

Navigation System of Group A2

Wu Chengyu 7086cmd@gmail.com

February 1, 2024

Contents

1	Introduction	3
2	Targets	3
2.1	Localization & Navigation	3
2.2	Block Recognition	3
3	Methodological Analysis	3
3.1	Basic Parameters	4
3.2	6-point Adjustment	4
3.3	3-point Localization	4
3.4	1-point Emergency Localization	5
3.5	Turn Around	5
3.6	Block Recognition	5
3.6.1	Color Recognition	5
3.6.2	Shape Recognition	6
3.7	Communicating	6
4	Algorithm Implementation	6

1 Introduction

THE map of the contest is a 2D plane with **apriltags** on it. The **apriltags** are asymmetrical and easy to recognize.

Through the camera, we can fetch all the **apriltags** in the view of the camera. Using **OpenCV**, we can get the position of the **apriltags** in the image. Then we can use the position of the **apriltags** to calculate the position of the camera.

The ROS (Robot Operating System) provides apis to get the position of the **apriltags** in the image. However, we also need the camera to recognize other blocks, e.g. the “fish” in the contest. So we deprecated the ROS and simply use Python with **OpenCV** to get the position of the **apriltags** in order to locate.

Also, we need to recognize blocks colored with red, yellow, green, and blue (size: 5cm × 5cm × 5cm). We should use **OpenCV** too, so it’s a good manner to combine these two parts together.

That’s the biggest reason why I strongly recommend navigating with **OpenCV**.

2 Targets

2.1 Localization & Navigation

- We need fetch the location of the robot (not only position, but also the orientation) in the ground coordinate system. The ground coordinate system is defined by the **apriltags**.
- We need to adjust the position, making it more accurate. The accuracy should be less than 1cm.
- We need to handle with the emergency situation. If there are less than 3 **apriltags** in the view of the camera, we should use corner of single & double **apriltags** to calculate the position of the camera.
- If there is no **apriltags** in the view of the camera, we should turn around and find the **apriltags**.

2.2 Block Recognition

- We need to recognize the color of the block. The color of the block is red, yellow, green, and blue. If necessary, we need to recognize the orange block too.
- We should get the camera matrix, the distortion coefficients. However, it’s not my task.
- We should also get the position of the block (only position) in the ground coordinate system.

3 Methodological Analysis

In general, positioning requires 6 degrees of freedom. That is to say, we need at least 6 points to calculate the accurate position of the camera. However, the 3 degrees of freedom will remain constant during the robot’s motion.

For spatial coordinates, we use the Cartesian coordinate system (x, y, z) . For the angular coordinates, we use the Euler angle (α, β, γ) . Or we can call it (roll, pitch, yaw).

The punctuation of **apriltags** is fixed, therefore, as long as we know the position of **apriltags** in the image and its punctuation, it is possible to correspond the two-dimensional image coordinates to the three-dimensional spatial coordinates. Using PnP algorithm, we can get the position of the camera.

3.1 Basic Parameters

We need the camera's internal reference and distortion coefficients to accurately confirm the conversion.

The camera's internal reference is a 3×3 matrix, which is the camera's focal length and the center of the image:

$$\text{camera_matrix} = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix}$$

The distortion coefficients are 5 parameters, which are used to correct the distortion of the image:

$$\text{distortion_coefficients} = (k_1 \quad k_2 \quad p_1 \quad p_2 \quad k_3)$$

The relevant parameters of the camera are not considered to vary excessively during the process. In other words, we can store these parameters directly at call time.

3.2 6-point Adjustment

Each location of the **apriltags** can define $\frac{1}{6}$ of the camera's position. Therefore, we need at least 6 **apriltags** to calculate the position of the camera.

$$\begin{pmatrix} x & y & z \\ \text{roll} & \text{pitch} & \text{yaw} \end{pmatrix}$$

Through the PnP algorithm provided by **OpenCV**, we can calculate the transform vector and the rotation vector of the camera easily.

After getting **tvec** and **rvec**, we can use the **Rodrigues** function to convert the rotation vector to the rotation matrix:

$$\mathbf{R_mtx} = \text{Rodrigues}(\mathbf{rvec})$$

Then, through some simple calculations, we can get the Euler angle.

3.3 3-point Localization

The **apriltags**'s center point is accurate enough to calculate the position of the camera. That is to say, through at least 3 **apriltags**, we can calculate the position of the camera through the PnP algorithm.

Knowing z , roll and pitch, we should calculate the "bias" matrix.

Define the original matrixes of rotate and transform through these elements:

$$\mathbf{R}_x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos \alpha & -\sin \alpha \\ 0 & \sin \alpha & \cos \alpha \end{pmatrix}, \mathbf{R}_z = \begin{pmatrix} \cos \gamma & -\sin \gamma & 0 \\ \sin \gamma & \cos \gamma & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad (1)$$

Then, we can get the rotate “bias” matrix:

$$\mathbf{R}_{\text{bias}} = \mathbf{R}_x \cdot \mathbf{R}_z = \begin{pmatrix} \cos \gamma & -\sin \gamma & 0 \\ \sin \gamma \cos \alpha & \cos \gamma \cos \alpha & -\sin \alpha \\ \sin \gamma \sin \alpha & \cos \gamma \sin \alpha & \cos \alpha \end{pmatrix} \quad (2)$$

Through **Rodrigues** method, we can get the rotate vector of the “bias” matrix.

Also, the transform vector is easy to get:

$$\mathbf{t}_{\text{bias}} = \begin{pmatrix} 0 \\ 0 \\ -z \end{pmatrix} \quad (3)$$

3.4 1-point Emergency Localization

We can know each corner of the **apriltags** of the map. Therefore, we can calculate the position of the camera through the PnP algorithm.

However, I strongly recommend that we should not use this method. It is not so accurate and it is easy to make mistakes.

If there is less than 3 **apriltags** in the view of the camera, we can calculate corners of the **apriltags** in the image. Then we can calculate the position of the camera through the PnP algorithm.

3.5 Turn Around

The robot can never gonna give you up, never gonna let you down. It can never gonna run around and desert you.

So, don't cry, don't say goodbye, don't tell a lie and hurt you.

If there's no **apriltags**, what the robot should do is to turn around and find the **apriltags**. There is no place without **apriltag** around the robot.

3.6 Block Recognition

3.6.1 Color Recognition

Through the HSV color space, we can easily recognize the color of the block. The HSV color space is a cylindrical coordinate system. The three dimensions represent the hue, saturation, and value, respectively.

We use the opening operation to remove the noise. That's because there are white dots on colored blocks and it is not so easy to remove. The opening operation can relatively remove the white dots, making the color recognition more accurate.

The adjustment of target range is a hard work. We should adjust the range of the color recognition to make it more accurate.

3.6.2 Shape Recognition

Recognition Due to the interference of the points in the dice, we cannot determine the outline of each dice very accurately. But the closer the distance, the higher the definition.

We use HSV color space to recognize the color of the dice. Before recognizing, we use the opening operation to remove some noise. You can see the action in the next section.

Then, we calculated the transform matrix, and use the `warpPerspective` function to transform the image:

$$\mathbf{R}_x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta \\ 0 & \sin \theta & \cos \theta \end{pmatrix}, \mathbf{R}_y = \begin{pmatrix} \cos \phi & 0 & \sin \phi \\ 0 & 1 & 0 \\ -\sin \phi & 0 & \cos \phi \end{pmatrix}, \mathbf{R}_z = \begin{pmatrix} \cos \psi & -\sin \psi & 0 \\ \sin \psi & \cos \psi & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad (4)$$

Then, we firstly dot the \mathbf{R}_x and \mathbf{R}_y matrixes, and then dot the result with \mathbf{R}_z matrix. We can get the transform matrix.

$$\mathbf{R}_{\text{transform}} = \mathbf{R}_z \cdot (\mathbf{R}_x \cdot \mathbf{R}_y) \quad (5)$$

The vertical transform vector is quite easy. Just simply resize it.

Then, we can get the full matrix through `hstack` and `vstack` method. You can see the full algorithm in next section.

Calculation The camera can only capture the two-dimensional image. So we simply use the N2 algorithm to abstract objects into particles:

$$(x \ y) = \left(\sqrt{\frac{\sum x_i^2}{n}} \quad \sqrt{\frac{\sum y_i^2}{n}} \right)$$

The reason why the arithmetic mean is not used is because I think it is not elegant enough.

3.7 Communicating

4 Algorithm Implementation

Using PnP method, we can get the location of the camera easily.