Universitat de Girona

Pràctica 2 Visió

Jordi Bolance, 1988989, u1988989@campus.udg.edu, GEINF
Pol Torrent, 1987444, u1987444@campus.udg.edu, GEINF
Marc Cassanmagnago, u1987293, u1987293@campus.udg.edu, GEINF

Divendres, 17:30-19:30, Esteve-Amadeu Hernandez Uptegrove

Girona, 2024/25 3r de GEINF

Introducció

En aquesta pràctica s'ha abordat el desenvolupament d'aplicacions de visió per computadora utilitzant OpenCV, centrant-nos en l'ús de marques fiducials ArUco per a diferents tasques. L'objectiu principal ha estat implementar programes que permetin:

- La generació de marques ArUco individuals i de taulers amb diverses marques.
- La detecció en temps real de les marques ArUco mostrades per càmera.
- El calibratge de la càmera a partir de la captació d'un conjunt d'imatges amb un tauler d'ArUco, obtenint així la matriu de la càmera i els coeficients de distorsió.
- La implementació de realitat augmentada, entenent i estimant la posició i orientació de les marques per sobre impressionar-hi objectes virtuals, com un sistema de coordenades i un cub 3D.

Aquesta pràctica permet posar en relleu la importància d'entendre els principis teòrics darrere del calibratge, la detecció de patrons i la projecció d'objectes 3D en entorns reals, aspectes fonamentals en camps com la robòtica, el cinema i els videojocs.

Explicació dels programes creats

1. generate_marker

Aquest programa rep com a paràmetres el diccionari ArUco, l'identificador del marcador, la mida en píxels del marcador i el nom del fitxer de sortida. La funció drawMarker del mòdul ArUco s'encarrega de generar el codi binari corresponent, mentre que s'afegeix un marge blanc al voltant per millorar el contrast, aspecte clau per la detecció posterior.

2. generate_board

Amb aquest programa es genera un tauler (matriu) de marques ArUco. Es defineix el nombre de files i columnes, la mida individual per cada marca, la separació entre marcs i es fa ús de les primeres marques presents en el diccionari especificat. El tauler es dibuixa en una imatge que posteriorment es desa, facilitant així la creació d'un patró de calibratge.

3. detect_markers

Aquest programa obre la càmera per a la captura de vídeo i, en temps real, detecta les marques ArUco presents a cada frame. Un cop detectades, es dibuixen un requadre verd entorn de cada marca i es destaca la cantonada superior esquerra amb un cercle vermell, a més d'imprimir l'identificador al camp de visió. Aquesta funcionalitat serveix tant per a la validació de la detecció com per a futures aplicacions de RA.

4. calibrate_camera

En aquest programa es realitza el calibratge de la càmera a partir d'un conjunt d'imatges capturades que han de contenir el tauler d'ArUco en diferents posicions i orientacions. S'utilitzen les funcions d'OpenCV per identificar les cantonades dels marcadors i, amb aquesta informació, es calcula la matriu de la càmera i els coeficients de distorsió. Aquests paràmetres és desan en un fitxer YAML per a la seva posterior utilització en la projecció d'objectes virtuals.

5. pose_estimation

Aquest programa s'encarrega d'estimar la posició i orientació d'un marcador concret dins la imatge. A partir dels paràmetres de calibratge i utilitzant els vectors de rotació i translació obtinguts, es dibuixa un sistema de coordenades (amb eixos codificats en diferents colors) i es sobreimprimeixen les coordenades X, Y i Z a la imatge. Això permet verificar que la detecció i el càlcul de la pose són correctes i coherents amb el moviment del marcador.

6. draw_cube

Similars al programa d'estimació de pose, en aquest cas es dibuixa un cub 3D sobre la marca ArUco detectada. Utilitzant la projecció dels punts 3D del cub, es creen les línies que defineixen la seva estructura, fent que el cub aparegui integrat de forma coherent amb el moviment i la perspectiva del món real deteccionat per la càmera.

Codi

generate_marker

```
// Inclou les llibreries necessàries d'OpenCV i C++
#include <opencv2/aruco.hpp>
#include <opencv2/imgcodecs.hpp>
#include <opencv2/core.hpp>
#include <iostream>
#include <string>
#include <map>


using namespace cv;
using namespace std;

// Funció per obtenir el diccionari ArUco a partir del nom passat per paràmetre
Ptr<aruco::Dictionary> seleccionarDiccionari(const string& nomDiccionari) {
    static const map<string, aruco::PREDEFINED_DICTIONARY_NAME> diccionaris = {
        {"DICT_4X4_50", aruco::DICT_4X4_50}, {"DICT_4X4_100", aruco::DICT_4X4_100},
        {"DICT_4X4_250", aruco::DICT_4X4_250}, {"DICT_5X5_50", aruco::DICT_5X5_50},
        {"DICT_6X6_250", aruco::DICT_6X6_250}, {"DICT_ARUCO_ORIGINAL",
aruco::DICT_ARUCO_ORIGINAL}
    };

    auto it = diccionaris.find(nomDiccionari);
    // Retorna el diccionari si existeix, sinó retorna nullptr
    return (it != diccionaris.end()) ? aruco::getPredefinedDictionary(it->second) : nullptr;
}

// Funció per generar i desar una marca ArUco
// - diccionari: diccionari ArUco utilitzat
// - id: identificador de la marca
// - mida: mida de la marca (en píxels)
// - fitxerSortida: nom del fitxer on es desa la imatge
void generarMarca(const Ptr<aruco::Dictionary>& diccionari, int id, int mida, const string&
fitxerSortida) {
    Mat marca;
    // Genera la imatge de la marca ArUco
    aruco::drawMarker(diccionari, id, mida, marca, 1);

    int marge = mida / 8; // Calcula el marge blanc
    // Afegeix un marge blanc al voltant de la marca
    copyMakeBorder(marca, marca, marge, marge, marge, marge, BORDER_CONSTANT,
Scalar(255));
```

```
    // Desa la imatge al fitxer especificat
    if (!imwrite(fitxerSortida, marca)) {
        cerr << "Error en desar el fitxer: " << fitxerSortida << endl;
    } else {
        cout << "Marca generada correctament: " << fitxerSortida << endl;
    }
}

int main(int argc, char** argv) {
    // Comprova que s'han passat els arguments necessaris
    if (argc != 5) {
        cout << "Ús: " << argv[0] << " <diccionari> <id_marca> <mida_pixels> <fitxer_sortida>"
<< endl;
        return 1;
    }

    // Llegeix els arguments de la línia de comandes
    string nomDiccionari = argv[1];
    int idMarca = stoi(argv[2]);
    int midaPixels = stoi(argv[3]);
    string fitxerSortida = argv[4];

    // Selecciona el diccionari d'ArUco
    Ptr<aruco::Dictionary> diccionari = seleccionarDiccionari(nomDiccionari);
    if (!diccionari) {
        cerr << "Error: diccionari '" << nomDiccionari << "' no reconegut." << endl;
        return 1;
    }

    // Genera i desa la marca ArUco
    generarMarca(diccionari, idMarca, midaPixels, fitxerSortida);
    return 0;
}
```

generate_board

```
// Inclou les llibreries necessàries d'OpenCV i C++
#include <opencv2/aruco.hpp>
#include <opencv2/imgcodecs.hpp>
#include <opencv2/core.hpp>
#include <iostream>
#include <map>

using namespace cv;
```

```
using namespace std;

// Funció per obtenir el diccionari ArUco a partir del nom passat per paràmetre
Ptr<aruco::Dictionary> seleccionarDiccionari(const string& nomDiccionari) {
    static const map<string, aruco::PREDEFINED_DICTIONARY_NAME> diccionaris = {
        {"DICT_4X4_50", aruco::DICT_4X4_50}, {"DICT_4X4_100", aruco::DICT_4X4_100},
        {"DICT_4X4_250", aruco::DICT_4X4_250}, {"DICT_5X5_50", aruco::DICT_5X5_50},
        {"DICT_6X6_250", aruco::DICT_6X6_250}, {"DICT_ARUCO_ORIGINAL",
aruco::DICT_ARUCO_ORIGINAL}
    };

    auto it = diccionaris.find(nomDiccionari);
    // Retorna el diccionari si existeix, sinó retorna nullptr
    return (it != diccionaris.end()) ? aruco::getPredefinedDictionary(it->second) : nullptr;
}

// Funció per generar la matriu de marques ArUco i desar-la com a imatge
// - files: nombre de files de la matriu
// - columnes: nombre de columnes de la matriu
// - diccionari: diccionari ArUco utilitzat
// - mida: mida de cada marca (en píxels)
// - separacio: separació entre marques (en píxels)
// - fitxerSortida: nom del fitxer on es desa la imatge
void generarMatriuArUco(int files, int columnes, Ptr<aruco::Dictionary> diccionari,
                        float mida, float separacio, const string& fitxerSortida) {
    // Crea el tauler de marques ArUco
    Ptr<aruco::GridBoard> board = aruco::GridBoard::create(files, columnes, mida, separacio,
diccionari);
    // Calcula la mida de la imatge resultant
    Size imageSize((columnes * mida) + ((columnes - 1) * separacio),
                   (files * mida) + ((files - 1) * separacio));
    Mat boardImage;
    // Dibuixa el tauler a la imatge
    board->draw(imageSize, boardImage, 0, 1);

    // Desa la imatge al fitxer especificat
    if (!imwrite(fitxerSortida, boardImage)) {
        cerr << "Error en desar el fitxer: " << fitxerSortida << endl;
    } else {
        cout << "Matriu ArUco desada correctament a " << fitxerSortida << endl;
    }
}

int main(int argc, char** argv) {
    // Comprova que s'han passat els arguments necessaris
    if (argc != 7) {
        cout << "Ús: " << argv[0] << " <files> <columnes> <diccionari> <mida_pixels>
<separacio_pixels> <fitxer_sortida>" << endl;
```

```
        return 1;
    }

    // Llegeix els arguments de la línia de comandes
    int files = stoi(argv[1]);
    int columnes = stoi(argv[2]);
    string nomDiccionari = argv[3];
    float midaPixels = stof(argv[4]);
    float separacioPixels = stof(argv[5]);
    string fitxerSortida = argv[6];

    // Selecciona el diccionari d'ArUco
    Ptr<aruco::Dictionary> diccionari = seleccionarDiccionari(nomDiccionari);
    if (!diccionari) {
        cerr << "Error: diccionari '" << nomDiccionari << "' no reconegut." << endl;
        return 1;
    }

    // Genera i desa la matriu de marques ArUco
    generarMatriuArUco(files, columnes, diccionari, midaPixels, separacioPixels,
fitxerSortida);
    return 0;
}
```

detect_markers

```
// Inclou les llibreries necessàries d'OpenCV i C++
#include <opencv2/aruco.hpp>
#include <opencv2/videoio.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>
#include <iostream>
#include <map>

using namespace cv;
using namespace std;

// Funció per obtenir el diccionari ArUco a partir del nom passat per paràmetre
Ptr<aruco::Dictionary> seleccionarDiccionari(const string& nomDiccionari) {
    static const map<string, aruco::PREDEFINED_DICTIONARY_NAME> diccionaris = {
        {"DICT_4X4_50", aruco::DICT_4X4_50}, {"DICT_4X4_100", aruco::DICT_4X4_100},
        {"DICT_4X4_250", aruco::DICT_4X4_250}, {"DICT_5X5_50", aruco::DICT_5X5_50},
        {"DICT_6X6_250", aruco::DICT_6X6_250}, {"DICT_ARUCO_ORIGINAL",
aruco::DICT_ARUCO_ORIGINAL}
    };
```

```
    auto it = diccionaris.find(nomDiccionari);
    // Retorna el diccionari si existeix, sinó retorna nullptr
    return (it != diccionaris.end()) ? aruco::getPredefinedDictionary(it->second) : nullptr;
}

// Funció per detectar i dibuixar les marques ArUco a la imatge
void detectarIMostrarMarques(Mat& frame, Ptr<aruco::Dictionary> diccionari) {
    vector<vector<Point2f>> corners; // Cantonades dels marcadors detectats
    vector<int> ids; // Identificadors dels marcadors
    Ptr<aruco::DetectorParameters> params = aruco::DetectorParameters::create(); //
Paràmetres per la detecció

    // Detecta els marcadors ArUco a la imatge
    aruco::detectMarkers(frame, diccionari, corners, ids, params);
    if (!ids.empty()) {
        // Dibuixa els marcadors detectats sobre la imatge
        aruco::drawDetectedMarkers(frame, corners, ids);
        for (size_t i = 0; i < ids.size(); ++i) {
            // Dibuixa un cercle vermell a la cantonada superior esquerra de cada marcador
            circle(frame, corners[i][0], 5, Scalar(0, 0, 255), FILLED);
        }
    }
}

int main(int argc, char** argv) {
    // Comprova que s'ha passat el nom del diccionari per paràmetre
    if (argc != 2) {
        cout << "Ús: " << argv[0] << " <diccionari>" << endl;
        return 1;
    }

    string nomDiccionari = argv[1];
    Ptr<aruco::Dictionary> diccionari = seleccionarDiccionari(nomDiccionari);
    if (!diccionari) {
        cerr << "Error: diccionari '" << nomDiccionari << "' no reconegut." << endl;
        return 1;
    }

    VideoCapture cap(0);
    if (!cap.isOpened()) {
        cerr << "No es pot obrir la càmera" << endl;
        return 1;
    }

    namedWindow("Detecció ArUco", WINDOW_AUTOSIZE);  // 🔧 crea la finestra només
una vegada
```

```
    cout << "Prem ESC per sortir." << endl;
    Mat frame;

    while (cap.read(frame)) {
        detectarIMostrarMarques(frame, diccionari);
        imshow("Detecció ArUco", frame);
        if (waitKey(1) == 27) break; // ESC
    }

    cap.release();
    destroyAllWindows(); // 🔧 tanca la finestra correctament

    return 0;
}
```

calibrate_camera

```
#include <opencv2/aruco.hpp>
#include <opencv2/videoio.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>
#include <opencv2/core.hpp>
#include <opencv2/calib3d.hpp>
#include <opencv2/core/mat.hpp>
#include <iostream>
#include <fstream>
#include <map>

using namespace cv;
using namespace std;

Ptr<aruco::Dictionary> seleccionarDiccionari(const string& nomDiccionari) {
    static const map<string, aruco::PREDEFINED_DICTIONARY_NAME> diccionaris = {
        {"DICT_4X4_50", aruco::DICT_4X4_50}, {"DICT_4X4_100", aruco::DICT_4X4_100},
        {"DICT_4X4_250", aruco::DICT_4X4_250}, {"DICT_5X5_50", aruco::DICT_5X5_50},
        {"DICT_6X6_250", aruco::DICT_6X6_250}, {"DICT_ARUCO_ORIGINAL",
aruco::DICT_ARUCO_ORIGINAL}
    };

    auto it = diccionaris.find(nomDiccionari);
    return (it != diccionaris.end()) ? aruco::getPredefinedDictionary(it->second) : nullptr;
}

int main(int argc, char** argv) {
    if (argc != 8) {
```

```
        cout << "Ús: " << argv[0] << " <diccionari> <detector_params.yaml> <files>
<columnes> <mida_marca> <distancia_marques> <sortida.yaml>" << endl;
        return 1;
    }

    string nomDiccionari = argv[1];
    string fitxerParams = argv[2]; // no s'utilitza realment
    int files = stoi(argv[3]);
    int columnes = stoi(argv[4]);
    float midaMarca = stof(argv[5]);
    float distanciaMarques = stof(argv[6]);
    string fitxerSortida = argv[7];

    Ptr<aruco::Dictionary> diccionari = seleccionarDiccionari(nomDiccionari);
    if (!diccionari) {
        cerr << "Error: diccionari '" << nomDiccionari << "' no reconegut." << endl;
        return 1;
    }

    Ptr<aruco::DetectorParameters> params = aruco::DetectorParameters::create();

    Ptr<aruco::GridBoard> board = aruco::GridBoard::create(files, columnes, midaMarca,
distanciaMarques, diccionari);

    VideoCapture cap(0);
    if (!cap.isOpened()) {
        cerr << "No es pot obrir la càmera." << endl;
        return 1;
    }

    namedWindow("Detecció ArUco", WINDOW_AUTOSIZE);

    vector<vector<Point2f>> allCorners;
    vector<int> allIds;
    vector<Mat> capturedImages;

    cout << "Prem 'c' per capturar una imatge, ESC per finalitzar." << endl;
    Mat frame;

    while (true) {
        cap >> frame;
        if (frame.empty()) continue;

        vector<vector<Point2f>> corners;
        vector<int> ids;

        aruco::detectMarkers(frame, diccionari, corners, ids, params);
        aruco::drawDetectedMarkers(frame, corners, ids);
```

```
        char key = (char)waitKey(30);

        // Només mostra la imatge si es detecten marques o si es prem 'c'
        if (!ids.empty()) {
            imshow("Detecció ArUco", frame);
        }

        if (key == 'c' && !ids.empty()) {
            capturedImages.push_back(frame.clone());
            allCorners.push_back(corners[0]);
            allIds.push_back(ids[0]);
            cout << "Imatge capturada!" << endl;
        } else if (key == 27) {
            break;
        }
    }

    cap.release();
    destroyAllWindows();

    Mat cameraMatrix, distCoeffs;
    vector<Mat> rvecs, tvecs;

    if (!capturedImages.empty()) {
        vector<int> markerCounterPerFrame(capturedImages.size(), 1); // 1 marcador per
imatge
        double error = aruco::calibrateCameraAruco(allCorners, allIds,
markerCounterPerFrame, board, Size(frame.cols, frame.rows), cameraMatrix, distCoeffs,
rvecs, tvecs);
        cout << "Error de calibratge: " << error << endl;

        FileStorage fsOut(fitxerSortida, FileStorage::WRITE);
        fsOut << "camera_matrix" << cameraMatrix;
        fsOut << "distortion_coefficients" << distCoeffs;
        fsOut.release();
        cout << "Paràmetres de calibratge desats a: " << fitxerSortida << endl;
    } else {
        cout << "No s'han capturat imatges suficients per calibrar." << endl;
    }

    return 0;
}
```

pose_estimation

```
#include <opencv2/aruco.hpp>
#include <opencv2/videoio.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>
#include <opencv2/core.hpp>
#include <opencv2/calib3d.hpp>
#include <iostream>
#include <map>

using namespace cv;
using namespace std;

Ptr<aruco::Dictionary> seleccionarDiccionari(const string& nomDiccionari) {
    static const map<string, aruco::PREDEFINED_DICTIONARY_NAME> diccionaris = {
        {"DICT_4X4_50", aruco::DICT_4X4_50}, {"DICT_4X4_100", aruco::DICT_4X4_100},
        {"DICT_4X4_250", aruco::DICT_4X4_250}, {"DICT_5X5_50", aruco::DICT_5X5_50},
        {"DICT_6X6_250", aruco::DICT_6X6_250}, {"DICT_ARUCO_ORIGINAL",
aruco::DICT_ARUCO_ORIGINAL}
    };
    auto it = diccionaris.find(nomDiccionari);
    return (it != diccionaris.end()) ? aruco::getPredefinedDictionary(it->second) : nullptr;
}

void dibuixarSistemaCoordenades(Mat& frame, const vector<Point2f>& corners, const
Vec3d& rvec, const Vec3d& tvec,
                                const Mat& cameraMatrix, const Mat& distCoeffs, float mida) {
    vector<Point3f> axis = {{0, 0, 0}, {mida, 0, 0}, {0, mida, 0}, {0, 0, -mida}};
    vector<Point2f> imagePoints;
    projectPoints(axis, rvec, tvec, cameraMatrix, distCoeffs, imagePoints);

    line(frame, imagePoints[0], imagePoints[1], Scalar(0, 0, 255), 2);  // X
    line(frame, imagePoints[0], imagePoints[2], Scalar(0, 255, 0), 2);  // Y
    line(frame, imagePoints[0], imagePoints[3], Scalar(255, 0, 0), 2);  // Z
}

void detectarIMostrarPose(Mat& frame, Ptr<aruco::Dictionary> diccionari, int markerId, float
midaMarca,
                          const Mat& cameraMatrix, const Mat& distCoeffs) {
    vector<vector<Point2f>> corners;
    vector<int> ids;
    Ptr<aruco::DetectorParameters> params = aruco::DetectorParameters::create();

    aruco::detectMarkers(frame, diccionari, corners, ids, params);
    if (!ids.empty()) {
        aruco::drawDetectedMarkers(frame, corners, ids);

        vector<Vec3d> rvecs, tvecs;
```

```
        aruco::estimatePoseSingleMarkers(corners, midaMarca, cameraMatrix, distCoeffs,
rvecs, tvecs);

        for (size_t i = 0; i < ids.size(); ++i) {
            if (ids[i] == markerId) {
                drawFrameAxes(frame, cameraMatrix, distCoeffs, rvecs[i], tvecs[i], midaMarca *
0.5);
                dibuixarSistemaCoordenades(frame, corners[i], rvecs[i], tvecs[i], cameraMatrix,
distCoeffs, midaMarca);

                putText(frame, "X: " + to_string(tvecs[i][0]) + "m", Point(10, 30),
FONT_HERSHEY_SIMPLEX, 0.7, Scalar(255, 255, 255), 2);
                putText(frame, "Y: " + to_string(tvecs[i][1]) + "m", Point(10, 60),
FONT_HERSHEY_SIMPLEX, 0.7, Scalar(255, 255, 255), 2);
                putText(frame, "Z: " + to_string(tvecs[i][2]) + "m", Point(10, 90),
FONT_HERSHEY_SIMPLEX, 0.7, Scalar(255, 255, 255), 2);
            }
        }
    }
}

int main(int argc, char** argv) {
    if (argc != 4) {
        cout << "\u00das: " << argv[0] << " <diccionari> <id_marca> <mida_marca_m>" <<
endl;
        return 1;
    }

    string nomDiccionari = argv[1];
    int markerId = stoi(argv[2]);
    float midaMarca = stof(argv[3]);

    Ptr<aruco::Dictionary> diccionari = seleccionarDiccionari(nomDiccionari);
    if (!diccionari) {
        cerr << "Error: diccionari no reconegut." << endl;
        return 1;
    }

    Mat cameraMatrix, distCoeffs;
    FileStorage fs("camera_calibration.yaml", FileStorage::READ);
    if (!fs.isOpened()) {
        cerr << "Error: no s'ha pogut llegir camera_calibration.yaml" << endl;
        return 1;
    }
    fs["camera_matrix"] >> cameraMatrix;
    fs["distortion_coefficients"] >> distCoeffs;
    fs.release();
```

```
    VideoCapture cap(0);
    if (!cap.isOpened()) {
        cerr << "\u274c No es pot obrir la c\u00e0mera. Comprova VirtualBox o prova amb
v\u00eddeo!" << endl;
        return 1;
    }

    namedWindow("Estimaci\u00f3 de posici\u00f3 ArUco", WINDOW_AUTOSIZE);

    cout << "Prem ESC per sortir." << endl;
    Mat frame;
    while (cap.isOpened()) {
        cap >> frame;
        if (frame.empty()) {
            cerr << "\u26a0\ufe0f  No s'ha rebut cap imatge de la c\u00e0mera (potser la
c\u00e0mera no funciona amb OpenCV)." << endl;
            break;
        }

        detectarIMostrarPose(frame, diccionari, markerId, midaMarca, cameraMatrix,
distCoeffs);

        imshow("Estimaci\u00f3 de posici\u00f3 ArUco", frame);

        if (getWindowProperty("Estimaci\u00f3 de posici\u00f3 ArUco", WND_PROP_VISIBLE)
< 1) {
            break;
        }

        if (waitKey(30) == 27) {
            break;
        }
    }

    destroyAllWindows();
    return 0;
}
```

draw_cube

```
#include <opencv2/aruco.hpp>
#include <opencv2/videoio.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>
```

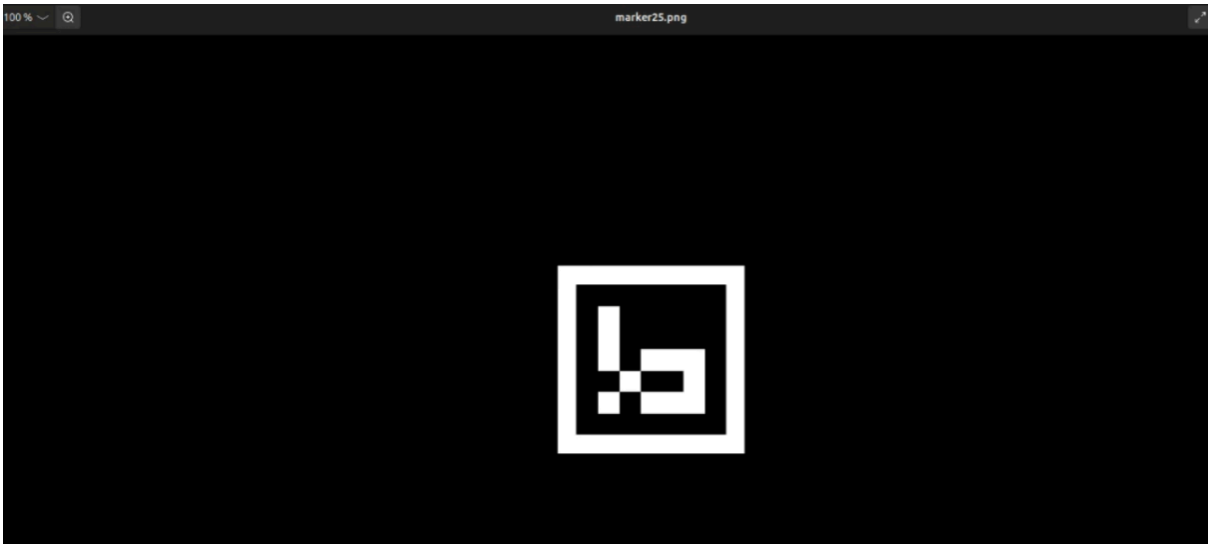
```
#include <opencv2/core.hpp>
#include <opencv2/calib3d.hpp>
#include <iostream>
#include <map>

using namespace cv;
using namespace std;

// Funció per obtenir el diccionari ArUco a partir del nom passat per paràmetre
Ptr<aruco::Dictionary> seleccionarDiccionari(const string& nomDiccionari) {
    static const map<string, aruco::PREDEFINED_DICTIONARY_NAME> diccionaris = {
        {"DICT_4X4_50", aruco::DICT_4X4_50}, {"DICT_4X4_100", aruco::DICT_4X4_100},
        {"DICT_4X4_250", aruco::DICT_4X4_250}, {"DICT_5X5_50", aruco::DICT_5X5_50},
        {"DICT_6X6_250", aruco::DICT_6X6_250}, {"DICT_ARUCO_ORIGINAL",
aruco::DICT_ARUCO_ORIGINAL}
    };

    auto it = diccionaris.find(nomDiccionari);
    return (it != diccionaris.end()) ? aruco::getPredefinedDictionary(it->second) : nullptr;
}

// Funció per dibuixar el cub 3D sobre la marca ArUco
void dibuixarCub(Mat& frame, const vector<Point2f>& corners, const Vec3d& rvec, const
Vec3d& tvec, const Mat& cameraMatrix, const Mat& distCoeffs, float mida) {
    vector<Point3f> cub = {
        {0, 0, 0}, {mida, 0, 0}, {mida, mida, 0}, {0, mida, 0},
        {0, 0, -mida}, {mida, 0, -mida}, {mida, mida, -mida}, {0, mida, -mida}
    };
    vector<Point2f> imagePoints;
    projectPoints(cub, rvec, tvec, cameraMatrix, distCoeffs, imagePoints);

    for (int i = 0; i < 4; i++) {
        line(frame, imagePoints[i], imagePoints[(i + 1) % 4], Scalar(0, 255, 0), 2);
        line(frame, imagePoints[i + 4], imagePoints[(i + 1) % 4 + 4], Scalar(0, 255, 0), 2);
        line(frame, imagePoints[i], imagePoints[i + 4], Scalar(0, 255, 0), 2);
    }
}

// Funció per detectar la marca ArUco i dibuixar el cub si es troba la marca amb l'id indicat
void detectarIMostrarCub(Mat& frame, Ptr<aruco::Dictionary> diccionari, int markerId, float
midaMarca, const Mat& cameraMatrix, const Mat& distCoeffs) {
    vector<vector<Point2f>> corners;
    vector<int> ids;
    Ptr<aruco::DetectorParameters> params = aruco::DetectorParameters::create();

    aruco::detectMarkers(frame, diccionari, corners, ids, params);
    if (!ids.empty()) {
        aruco::drawDetectedMarkers(frame, corners, ids);
```

```
        vector<Vec3d> rvecs, tvecs;
        aruco::estimatePoseSingleMarkers(corners, midaMarca, cameraMatrix, distCoeffs,
rvecs, tvecs);

        for (size_t i = 0; i < ids.size(); ++i) {
            if (ids[i] == markerId) {
                dibuixarCub(frame, corners[i], rvecs[i], tvecs[i], cameraMatrix, distCoeffs,
midaMarca);
            }
        }
    }
}

int main(int argc, char** argv) {
    if (argc != 4) {
        cout << "Ús: " << argv[0] << " <diccionari> <id_marca> <mida_marca_m>" << endl;
        return 1;
    }

    string nomDiccionari = argv[1];
    int markerId = stoi(argv[2]);
    float midaMarca = stof(argv[3]);

    Ptr<aruco::Dictionary> diccionari = seleccionarDiccionari(nomDiccionari);
    if (!diccionari) {
        cerr << "Error: diccionari '" << nomDiccionari << "' no reconegut." << endl;
        return 1;
    }

    Mat cameraMatrix, distCoeffs;
    FileStorage fs("camera_calibration.yaml", FileStorage::READ);
    if (fs.isOpened()) {
        fs["camera_matrix"] >> cameraMatrix;
        fs["distortion_coefficients"] >> distCoeffs;
        fs.release();
    } else {
        cerr << "Error: no es troben els paràmetres de calibratge." << endl;
        return 1;
    }

    VideoCapture cap(0);
    if (!cap.isOpened()) {
        cerr << "No es pot obrir la càmera." << endl;
        return 1;
    }

    cout << "Prem ESC per sortir." << endl;
```

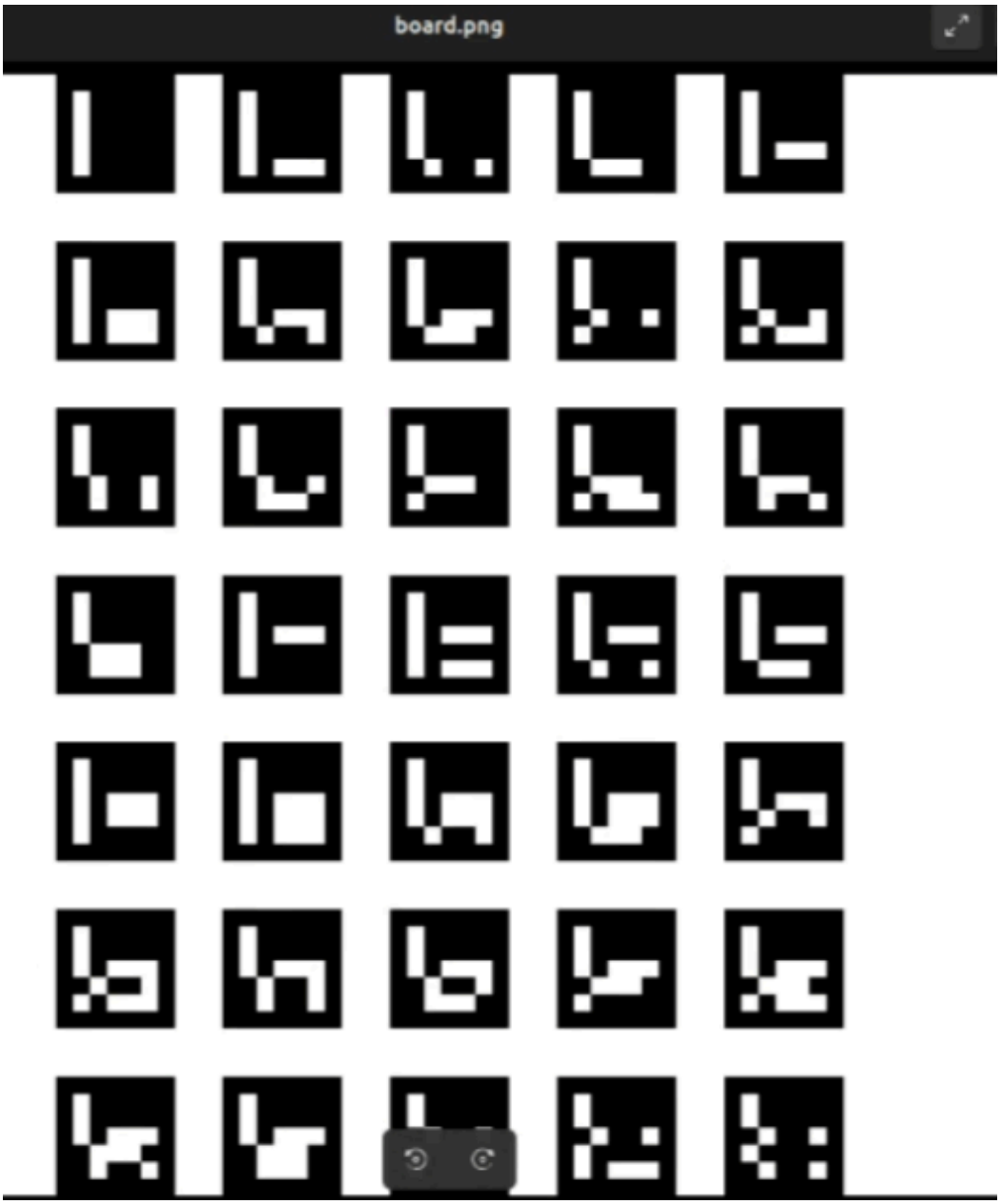
```
    Mat frame;
    while (cap.read(frame)) {
        detectarIMostrarCub(frame, diccionari, markerId, midaMarca, cameraMatrix,
distCoeffs);
        imshow("Realitat augmentada - Cub 3D", frame);
        if (waitKey(1) == 27) break;
    }

    return 0;
}
```

Imatges resultants

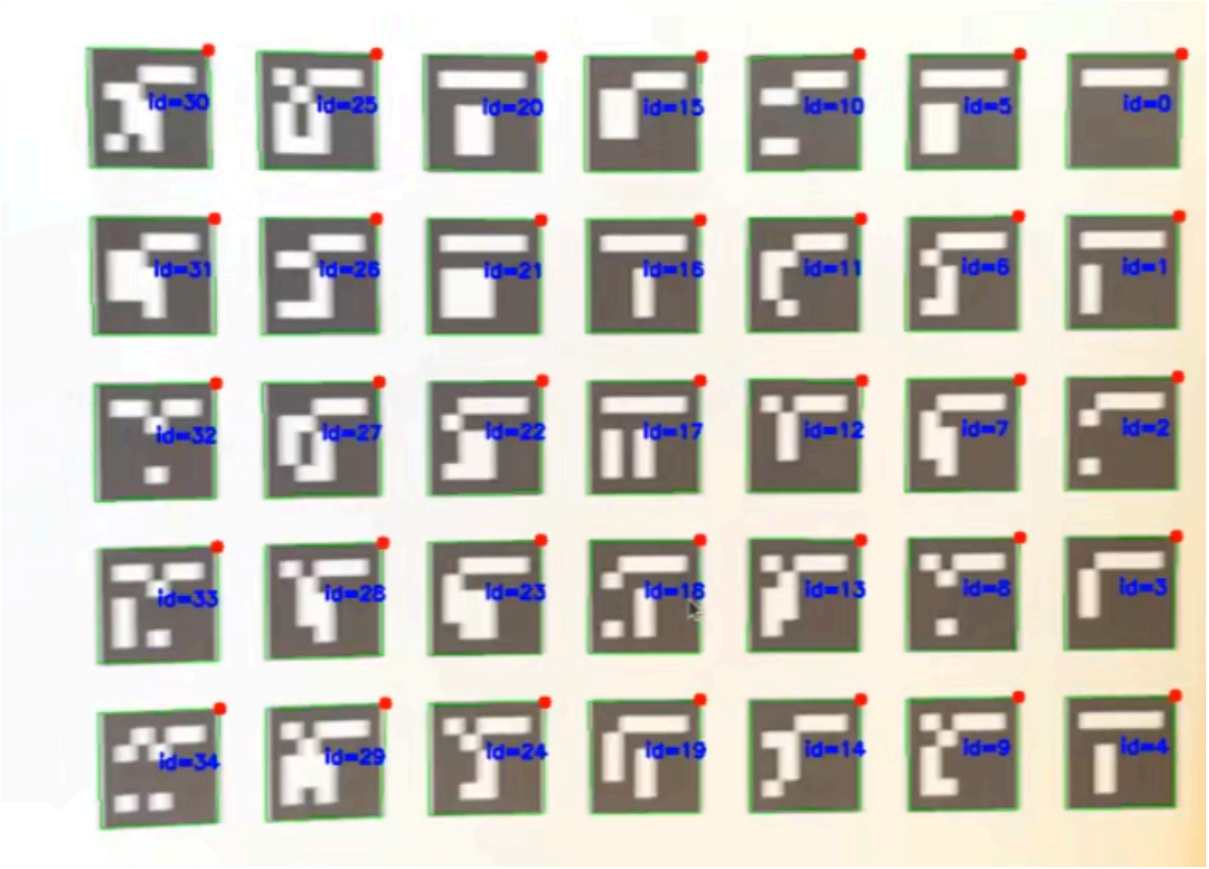
generate_marker



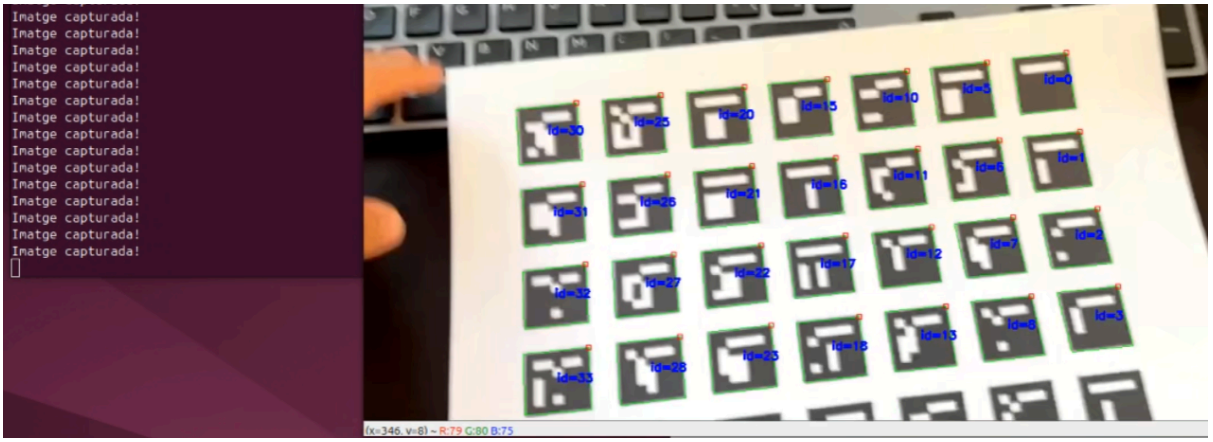
generate_board

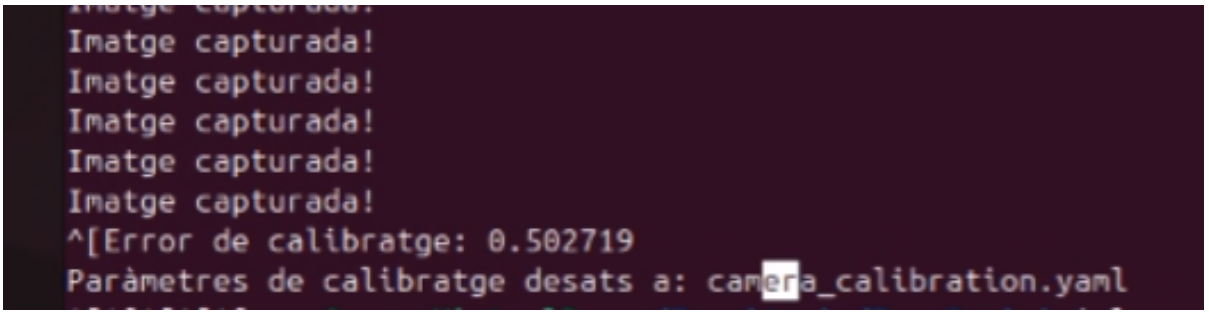


detect_marker

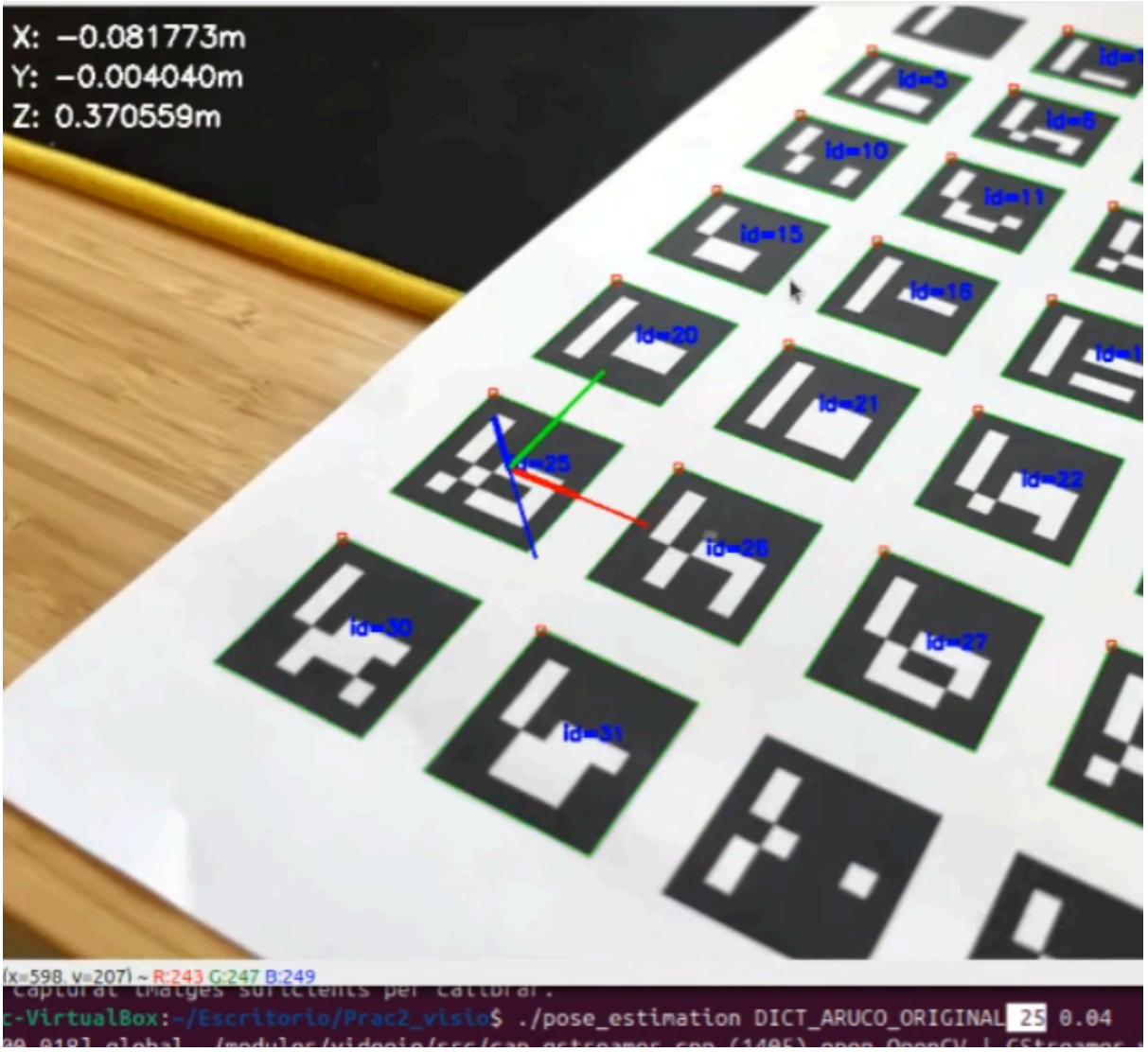


calibrate_camera

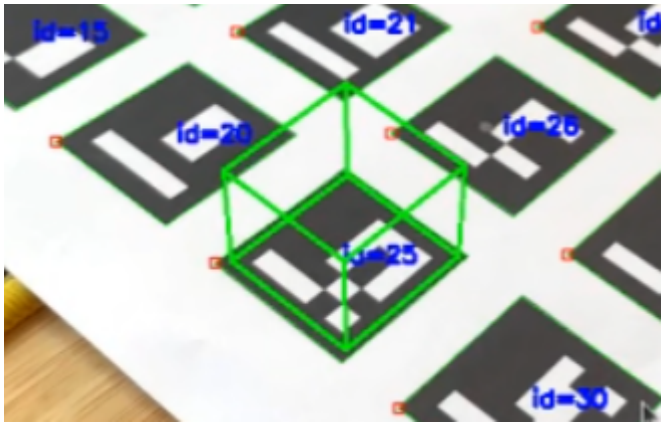




pose_estimation



draw_cube



Aspectes de millora

Optimització del processat en temps real: Encara que la detecció de marcadors i l'estimació de la pose funcionen de forma acceptable, es pot millorar la velocitat i eficiència del codi. Això podria incloure la paral·lelització de certes operacions o la reducció de la resolució d'imatge en situacions en què la precisió no es veu compromesa.

Gestió d'errors i robustesa: Implementar una gestió més robusta dels errors pot ajudar a evitar interrupcions en entorns amb poca il·luminació, soroll excessiu o problemes en la captura d'imatges. Per exemple, es podria afegir detecció de situacions en què el fitxer YAML de calibratge no es trobi o que la càmera no obri correctament.

Afinació dels paràmetres de detecció: Els paràmetres del detector ArUco (definits en el fitxer YAML) poden ser ajustats per millorar la detecció en diferents condicions ambientals. Experimentar amb aquests valors pot portar a una detecció més estable i fiable dels marcadors.

Interfície d'usuari: Tot i que l'execució actual es basa en línia d'ordres amb sortida per a consola i finestres d'OpenCV, una interfície gràfica d'usuari (GUI) podria millorar l'experiència, permetent als usuaris ajustar fàcilment paràmetres i visualitzar els resultats.

Extensió a altres fiducials o funcions: Encara que la pràctica es centra en les marques ArUco, es podria ampliar el projecte per integrar altres tipus de fiducials o tècniques de detecció, així com la implementació d'altres objectes 3D més complexos en la realitat augmentada.

Conclusions

La realització d'aquesta pràctica ha permès integrar i posar en pràctica diversos conceptes essencials en el camp de la visió per computadora i la realitat augmentada. S'ha après a generar i gestionar marques fiducials amb ArUco, a calibrar una càmera per millorar la precisió en la projecció d'objectes 3D, i a implementar aplicacions en temps real que combinen el món digital amb el real.

L'experiència ha demostrat la importància de tenir un bon calibratge per obtenir dades fiables i ha ressaltat com la detecció de patrons poden ser la base per a aplicacions més avançades en augmented reality. A més, el projecte evidencia com la integració d'eines com OpenCV pot simplificar processos complexos, fent-los accessibles tant en àmbits acadèmics com en aplicacions industrials.

Finalment, els possibles aspectes de millora esdevenen llançaments per a futures ampliacions, ja que millorar la robustesa, optimitzar el processat i desenvolupar interfícies més amigables pot portar aquest projecte a un nivell més alt tant en precisió com en aplicabilitat pràctica.