Robot Planning and its Applications

Aravind Swaminathan

October 19, 2020

Contents

1 Image Saver- genericImageListener():

To do the intrinsic calibration , the steps to be followed are by running the AR_simulator_calibration and AR_pipeline_img_saver. Through this the images are saved from the simulator which contains checkerboard . Result of this function will be intrinsic parameters. To save the image, the function in genericImageListener in the student_interface.cpp was changed acccordingly.

Implemented Function: genericImageListener
Functionality: To store the images in the config folder
Function Available in: src/student_interface.cpp
Reference for implementation: professor_interface.cpp
Result available in: All stored images can be found in camera_image_captured_gazebo folder.

2 Intrinsic calibration:

The intrinsic calibration was carried out using the tool that was provided during the lecture. There was no change done in the code. Just used the tool and got the intrinsic parameters in the xml format. And then I copied the parameters to the ”camera_params.config” available in the config folder of professor interface.
Used Tool: Calibration tool
Result available in: camera_params.config

3 Image undistortion:

Using the distortion coefficients obtained in the previous steps, I need to remove the distorted effect on the image. This is done using the opencv undistort function.
Implemented Function: imageUndistort
Functionality: To Undistort the image and obtain the distortion coefficients
Function Available in: src/extrisnicCalib.cpp
Reference for implementation: professor_interface.cpp and opencv library

4 Extrinsic Calibration:

Now after the intrinsic calibration, I did the extrinsic calibration to determine the Rotational and translational matrix. Four points will be chosen in the image plane and then these 4 points will be solved using the solvePnP interface from opencv to solve the extrinsic problem.
Implemented Function: extrinsicCalib
Functionality: To find the rotational and translational vector
Function Available in: src/extrisnicCalib.cpp
Reference for implementation: professor_interface.cpp and opencv library
Directly Copied functions: mouseCallback() and pickNPoints()

5 Perspective Projection and Unwarping:

Now to have a bird's eye view of the image, where we need to project the 3D objects in a image plane, I carried out Perspective Projection and Unwarping of image. This is again carried out the opencv interfaces. First, findPlaneTransform() has to caarried through which we get a perspective projection matrix, through which we can unwarp the image
Implemented Function: findPlaneTransform() and unwarp()
Functionality: To get a unwarped image
Function Available in: src/extrisnicCalib.cpp
Reference for implementation: professor_interface.cpp and opencv library

6 Process Map:

After calibration, we need to process map is an important function for futher navigation steps. It is decided that obstacles will be Red color with different shapes, gate as green rectangle and victims as green circles.

6.1 Obstacles detection- Red shapes:

Steps followed: RGB→ HSV→Apply red mask→ Contours→ approximate Polynomial→ Return Obstacles
Implemented Function: processObstacles()
Functionality: To get all obstacles information
Function Available in: src/processMap.cpp
Reference for implementation: professor_interface.cpp, demo code and opencv library

6.2 Gate Detection - Green Rectangle :

Steps followed: RGB→ HSV→Apply Green mask→ Contours→ approximate Polynomial→ detect 4 points→ Return Gate
Implemented Function: processGate()
Functionality: To get all gate/Destination information
Function Available in: src/processMap.cpp
Reference for implementation: professor_interface.cpp, demo code and opencv library

6.3 Victim Detection - Green Circle :

Steps followed: RGB→ HSV→Apply Green mask→ Contours→ approximate Polynomial→ detect more than 6 points→ Return Victims
Implemented Function: processVictims()
Functionality: To get all victims location
Function Available in: src/processMap.cpp
Reference for implementation: professor_interface.cpp, demo code and opencv library

6.4 Victim Id detection:

This is mainly done to detect the number of Victims, so that the priority to save the victims can be known to the planning algorithm. This involves template matching majorly and the templates were provided in lecture. I have used majorly the opencv template matching methods. But I also implemented the tesseract-ocr method, although the results were weird from them. None of the digits were recognized properly.

I found a logic in internet somewhere about the digit recognition including the orientation. To put it in simple words, each template will be rotated by 5 degrees and then they are set to calculate the score and the maximum of all of that will be be finalized as digits. With this logic, the digits are getting recognized, but there is a small issue here ass well. If I provide template for all the digits, say "0-9", then there are some mis-classifications. But if I give from digits, "1-5" , then the code works fine.
Steps followed: Detect Victims → get bounding rect → mask and filter image → change background → read templates → Resize and filter the ROI → change different orientation of ROI →match template → return Digit
Implemented Function: get_victim_id()
Functionality: To get the victim's priority
Function Available in: src/processMap.cpp
Reference for implementation: professor_interface.cpp, demo code and opencv library

7 Find Robot - blue triangle:

I directly utilized the function provided by the teaching assistant as I found that implementation was already in the best shape.
Steps followed: RGB → HSV → blue mask → Contours → Approximate polynomial → find 3 points of polynomial → Find center of triangle → Find angle between top vertex and center(Orientation)→ return state(x,y,ψ)
Implemented Function: findRobot()
Functionality: To get the robot location
Function Available in: src/findRobot.cpp

8 Load Image:

I directly utilized the function provided by the teaching assistant as I found that implementation was already in the best shape.
Implemented Function: loadImage()
Functionality: To load a image outside simulator
Function Available in: src/loadImage.cpp