ARAM User Guide

Version	2.1	Application date	23.10.2014
Code	REA-PUB-MOP-0001	Revision date	23.10.2015
Applied to		ARAM 2.3	

	Written by	Approved by
Name	Alexandre KORNMANN	Jean-Pierre RADOUX
Function	C++ Developer	Scientific Director

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).
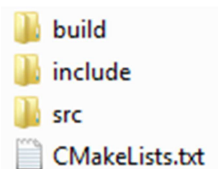-

Figure 1 ARAM Workspace

First, we generate our project file (Visual Studio solution, **.sln**), using **CMake GUI (**Figure 2**)**.
1. Indicate the path of the **CMakeLists.txt on the "where is the source code"**
2. Indicate the path of the **Aram Library on the "where to build the libraries"**
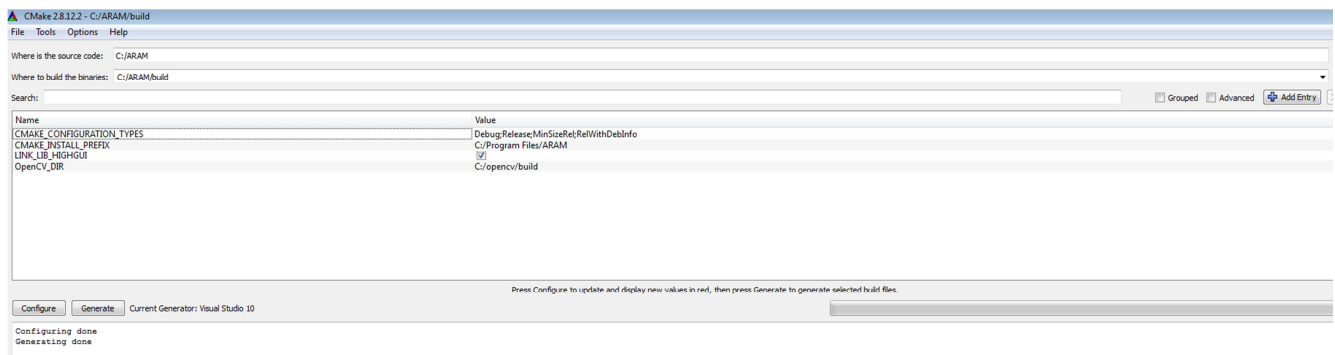

Figure 2 CMake GUI configuration

Click on **Configure** to choose your project generator (visual studio 10 for example)
After click on **Generate**, your directory should contain many files. Open **ARAM.sln**, and start the program!
If CMake fails in OpenCV linking, you have to indicate where OpenCV is in OpenCV_DIR variable. ('opencv/build')
Open the ARAM.sln with visual studio and compile it (Release).

Title:	**ARAM User Guide**		Code	*REA-PUB-MOP-0001*
Written by:	Alexandre Kornmann	Function:	C++ Developer	
Approved by:	Jean-Pierre Radoux	Function:	Scientific Director	
Version :	2.1	Application date	23.10.2014	Page **4 / 20**

III. First application

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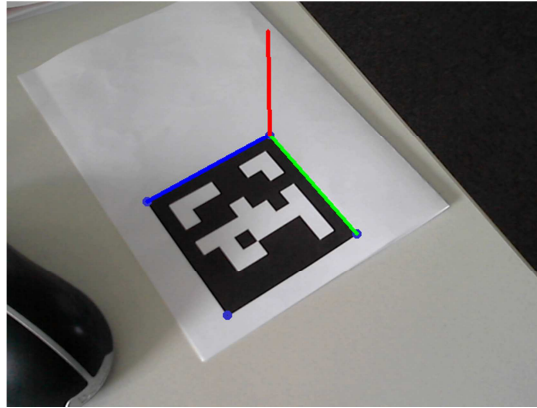

Figure 3 First application output

c. CMakeLists.txt

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

```
cmake_minimum_required(VERSION 2.8)

project(FirstApp)
set(EXECUTABLE_OUTPUT_PATH bin)


# ARAM include
list(APPEND CMAKE_MODULE_PATH ${CMAKE_CURRENT_SOURCE_DIR}/../../cmake)
set(ARAM_DIR CACHE STRING "ARAM install path")
find_package(ARAM REQUIRED)
include_directories(${ARAM_INCLUDE_DIR})

# OpenCV includes
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_LIBRARY} ${OpenCV_LIBS})
```

Remark: Create "main.cpp" in /src. This file contains your code.

d. Tags detection

This is a typical structure for ARAM application:

```cpp
#include <ARAM/TagDetector.hpp> // Main ARAM class
#include <ARAM/tag/HammingTagMatcher.hpp> // Tag validator
#include <ARAM/ROIDetector/CannyFittingDetector.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::CannyFittingDetector, aram::HammingTagMatcher>
myDetector;

                // Tag detector instanciation
                myDetector *detector = new myDetector();
                // Intrinsics parameters
                aram::Intrinsic intr("camera_data.xml");
                // Video input (see openCV doc)
                // use default video input (usually your webcam)
                cv::VideoCapture cap(0);
                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:

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

for example:

```cpp
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);
}
```

Before using this application, you have to paste near to your binary two files:

camera_data.xml
generatedMarker.xml

You can find these two files in samples folder, or generate it, using respectively the OpenCV calibration procedure, and the tools called MarkerGenerator (see g.).
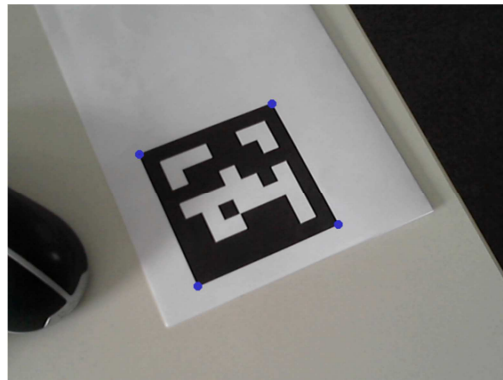
Now, your output must look as following:

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 near my binary.

```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:

```cpp
// Intrinsic parameters
aram::Intrinsic intr("camera_data.xml");
```

MEDIC@ /aLTRan

Title:	**ARAM User Guide**		Code	*REA-PUB-MOP-0001*	
Written by:	Alexandre Kornmann	Function:	C++ Developer		
Approved by:	Jean-Pierre Radoux	Function:	Scientific Director		
Version :	2.1	Application date	23.10.2014	Page **8 / 20**	

And in the tag iteration loop:

```cpp
// 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::Extrinsic 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::Extrinsic 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::Extrinsic::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
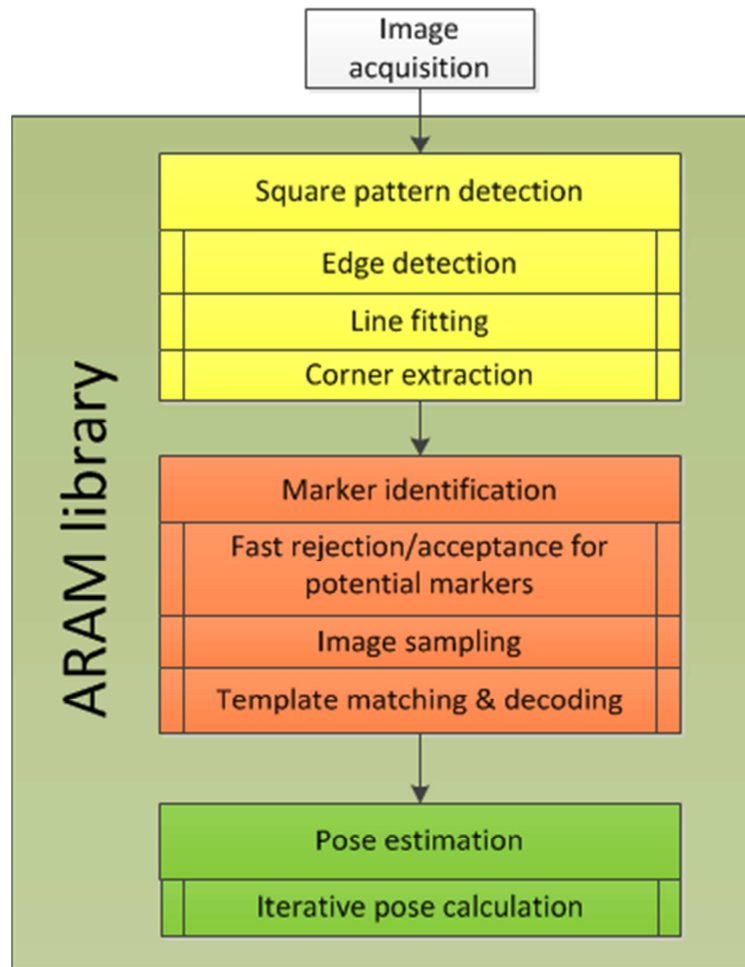
Figure 6 Basic diagramm of ARAM library operation

g. ARAM tag dictionary

ARAM is provided with a dictionary of 50 tags. tag[id].png, with id=0..49, with a file called generatedMarker.xml. This file makes the link between tag's pictures and tags in your application memory.
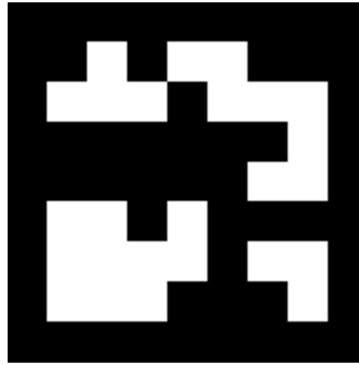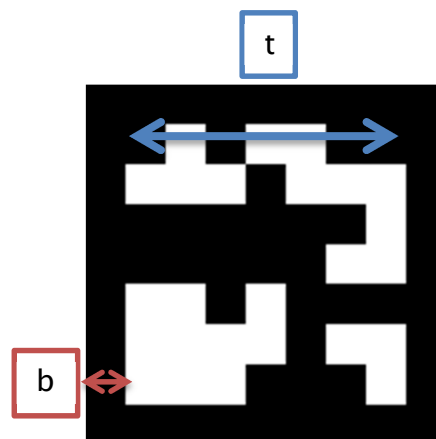

Figure 7 tag0.png

You can also generate your own dictionary, using the additional tools named MarkerGenerator. This tool generate as many tags as you want, associated with a new generatedMarker.xml.

You can use these options in command line:

 -t Tag size (define in number of squares- default value is: 7)

-b Border size (define in number of squares- default value is: 1)

-d Dictionary size (define in number of tag to generate – default value is: 50)

-h Initial hamming distance (default value is: 10)

-r Number of retry before decreasing hamming distance (default value is: 250)

To increase the abilities of the generator to produce more tags, you have to increase the tag size.

To increase the robustness of the dictionary, you have to set a bigger hamming distance. If you set a too big value for hamming distance, after a certain number of failed, the generator automatically decrease the hamming distance (r parameter).

V.   Multi tracking

ARAM can use a set of markers that define only one position! Multi tracking is a way to deal with occlusions.
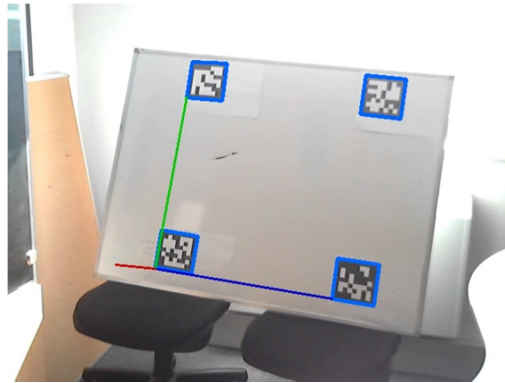
Figure 8 Multi tracking output
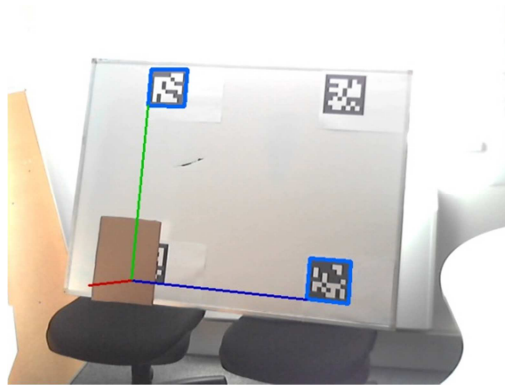
Figure 9 Deal with occlusion

h. Build a Grid

ARAM supports only coplanar tags detection. You can use following examples :
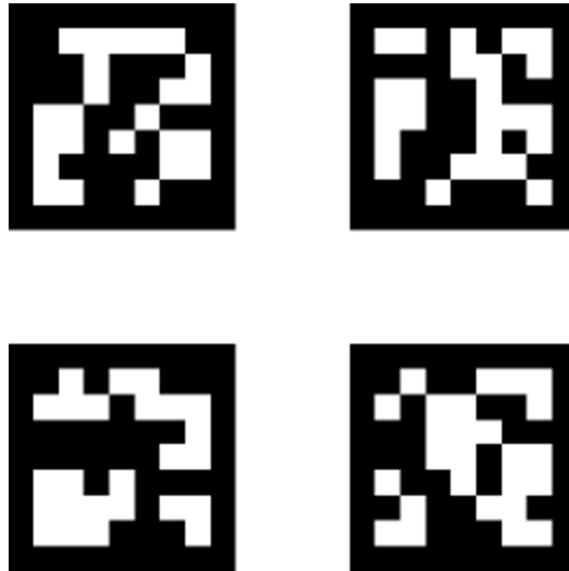
Be consistent with your units! In this case, we use millimeters.

In your code:

```cpp
aram::MultiTag mt;
float m_size = 28.0;
float m_delta = 14.0;

aram::TagInfo t0(0,aram::Point2D(0.0,m_delta+m_size),m_size);
aram::TagInfo t1(1,aram::Point2D(0.0,0.0),m_size);
aram::TagInfo t2(2,aram::Point2D(m_delta+m_size,0.0),m_size);
aram::TagInfo t3(3,aram::Point2D(m_delta+m_size,m_delta+m_size),m_size);

mt.addTagInfo(t0);
mt.addTagInfo(t1);
mt.addTagInfo(t2);
mt.addTagInfo(t3);
```

i. Multi tracking application

aram::MultiTag::compute return an aram::Extrinsic. We already use aram::Extrinsic in e. Pose estimation.

The minimal code for multi tracking is:

```cpp
#include <ARAM/TagDetector.hpp> // Main ARAM class
#include <ARAM/tag/LocalThreshTagMatcher.hpp> // Tag validator
#include <ARAM/ROIDetector/CannyFittingDetector.hpp> // Region of interest detection
#include <ARAM/coordinate/MultiTag.hpp> // Multi tracking

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

#include <exception> //std::exception

void drawPyramide(cv::Mat &mat, float size, aram::Point3D origin, cv::Scalar color, aram::Extrinsic extr)
{
    size *= 0.5;
    // Project 3D world coordinate -> 2D image coordinate
    aram::Point3D a(-size,-size,0.0);
    aram::Point3D b(-size,size,0.0);
    aram::Point3D c(size,size,0.0);
    aram::Point3D d(size,-size,0.0);
    aram::Point3D e(0.0,0.0,-size);

    a+=origin;
    b+=origin;
    c+=origin;
    d+=origin;
    e+=origin;

    aram::Point2D ap = extr.project(a);
    aram::Point2D bp = extr.project(b);
    aram::Point2D cp = extr.project(c);
    aram::Point2D dp = extr.project(d);
    aram::Point2D ep = extr.project(e);


    cv::line(mat,ap,bp,color,2);
    cv::line(mat,bp,cp,color,2);
    cv::line(mat,cp,dp,color,2);
    cv::line(mat,dp,ap,color,2);
    cv::line(mat,ap,ep,color,2);
    cv::line(mat,bp,ep,color,2);
    cv::line(mat,cp,ep,color,2);
    cv::line(mat,dp,ep,color,2);

    return;
}

int main(int argc, char** argv)
{
        try
        {
            // Detection parameters :
```

MEDIC@ /ALTRAN

Title:	**ARAM User Guide**		Code	*REA-PUB-MOP-0001*
Written by:	Alexandre Kornmann	Function:	C++ Developer	
Approved by:	Jean-Pierre Radoux	Function:	Scientific Director	
Version :	2.1	Application date	23.10.2014	Page **16** / **20**

```cpp
            // -> Region of interest detection
            // -> Tag validator
            typedef aram::TagDetector<aram::CannyFittingDetector,aram::LocalThreshTagMatcher>
myDetector;

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

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

            aram::MultiTag mt;
            float m_size = 28.0;
            float m_delta = 14.0;

            aram::TagInfo t0(0,aram::Point2D(0.0,m_delta+m_size),m_size);
            aram::TagInfo t1(1,aram::Point2D(0.0,0.0),m_size);
            aram::TagInfo t2(2,aram::Point2D(m_delta+m_size,0.0),m_size);
            aram::TagInfo t3(3,aram::Point2D(m_delta+m_size,m_delta+m_size),m_size);

            mt.addTagInfo(t0);
            mt.addTagInfo(t1);
            mt.addTagInfo(t2);
            mt.addTagInfo(t3);


            // 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);

                    // Tag list iterator
                    aram::iteratorTag it;

                    for(it=detector->begin();it!=detector->end();++it)
                    {
                            aram::vecPoint2D corners = (*it)->corners();

                            for(unsigned int i=0;i<corners.size();++i)
                            {
        cv::line(frame,corners[i%4],corners[(i+1)%4],cv::Scalar(100,150,150),2);
                            }
                    }

                    // If any tags was detected
                    if(detector->begin()!=detector->end())
                    {
```

```cpp
                    // Get extrinsics parameters
                    aram::Extrinsic e = mt.compute(detector->begin(),detector->end(),intr);

        drawPyramide(frame,m_size*2+m_delta,aram::Point3D(m_size+m_delta/2,m_size+m_delta/2,0),cv::S
calar(0,255,0),e);
                }

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

        return 0;
}
```
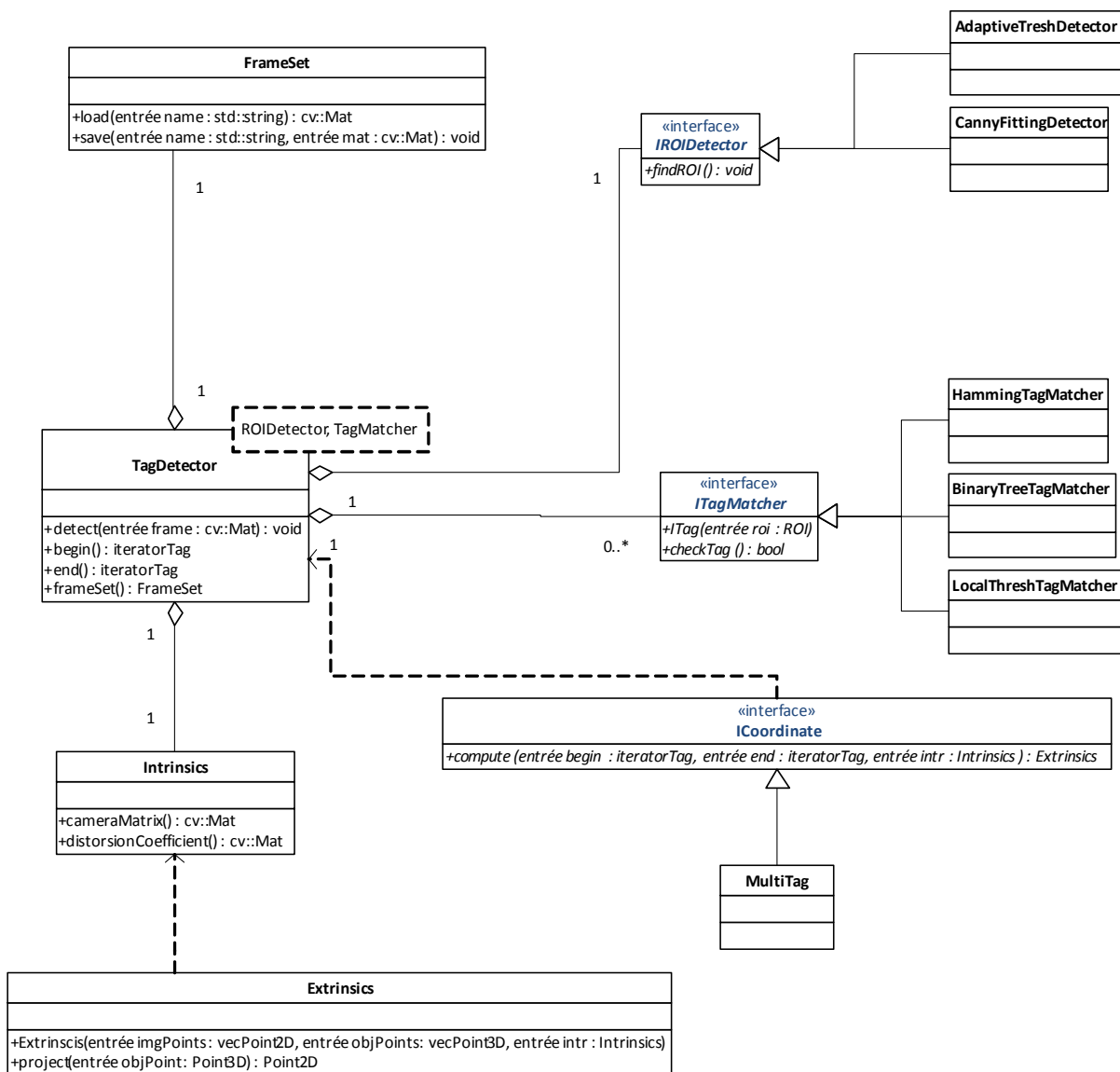
VI. ARAM API

j. UML diagram

FrameSet

+load(entrée name : std:string) : cv::Mat
+save(entrée name : std::string, entrée mat : cv::Mat) : void

«interface»
IROIDetector

+findROI () : void

AdaptiveTreshDetector

CannyFittingDetector

1

1

ROIDetector, TagMatcher

TagDetector

+detect(entrée frame : cv::Mat) : void
+begin() : iteratorTag
+end() : iteratorTag
+frameSet() : FrameSet

1

HammingTagMatcher

BinaryTreeTagMatcher

LocalThreshTagMatcher

«interface»
ITagMatcher

+ITag(entrée roi : ROI)
+checkTag () : bool

1

0..*

1

1

Intrinsics

+cameraMatrix() : cv::Mat
+distorsionCoefficient() : cv::Mat

1

1

«interface»
ICoordinate

+compute (entrée begin : iteratorTag, entrée end : iteratorTag, entrée intr : Intrinsics) : Extrinsics

MultiTag

Extrinsics

+Extrinscis(entrée imgPoints : vecPoint2D, entrée objPoints : vecPoint3D, entrée intr : Intrinsics)
+project(entrée objPoint : Point3D) : Point2D

2. Revision history

Revision history			
Version	**Date**	**Author**	**Object**
1.0	23.04.2014	Alexandre Kornmann	Creation
2.0	29.07.2014	Alexandre Kornmann	Update for ARAM 2.2
2.1	23.10.2014	Alexandre Kornmann	Add MarkerGenerator Fix CMakeLists.txt