Kibo-RPC AR 標籤檢測與影像矯正教學

新增功能概述

這個版本的程式碼在前兩個範例的基礎上，新增了 AR 標籤檢測和影像矯正功能。這些功能是 Kibo-RPC 中物品識別的核心技術，能夠準確檢測物品上的 AR 標籤並矯正相機鏡頭造成的影像畸變。

完整程式碼

```java
package jp.jaxa.iss.kibo.rpc.sampleapk;

import jp.jaxa.iss.kibo.rpc.api.KiboRpcService;

import gov.nasa.arc.astrobee.types.Point;
import gov.nasa.arc.astrobee.types.Quaternion;

import org.opencv.core.Mat;

// new imports
import android.util.Log;

import java.util.List;
import java.util.ArrayList;


// new OpenCV imports
import org.opencv.aruco.Dictionary;
import org.opencv.aruco.Aruco;
import org.opencv.calib3d.Calib3d;
import org.opencv.core.CvType;
/**
 * Class meant to handle commands from the Ground Data System and execute them in
Astrobee.
 */

public class YourService extends KiboRpcService {

    // The TAG is used for logging.
    // You can use it to check the log in the Android Studio.
    private final String TAG = this.getClass().getSimpleName();

    @Override
    protected void runPlan1(){

        // Log the start of the mission.
        Log.i(TAG, "Start mission");
        // The mission starts.
        api.startMission();
```

```
        // Move to a point.
        Point point = new Point(10.9d, -9.92284d, 5.195d);
        Quaternion quaternion = new Quaternion(0f, 0f, -0.707f, 0.707f);
        api.moveTo(point, quaternion, false);

        // Get a camera image.
        Mat image = api.getMatNavCam();
        // Save the image to a file.
        api.saveMatImage(image, "test.png");


        /*
********************************************************************************
*/
        /* Write your code to recognize the type and number of landmark items in
each area! */
        /* If there is a treasure item, remember it.
*/
        /*
********************************************************************************
*/

        //
        /**
         * Retrieves a predefined Aruco dictionary for 6x6 markers containing 250
distinct patterns.
         * This dictionary is used for detecting and tracking Aruco markers in
images.
         *
         * The call to Aruco.getPredefinedDictionary(Aruco.DICT_6X6_250) selects a
standard set of marker patterns,
         * making it easier to consistently identify markers during image
processing.
         */
        Dictionary dictionary = Aruco.getPredefinedDictionary(Aruco.DICT_5X5_250);

        // Detect markers in the image using the specified dictionary.
        // The detectMarkers function analyzes the image and identifies the
locations of Aruco markers.
        // The detected markers are stored in the corners list.
        // The corners list contains the coordinates of the detected markers in
the image.

        List<Mat> corners = new ArrayList<>();
        Mat ids = new Mat();
        // The ids list contains the IDs of the detected markers.
        Aruco.detectMarkers(image, dictionary, corners, ids);



        // Undistort the detected markers using the camera matrix and distortion
coefficients.
        Mat cameraMatrix = new Mat(3,3,CvTtpe.CV64F);
        cameraMatrix.put(0, 0, api.getNavCamIntrinsics()[0]); //
```

```
getNavCamIntrinsics will  return  ( cameraMatrix, distortionCoefficients )

        // Get len distortion parameters
        Mat cameraCoefficients = new Mat(1, 5, CvType.CV_64F);
        cameraCoefficients.put(0, 0, api.getNavCamIntrinsics()[1]);
        cameraCoefficients.convertTo(cameraCoefficients, CvType.CV_64F);

        // Undistort the detected markers using the camera matrix and distortion
coefficients.
        Mat UndistortImg = new Mat();
        Calib3d.undistort(image, UndistortImg, cameraMatrix, cameraCoefficients);





        // When you recognize landmark items, let's set the type and number.
        api.setAreaInfo(1, "item_name", 1);

        /* **************************************************** */
        /* Let's move to each area and recognize the items. */
        /* **************************************************** */

        // When you move to the front of the astronaut, report the rounding
completion.
        point = new Point(11.143d, -6.7607d, 4.9654d);
        quaternion = new Quaternion(0f, 0f, 0.707f, 0.707f);
        api.moveTo(point, quaternion, false);
        api.reportRoundingCompletion();

        /* ********************************************************** */
        /* Write your code to recognize which target item the astronaut has. */
        /* ********************************************************** */

        // Let's notify the astronaut when you recognize it.
        api.notifyRecognitionItem();

        /*
**********************************************************************************
******************** */
        /* Write your code to move Astrobee to the location of the target item
(what the astronaut is looking for) */
        /*
**********************************************************************************
******************** */

        // Take a snapshot of the target item.
        api.takeTargetItemSnapshot();
    }

    @Override
    protected void runPlan2(){
       // write your plan 2 here.
    }
```

```
    @Override
    protected void runPlan3(){
        // write your plan 3 here.
    }

    // You can add your method.
    private String yourMethod(){
        return "your method";
    }
}
```

新增的程式碼解析

1. 新增匯入

```
import java.util.List;
import java.util.ArrayList;

// new OpenCV imports
import org.opencv.aruco.Dictionary;
import org.opencv.aruco.Aruco;
import org.opencv.calib3d.Calib3d;
import org.opencv.core.CvType;
```

說明：

- `List` 和 `ArrayList` - Java 集合類別，用於儲存檢測到的 AR 標籤資料
- `Dictionary` - ArUco 字典類別，定義 AR 標籤的圖案集合
- `Aruco` - OpenCV 的 AR 標籤檢測模組
- `Calib3d` - OpenCV 的相機校正和 3D 重建模組
- `CvType` - OpenCV 資料類型定義

2. AR 標籤字典建立

```
/**
 * Retrieves a predefined Aruco dictionary for 6x6 markers containing 250 distinct patterns.
 * This dictionary is used for detecting and tracking Aruco markers in images.
 *
 * The call to Aruco.getPredefinedDictionary(Aruco.DICT_6X6_250) selects a standard set of marker patterns,
 * making it easier to consistently identify markers during image processing.
 */
Dictionary dictionary = Aruco.getPredefinedDictionary(Aruco.DICT_5X5_250);
```

程式碼解析：

- `Aruco.getPredefinedDictionary()` - 獲取預定義的 ArUco 字典
- `Aruco.DICT_5X5_250` - 選用 5x5 像素的 AR 標籤，包含 250 種不同圖案
- **注意：**程式碼註解中提到 6x6，但實際使用的是 5x5，這是一個不一致的地方

AR 標籤字典類型說明：

```
Aruco.DICT_4X4_50     // 4x4 像素，50 種圖案
Aruco.DICT_4X4_100    // 4x4 像素，100 種圖案
Aruco.DICT_5X5_250    // 5x5 像素，250 種圖案（本程式使用）
Aruco.DICT_6X6_250    // 6x6 像素，250 種圖案
```

3. AR 標籤檢測

```
// Detect markers in the image using the specified dictionary.
// The detectMarkers function analyzes the image and identifies the locations of
Aruco markers.
// The detected markers are stored in the corners list.
// The corners list contains the coordinates of the detected markers in the image.

List<Mat> corners = new ArrayList<>();
Mat ids = new Mat();
// The ids list contains the IDs of the detected markers.
Aruco.detectMarkers(image, dictionary, corners, ids);
```

程式碼解析：

- `List<Mat> corners` - 儲存檢測到的 AR 標籤四個角落座標
- `Mat ids` - 儲存檢測到的 AR 標籤 ID 編號
- `Aruco.detectMarkers()` - 執行 AR 標籤檢測的主要函式

檢測結果說明：

- `corners` - 每個檢測到的標籤會有一個 Mat，包含四個角落的 (x,y) 座標
- `ids` - 對應每個標籤的唯一識別編號
- 如果沒有檢測到標籤，這些容器會是空的

4. 相機內參數獲取

```
// Undistort the detected markers using the camera matrix and distortion
coefficients.
Mat cameraMatrix = new Mat(3,3,CvTtpe.CV64F);
cameraMatrix.put(0, 0, api.getNavCamIntrinsics()[0]); // getNavCamIntrinsics will
return  ( cameraMatrix, distortionCoefficients )
```

程式碼解析：

- `Mat cameraMatrix` - 創建 3x3 的相機內參數矩陣
- **錯誤**：`CvTtpe.CV64F` 應該是 `CvType.CV_64F`
- `api.getNavCamIntrinsics()[0]` - 獲取相機內參數矩陣

相機內參數矩陣說明：

```
[fx  0  cx]
[ 0 fy  cy]
[ 0  0   1]
```

- fx, fy：焦距參數
- cx, cy：主點座標

5. 鏡頭畸變參數獲取

```
// Get len distortion parameters
Mat cameraCoefficients = new Mat(1, 5, CvType.CV_64F);
cameraCoefficients.put(0, 0, api.getNavCamIntrinsics()[1]);
cameraCoefficients.convertTo(cameraCoefficients, CvType.CV_64F);
```

程式碼解析：

- `Mat cameraCoefficients` - 創建 1x5 的畸變係數矩陣
- `api.getNavCamIntrinsics()[1]` - 獲取鏡頭畸變參數
- `convertTo()` - 確保資料類型為 64 位元浮點數

畸變係數說明：

```
[k1, k2, p1, p2, k3]
```

- k1, k2, k3：徑向畸變係數
- p1, p2：切向畸變係數

6. 影像矯正

```
// Undistort the detected markers using the camera matrix and distortion
coefficients.
Mat UndistortImg = new Mat();
Calib3d.undistort(image, UndistortImg, cameraMatrix, cameraCoefficients);
```

程式碼解析：

- `Mat UndistortImg` - 儲存矯正後的影像
- `Calib3d.undistort()` - 使用相機參數矯正影像畸變

矯正效果說明：

- 消除鏡頭造成的桶狀或枕狀畸變
- 讓直線在影像中保持直線
- 提高後續影像處理的精確度

實際應用建議

1. AR 標籤檢測最佳實務

```
private void detectAndLogMarkers(Mat image, String imageName) {
    Dictionary dictionary = Aruco.getPredefinedDictionary(Aruco.DICT_5X5_250);
    List<Mat> corners = new ArrayList<>();
    Mat ids = new Mat();

    Aruco.detectMarkers(image, dictionary, corners, ids);

    // 記錄檢測結果
    Log.i(TAG, "Detected " + corners.size() + " markers in " + imageName);

    // 如果有檢測到標籤，記錄詳細資訊
    if (!corners.isEmpty()) {
        for (int i = 0; i < corners.size(); i++) {
            Log.i(TAG, "Marker " + i + " ID: " + ids.get(i, 0)[0]);
        }
    }
}
```

2. 完整的影像處理流程

```
private Mat processImageWithUndistortion(String savePrefix) {
    // 1. 獲取原始影像
    Mat rawImage = api.getMatNavCam();
    api.saveMatImage(rawImage, savePrefix + "_raw.png");

    // 2. 獲取相機參數
    Mat cameraMatrix = new Mat(3, 3, CvType.CV_64F);
    cameraMatrix.put(0, 0, api.getNavCamIntrinsics()[0]);

    Mat distCoeff = new Mat(1, 5, CvType.CV_64F);
    distCoeff.put(0, 0, api.getNavCamIntrinsics()[1]);

    // 3. 矯正影像
    Mat undistortedImage = new Mat();
    Calib3d.undistort(rawImage, undistortedImage, cameraMatrix, distCoeff);
    api.saveMatImage(undistortedImage, savePrefix + "_undistorted.png");
```

```
    // 4. 檢測 AR 標籤
    Dictionary dictionary = Aruco.getPredefinedDictionary(Aruco.DICT_5X5_250);
    List<Mat> corners = new ArrayList<>();
    Mat ids = new Mat();
    Aruco.detectMarkers(undistortedImage, dictionary, corners, ids);

    Log.i(TAG, "Processed " + savePrefix + ": found " + corners.size() + "
markers");

    return undistortedImage;
}
```

3. 錯誤處理和驗證

```
private boolean validateCameraParameters() {
    try {
        double[][] intrinsics = api.getNavCamIntrinsics();

        // 檢查是否成功獲取參數
        if (intrinsics == null || intrinsics.length != 2) {
            Log.e(TAG, "Failed to get camera intrinsics");
            return false;
        }

        // 檢查相機矩陣
        if (intrinsics[0] == null || intrinsics[0].length != 9) {
            Log.e(TAG, "Invalid camera matrix");
            return false;
        }

        // 檢查畸變係數
        if (intrinsics[1] == null || intrinsics[1].length != 5) {
            Log.e(TAG, "Invalid distortion coefficients");
            return false;
        }

        Log.i(TAG, "Camera parameters validated successfully");
        return true;

    } catch (Exception e) {
        Log.e(TAG, "Error validating camera parameters: " + e.getMessage());
        return false;
    }
}
```