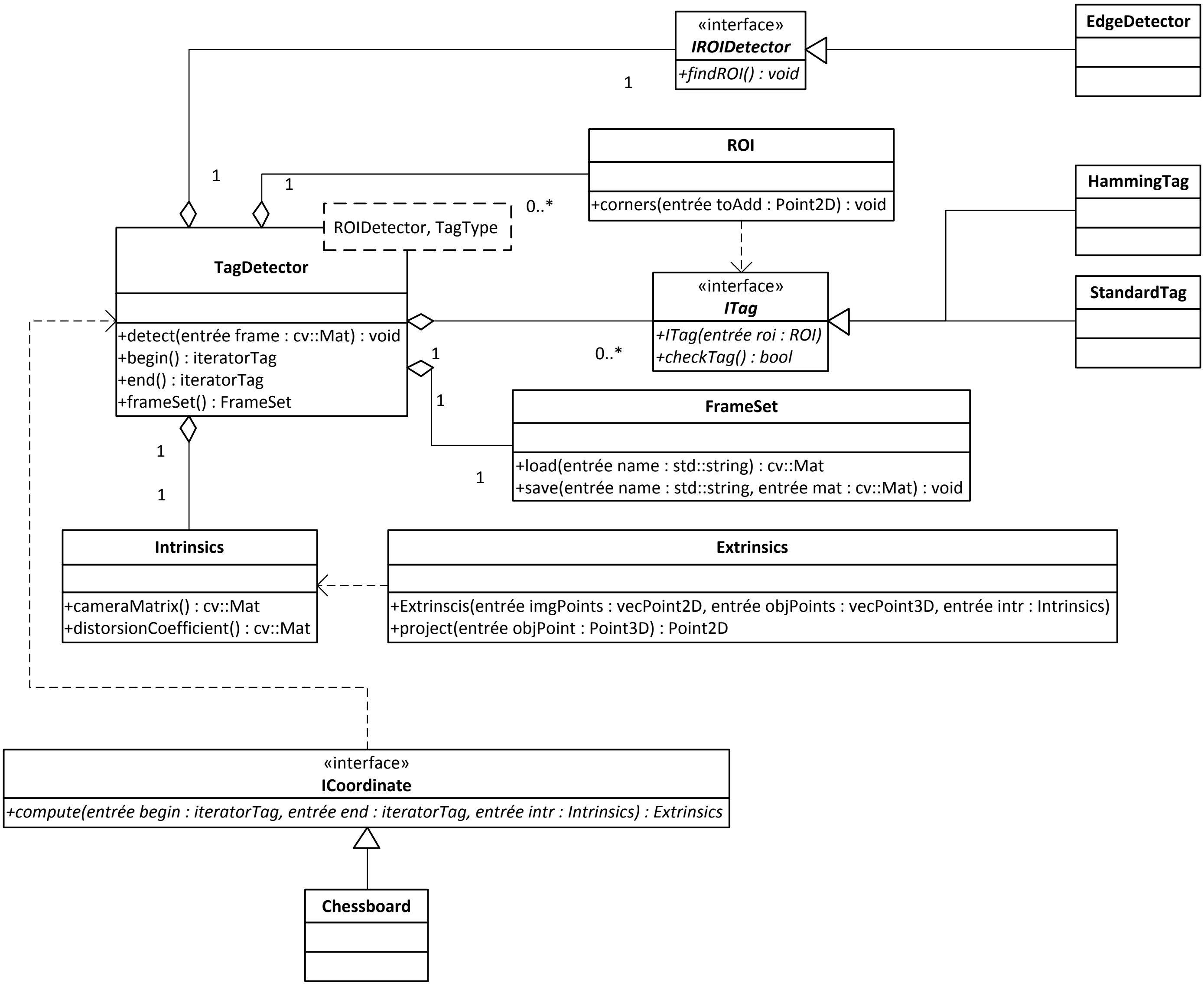
aram::Chessboard::compute return an aram::Extrinsics. We already use aram::Extrinsics in e. Pose estimation.

The minimal code for multi tracking is:

```
#include <ARAM/TagDetector.hpp> // Main ARAM class
#include <ARAM/tag/HammingTag.hpp> // Tag validator
#include <ARAM/ROIDetector/EdgeDetector.hpp> // Region of interest detection

#include <ARAM/tools/Grid.hpp> // Grid an TagInfo
#include <ARAM/coordinate/Chessboard.hpp> // Multi tracking Extrinsics compute

#include <opencv2/opencv.hpp> // OpenCV data structure

#include <exception> //std::exception

int main(int argc, char** argv)
{
	try
	{
		float size = 142.0;
		float delta = 654.0;

		aram::Grid g;
		aram::TagInfo t1(21,aram::Point2D(0.0,0.0),size);
		aram::TagInfo t2(22,aram::Point2D(0.0,delta),size);
		aram::TagInfo t3(19,aram::Point2D(delta,delta),size);
		aram::TagInfo t4(20,aram::Point2D(delta,0.0),size);

		g.addTagInfo(t1);
		g.addTagInfo(t2);
		g.addTagInfo(t3);
		g.addTagInfo(t4);
		aram::Chessboard *coord = new aram::Chessboard(g);


		// Detection parameters :
		// -> Region of interest detection
		// -> Tag validator USE HAMMINGTAG FOR MULTI TRACKING !
		typedef aram::TagDetector<aram::EdgeDetector,aram::HammingTag> myDetector;

		// Tag detector instanciation
		myDetector *detector = new myDetector();

		// Intrinsics parameters
		aram::Intrinsics intr("C:\\camera_data.xml");

		// Video input (see openCV doc)
		cv::VideoCapture cap(0); // use default video (usually your webcam)
```

```
		if(!cap.isOpened()) throw std::exception();

		cv::Mat frame;

		// Main loop
		while(true)
		{
			// next frame from video input
			cap >> frame;

			// Tag detection
			detector->detect(frame);

			// Intrinsics parameters
			aram::Intrinsics intr("C:\\camera_data.xml");

			// Tag list iterator
			aram::iteratorTag it;

			// If any tags was detected
			if(detector->begin()!=detector->end())
			{
				// Get extrinsics parameters
			aram::Extrinsics e = coord->compute(detector->begin(),detector->end(),intr);

				// Project 3D world coordinate -> 2D image coordinate
				aram::Point2D o = e.project(aram::Point3D(0.0,0.0,0.0));
				aram::Point2D x = e.project(aram::Point3D(delta,0.0,0.0));
				aram::Point2D y = e.project(aram::Point3D(0.0,delta,0.0));
				aram::Point2D z = e.project(aram::Point3D(0.0,0.0,delta/2.0));

				// draw axis
				cv::line(frame,o,x,cv::Scalar(200,0,0),2);
				cv::line(frame,o,y,cv::Scalar(0,200,0),2);
				cv::line(frame,o,z,cv::Scalar(0,0,200),2);
			}

			// render
			cv::imshow("render", frame);
			// GUI refresh (see openCV doc)
			if(cv::waitKey(10)>=0) break;
		}
	}
	catch(std::exception &)
	{
	}

	return 0;
}
```

j. UML diagram



2. Revision history

Revision history			
Version	Date	Author	Object
1.0	23.04.2014	Alexandre Kornmann	Creation