ROBOSOT RACE ROBOTIC SYSTEM

CHAN YEN YEE

AZIZI ABDULLAH

Fakulti Teknologi dan Sains Maklumat, Universiti Kebangsaan Malaysia

Abstract: The world is currently experiencing a turbulent industrial revolution 4.0. Robotics are becoming increasingly popular especially in education. There are various types of robotics competitions in the world, FIRA RoboWorld Cup is one of these robotics competitions that involved college-level participants. This project focuses on the development of a robotics system for ROS Turtlebot3 to overcome the Robosot Race challenge in the FIRA competition. During the game, the robot will be given 12 colored (red, blue, yellow) balls that need to be placed in the goal which corresponds to color of ball. Robots need to track the ball, plan their movements to take the ball and put them into the correct goal meanwhile avoid obstacles throughout the journey. Therefore, this is a ROS-based robotics system that able to perform functions such as object detection, path planning and obstacle avoidance. SLAM methods such as AMCL and Gmapping, path searching algorithm, Dijkstra and A-star are involved as path planning algorithms in robot navigation program. The accuracy of the Robosot Race Robot System has exceeded 91.67% and range of the average time to settle a ball is between 49 seconds until 1 minute 30 seconds. The system performance prove that the programs developed able to detect coloured ball, navigate the robot to approach the targeted ball and eventually push the ball into the matching goal space.

Keywords:Robosot Race, ball detection, robot navigation

INTRODUCTION

This project is about to develop a robotic system which specialized to encounter the Robosot Race Challenge in FIRA Roboworld Cup. Robosot Race is a small-sized soccer robot competition for wheeled platform. After the game starts, the robot must search for the color ball in the field and push the ball into the corresponding color area to get the score. The champion is who got the highest sum of the score. TurtleBot3 Burger manufactured by ROBOTIS company is used to compete in the Robosot Race game (FIRA, 2019a).

There are specific rules in the game which set by FIRA such as number of balls, number of colors, number of goal spaces, time given and etc. According to the latest rule, there are 12 balls to be used with 3 different colors, which are red, blue and yellow and therefore it will be 3 goal spaces. The time given to place as many balls as possible within each goal location is about 2 minutes (FIRA, 2019b).

There are a lot of ball detecting programs but none of them is developed to be used on Turtlebot3 and integrated with navigation function. Besides that, there is no any similar decision-making or logical control program which able to lead the robot for what do next if encounter a certain

condition such as choose the correct goal, look again if the ball slipped away, move to goal if ball is secured and etc.

The developed robotic system has to be able to (1) detects the balls through camera's and sensor's data, (2) determine the balls' coordinates and goals and (3) do path planning and obstacles avoidance autonomously in order to bring the detected ball to the corresponding goal space.

In order to result the mentioned functions, a few image processing techniques such as color filtering and Circle Hough Transform (CHT) can be used to differentiate ball's colour and detect ball's coordinates. SLAM methods such as AMCL and Gmapping will combined with path planning algorithms in ROS Navigation Stack in order to do robot navigation since Turtlebot3 Burger is a ROS-based robot. A logical control or decision-making program will be designed to integrate those algorithms and techniques.

Among the 6 tests carried out (3 single color tests, 3 complete tests), the developed system gave an average accuracy within 91.67% (complete tests)-100% (single color tests). The accuracy had achieved the project's objectives and also fulfilled the users' and system's requirements. The system can be used on real machine in order to compete in Robosot Race Challenge.

METHODS

The robotic system is developed on ROS Kinetic Kame which run on Ubuntu 16.04 due to Turtlebot3 Burger is a ROS-based robot. There are 3 steps in building this system:

1. Construct a Gazebo simulation environment (playing field)
2. Mapping and determine goal spaces coordinates.
3. Design the ball detection function (Ball Detector).
4. Build decision-making program (Commander).

Step 1. Construct Gazebo Simulation Environment

Gazebo is an official simulation platform which enable developed ROS robotic system to being tested. To achieve ROS integration with stand-alone Gazebo, a set of ROS packages named "gazebo_ros_pkgs" provides wrappers around the stand-alone Gazebo. They provide the necessary interfaces to simulate a robot in Gazebo using ROS messages, services and dynamic reconfigure *(GAZEBO, 2014)*. There are 2 important components in Gazebo which are "model" and "world". "Model" is an object that can be render in "world" whereas "world" is the environment which contain "model".

In this project, there are 3 models: balls, goal spaces and Turtlebot3. Gazebo provides a variety of model templates and meshes, users can directly use the desired objects and adjust their position, size, color or other physical features.

The figures below show the blue ball (Figure 1), goal spaces (Figure 2) and also the Turtlebot3 Burger (Figure 3) which created by Gazebo GUI.

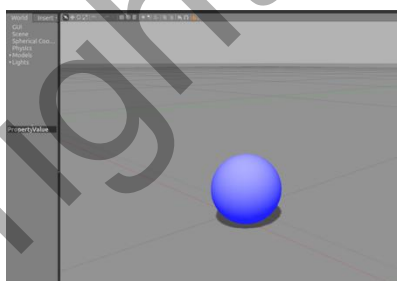

Figure 1. Blue ball model in Gazebo.

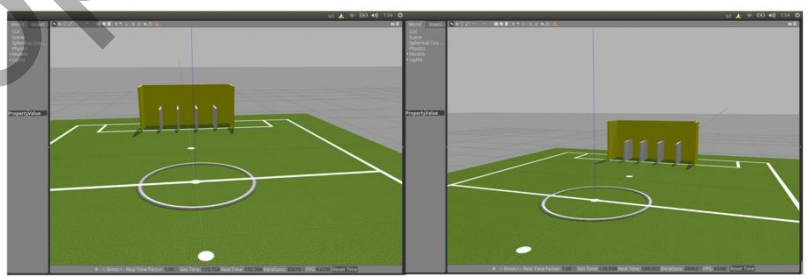

Figure 2. Goal spaces model in Gazebo.

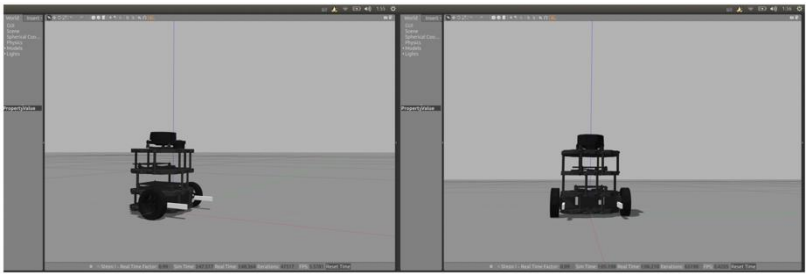

Figure 3. Turtlebot3 Burger model in Gazebo.

After done the object modelling, save the model data into a ROS package. A launch file can be designed in order to integrated both "world" and "model" into a single script so users able to load the simulation environment through $roslaunch command. Furthermore, users can individuate the number of balls and their color (red, blue, yellow) in the launch file. Figure 4 below show the complete Robosot Race playing field which after launced the $roslaunch command.

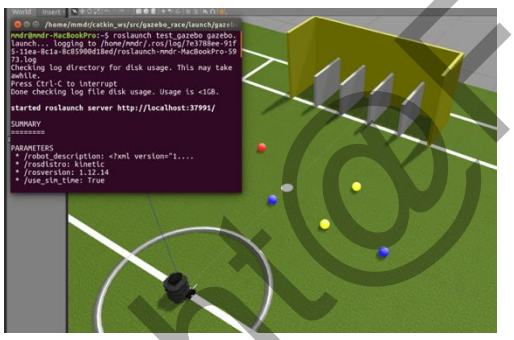

Figure 4. Complete Robosot Race playing field in Gazebo.

Step 2: Mapping and determine goal spaces coordinates.

Mapping is the process of static map construction, the mapping methods which used in this project is called Gmapping, which is one of the popular laser-based SLAM methods. Robot can only do localization and navigation when there is input of map data. The LiDAR sensor on Turtlebot3 Burger will scan its surrounding when users manually control its movement. Figure 5 below show the map of Robosot Race playing field which resulted from Gmapping.

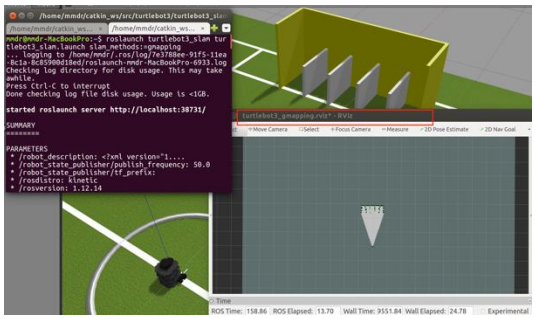

Figure 5. Map of Robosot Race playing field in Gazebo.

Furthermore, in order to determine coordinates of goal spaces, a useful ROS tool, Rviz is used. There is a function known as "PublishPoint" in Rviz which can print the coordinates of desired point that users clicked on constructed map. Therefore, the coordinates of every goal spaces can be identified and used for robot navigation and the Commander program.

Step 3. Design Ball Detector

Ball detection is one of the core functions of this system. In order to differentiate balls' color and their coordinates from the images took by Turtlebot3 Burger, 2 image processing techniques are used to develop Ball Detector program: color filtering and Circle Hough Transform. Furthermore, OpenCV library is used to perform the mentioned techniques.

a. Color Filtering

Color filtering is applied to clean out irrelevant color in the images based on the range of HSV (Hue, Saturation, Value) value which define by users.

By defining a list of boundaries in the HSV color space, where each entry in the list is a tuple with two values: a list of lower limits and a list of upper limits (Rosebrock, 2014). For instance, red color has "hue" value which within 0-28, "saturation" value within 10-255 and also "value" among 0-255, so it will be a set value in form of [0,10,0], [28,255,255].

In this project, RGB images which took by Turtlebot3 Burger will be converted into HSV format in order to perform color filtering. When the upper and lower boundaries of a color is found, "cv2.inRange(upper_limit, lower_limit)" function in OpenCV will help to filter out the color which is out of the range.

b. Circle Hough Transform

Hough Transform is a features detection algorithm invented by Paul Hough in 1962 which is widely used in robot soccer games *(Wojcikiewicz, 2008)*. It is very useful in detecting any parametric curves such as lines or conics. The advantage of Hough Transform is that it is relatively unaffected by gaps in curves and noise in images. Given a set of edge points and a parametric equation which describe the desired shape, Hough Transform will use a "voting" mechanism to find the line(s) which best fit the equation.

Circle Hough Transform (CHT) is an extension of Standard Hough algorithm. In this project, CHT will be applied on the processed image which resulted from color filtering step. There are a few steps that needed to be carried out before performing CHT:

1. Convert the RGB images into grayscale.

   The colored image need to be converted into grayscale in order to reduce irrelevant information. The conversion of an RGB pixel can be done by the equation below:

$$grayscale = \frac{R + G + B}{3}$$

   This is the standard formula for RGB-grayscale conversion. There are other combinations for different ration of RGB value that can be choose. For this project standard conversion is fine.

2. Canny Edge Detection

This is a step to extract edges in the image. Canny Operator selected because it practiced double thresholding method so it can resist the noise in image. To further reduce the noise in image, Gaussian blur will be applied in order to perform noise reduction. Gaussian blur is a type of image-blurring filters that uses a Gaussian function for calculating the transformation to apply to each pixel in the image. The formula of a Gaussian function, G(x) in one dimension is as follow *(Shapiro, 2001)*:

$$G(x) = \frac{1}{\sqrt{2\sigma^2\,\pi}}\; e^{-\frac{x^2}{2\sigma^2}}$$

Canny operator uses two 3×3 kernels which are convolved with the original image to calculate approximations of the derivatives – one for horizontal changes, and one for vertical *(R. Fisher, 2003)*. Let *A* as the source image, and *Gx* and *Gy* are two images which at each point contain the vertical and horizontal derivative approximations respectively, the computations are as follows:

$$\mathrm{G_x} = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & +2 \end{bmatrix} * A \qquad \mathrm{G_y} = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * \mathrm{A}$$

where * here denotes the 2-dimensional signal processing convolution operation.

The x-coordinate is defined here as increasing in the left to right direction (horizontally), and the y-coordinate is defined as increasing in the up to down direction (vertically). At each point in the image, the resulting gradient approximations can be combined to give the gradient magnitude, using:

$$G = \sqrt{{G_x}^2 + {G_y}^2}$$

By the gradient, G which get from previous step, the gradient's direction can be calculated based on:

$$\Theta = atan\left(\frac{G_y}{G_x}\right)$$

Following by non-maxima suppression on every pixel in image in horizontal as well as vertical direction to filter out the pixels which is not possible to be a part of edges. Double thresholding applied onto the pixels left in order to get strong pixel (pixel with intensity higher than threshold) and weak pixel (pixel which connected to strong pixel). The last step is edge tracking with Hysteresis, aims to transform weak pixel into strong pixel in order to form a continuous edge.

CHT can be performed on the image resulted from the 2 steps mentioned above, it will find the contour of circle in the edges set. A circle is represented mathematically as:

$$(x - x_{center})^2 + (y - y_{center})^2 = r^2$$

where (x_{center}, y_{center}) is the center of circle and *r* is the radius of circle.

From equation above, there are 3 parameters so a 3D accumulator for Hough Transform is needed, which would be highly ineffective. So OpenCV library uses more trickier method, Hough Gradient Method which uses the gradient information of edges. A function in OpenCV library which known as cv2.HoughCircles() can beused to detect balls' contour from image and determine their x, y coordinates as well as the radius. There are a few important parameters of desired circle which needed to be determine *(OpenCV, 2011)*:

1. image: 8-bit, single-channel, grayscale input image.
2. circles: Output vector of found circles. Each vector is encoded as a 3-element floating-point vector (x,y,radius).
3. method – Detection method to use. Currently, the only implemented method is CV_HOUGH_GRADIENT.
4. dp – Inverse ratio of the accumulator resolution to the image resolution. For example, if dp=1, the accumulator has the same resolution as the input image. If dp=2, the accumulator has half as big width and height.
5. minDist – Minimum distance between the centers of the detected circles. If the parameter is too small, multiple neighbouring circles may be falsely detected in addition to a true one. If it is too large, some circles may be missed.
6. param1 – First method-specific parameter. In case of CV_HOUGH_GRADIENT , it is the higher threshold of the two passed to the Canny() edge detector (the lower one is twice smaller).
7. param2 – Second method-specific parameter. In case of CV_HOUGH_GRADIENT, it is the accumulator threshold for the circle centers at the detection stage. The smaller it is, there are more false circles may be detected. Circles, corresponding to the larger accumulator values, will be returned first.
8. minRadius – Minimum circle radius.
9. maxRadius – Maximum circle radius.

Figure 5 below show the resulted image after applied color filtering technique (left) and CHT (right).

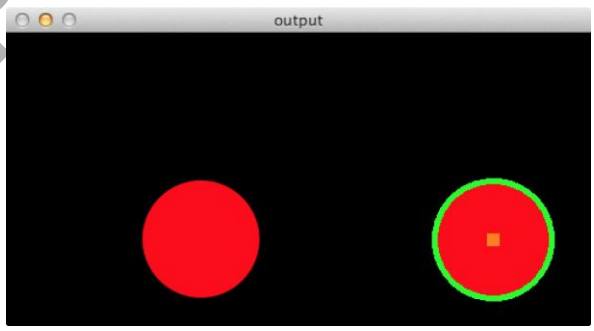

Figure 5. Output of Ball Detector.

Step 4. Build Decision-making Program

This decision-making (Commander) program is written using Python and "rospy" library. It acts as a robot commander which able to lead Turtlebot3 Burger to perform a series of actions in order to encounter the Robosot Race Challenge. Therefore, the concept of "state transition" is implemented in order to enable the robot will take different action when the conditions set fulfilled. Besides that, the goals' coordinates which determine at step 2 above can be included for goal matching in "State 4" in this Commander program.

There are 7 states designed in this program:

1. State 0: Rotate and scan the surrounding to look for ball.
   During this state, Turtlebot3 Burger will turn around and send image data to the Ball Detector for ball searching. If there is at least one ball detected, the State will be increase by 1.

2. State 1: Define the nearest ball among all the balls detected.
   From the ball(s) detected, algorithm will choose the ball which located closest with the Turtlebot3 Burger based on the coordinates data resulted from Ball Detector. Besides that, the color of ball will be determine as well. State will be transit into 2.

3. State 2: Move towards target ball.
   Robot will start to move towards the direction of target ball during this state. When the program detected that the robot has come close to the target ball, value of State will be change to 3. If the target ball loses or slipped away, State will be return to 0.

4. State 3: Approach target ball.
   During this state robot will be very close to the target ball, so program designed to lead robot to take the ball and move for a certain distance in order to secure the ball within its flippers. After robot finish the movement, State will be increased to 4 after the robot

5. State 4: Move towards goal space.
   Program will match the goal based on the target ball's color and send the corresponding coordinates to ROS Navigation Stack. ROS Navigation Stack will help to generate a path that is safe and accurate which able to make robot reach the correct goal space. The path planning algorithm used to generate global path are Dijkstra’s and A-star whereas the local path (command velocity) is generated by Dynamic Window Approach (DWA). After the robot reach the goal space with target ball, State will be change to 5.

6. State 5: Push ball into goal space.
   At this State robot will be move forward with target ball in order to push the ball into goal space. State will change to 6 after the ball is fully pushed inside the goal.

7. State 6: Reverse from goal space.
   This is the last step of solving a single ball. Robot will be reverse from goal space after successfully push in the ball into goal space and turn to face the direction of playing filed. State will be return to 0 to start a new cycle of ball detection.

FINDINGS AND ARGUMENTS

The number of color balls involved in each test is listed in the table below:

Test	Number of balls			Total number of balls
	Red	Blue	Yellow	
1	5	0	0	5
2	0	5	0	5
3	0	0	5	5
4	4	4	4	12

Table 1. Number of color balls in each test.

Based on table 1, there are 2 types of test: single color test (test 1-3) and complete test (test 4). The single color tests which are each of test 1 to 3 will be carried out once whereas the complete test (test 4a-4c) will be repeated for 3 times in order to guarantee the consistency of system.
The objective of running test 1 to 3 is to evaluate the system performance on detect single color balls whereas the complete test (test 4) is to find out the overall system performance.

The performance of system for each test is evaluated by the accuracy which calculated by the equation below:

$$Accuracy = \frac{Number\ of\ balls\ which\ correctly\ placed\ into\ goal}{Total\ number\ of\ balls}$$

Since test 4 in Table 1 will be repeated 3 times, the average accuracy needed to be calculated as following equation:

$$Average\ Accuracy = \frac{\sum Accuracy\ of\ each\ test}{3}$$

Besides that, since there is a time limit in the Robosot Race game, the average time taken to solve each ball will be calculated as follow:

$$Average\ time\ taken\ to\ solve\ a\ ball = \frac{Total\ time\ taken\ to\ solve\ all\ balls}{Total\ number\ of\ balls}$$

Among the 6 tests, system performance is shown in the table below:

Test	Number of balls which correctly placed into goal	Accuracy (%)	Total time taken to solve all balls	Average time taken to solve a ball
1	5	100	4minutes 5seconds	49seconds
2	5	100	6minutes 39seconds	1minute 20seconds
3	5	100	8minutes	1minute

				36seconds
4a	11	91.67	13minutes 26seconds	1minute 13seconds
4b	11	91.67	12minutes 29seconds	1minute 8seconds
4c	11	91.67	16minutes 35seconds	1minute 30seconds

Table 2. Test result.

From the Table 2 above, it can be observed that among 6 tests carried out, the Robosot Race Robotic System gave an accuracy within 91.67% - 100%, which means that only at most one ball is not being successfully located into corresponding goal space. The average time taken to solve a ball varies from 49 seconds - 1 minute 36 seconds. The difference in time taken is caused by the order in which robot solves the colored ball. If the robot detects the yellow ball at first, the time taken will be longer as the distance is the longest for robot to travel to yellow goal space. This means robot will need more time to solve all balls and finally affect the overall average time taken to solve a ball.

CONCLUSIONS

In conclusion, the system performance is satisfactory since it able to achieve at least 91.67% of accuracy. The time taken to solve a single ball is also fulfilled the non-functional system requirement, which is solve a single ball in 2 minutes.

The accuracy proved that the developed system able to detect colored ball and placed them into correct goal spaces. The ball detector and commander program able to collaborate and lead the Turtlebot3 Burger to encounter challenges in Robosot Race game.

However, the immature of obstacle avoidance algorithm caused the robot sometimes didn't notice the ball(s) on the way it moved. This scenario happened when robot is carrying a target ball towards goal and eventually miss located the ball which on the way into wrong goal together with the target ball. The figure 6 below will give an illustration to the scenario.

Figure 6. Example scenario of system's error.

From the figure 6 above, it can be observed that robot didn't notice the yellow ball when it is carrying red ball. This finally caused the yellow ball to be placed into red goal together with the targeted red ball.

Therefore, there is still improvements that can be done in order to enhance the consistency and accuracy of robot navigation algorithm. Since it is running in a controllable environment at the moment, so the system performance may vary when the system is used with real Turtlebot3 Burger.

REFERENCES

FIRA. (2019). FIRA RoboSot Race Competition Rules and Regulation for 2019. Retrieved from FIRA Official Website: https://bit.ly/2xlwpvj

GAZEBO. (2014). Tutorial: ROS integration overview. Retrieved from GAZEBO: http://gazebosim.org/tutorials?tut=ros_overview

Rosebrock, A. (2014). OpenCV and Python Color Detection. Retrieved from PyImageSearch : https://www.pyimagesearch.com/2014/08/04/opencv-python-color-detection/

Wojcikiewicz, W. (2008). Hough Transform Line Detection in Robot Soccer. Heriot Watt University.

R. Fisher, S. P. (2003). Sobel Edge Detector. Retrieved from Sobel Edge Detector: http://homepages.inf.ed.ac.uk/rbf/HIPR2/sobel.htm

Shapiro, L. G. (2001). Computer Vision. Prentice Hall, 137-150.

OpenCV. (2011). Feature Detection. Retrieved from OpenCV: https://docs.opencv.org/2.4/modules/imgproc/doc/feature_detection.html?highlight=houghcircles