COMP3631
Intelligent Systems and Robotics
Group 14 Project Written Report

Team: Isaiah Fergile-Leybourne, Yuvraj Mahida, Usman Sherdil, Thomas Marshall

Introduction

[Yuvraj Mahida]
For this project, we have designed and implemented a unique solution that controls a simulated Turtlebot to find and identify a Cluedo character in any Turtlebot Gazebo environment as requested by you, the client. In this report, we have explained our design process and justifications for implementation with careful consideration for program efficiency, robustness and overall performance. Our solution aims to successfully perform each task accurately in a range of environments and has been tested extensively on eight different worlds. Lastly, this document provides a performance evaluation that displays concrete examples of our working solution.

Design

[Isaiah Fergile-Leybourne]
A flow diagram was created after the first meeting. The flow diagram stays true to the specification outlined in the 'Project Description' document and aims to reinterpret it in a procedural manner.

After the program starts, the turtlebot must navigate the main room. After a circle is found it must be evaluated to see if it is red. If the circle is red, then the turtlebot must navigate to the next entrance and evaluate the next circle.

In either instance, if the circle is green the turtlebot needs to check if the circle is fully contained in the image and reposition itself if the circle is obscured or partially in view. Once the turtlebot has the entire green circle in its field of view a picture can be taken and saved.

The turtlebot can then move into the respective room. Once inside of this room, the turtlebot must navigate it looking for the cluedo character. Since there are multiple objects in this room, the cluedo character can be distinguished from them by colour and the rectangle that surrounds it. If a rectangle is found, it then uses colour to check if a character is inside of the rectangle.

Based on which colour is seen the character can be identified and a text file can be written with the characters name. Like the green circle, the character should be in full view of the camera. If it is not then the turtlebot must reposition itself. Once the character is in full view a picture can be taken and the program can terminate.

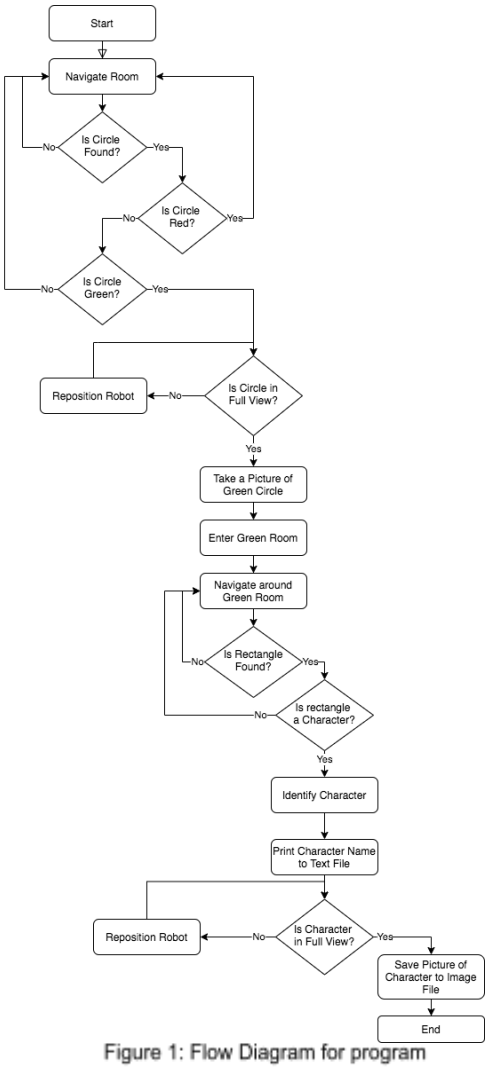


Figure 1: Flow Diagram for program

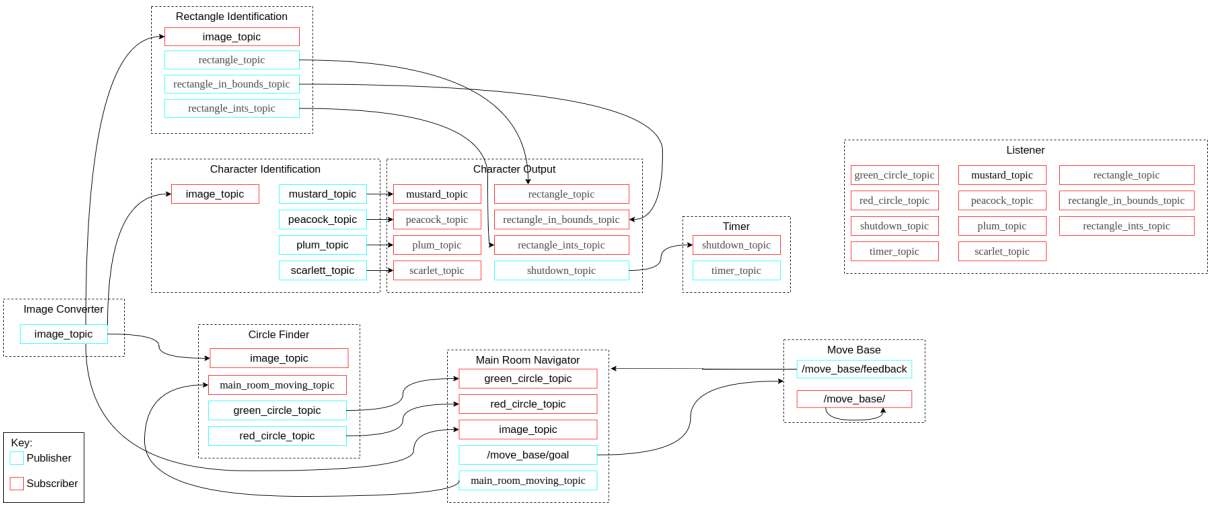


Figure 2: Node Diagram for program

To keep the program efficient each node aimed to use the minimal amount of memory necessary to operate. Libraries were only imported as needed and code was not reimplemented but instead each node listened to the relevant node's publishers.
Also the task is fairly similar to the tasks presented in lab sessions 1-4, a different and novel approach has been taken to identifying characters that uses both colour and shape (making use of the Hough Circle Transform). Additionally, how each node is interconnected has been well thought out.

[Usman Sherdil]
The method was decided upon due to its robustness. The chosen method ensures the robot always moves to the entrance of room one and then based upon what it sees there it would perform the necessary actions. So, the method would work, if either one of the two rooms are green, it will either move the robot to the centre of the first room if it is green or it will navigate the robot to the second room if it sees the circle is red for room one. This allows us to form the conclusion where the method will work successfully within any world and would always ensure the robot takes a picture at the entrance of the correct room as well as navigating to the correct room.

Implementation

Entering the Green Room

[Isaiah Fergile-Leybourne]
The program obtains the coordinates for the entrance and the center of each room from the yaml ‘input points’ file. The turtle bot then uses the move base node to send itself to the x and y coordinates of room one. To calculate the angle that the turtlebot should be facing, the move base node’s theta is set to be equal to the angle of the vector between the room’s entrance and the room’s center. This ensures that the turtlebot is facing the correct 180°.

If the circle is not in the view of the turtle the turtlebot will rotate, at most 360° clockwise, until a circle is found. If the circle is in view immediately upon reaching the room’s entrance, then the turtlebot will rotate in either direction to get the circle into the center of its view.

The circle is then evaluated. If the circle is green (as determined by the circle_finder node), then the turtlebot will enter that room and place itself in the center of it.

If the circle is red, then the turtlebot will move to the next entrance following the same procedure as the first entrance. The second entrance’s circle will then be evaluated (given that the first circle was red we would expect it to be green). If the circle is green then the turtlebot will move to the second room’s center. If the green circle is still not discovered then the turtlebot will enter the first room.

Circle Detection

[Isaiah Fergile-Leybourne]
The ‘Circle_Finder’ node is activated once the robot reaches either one of the two entrances for each room. As stated before, both the room’s center and entrance points are used to calculate the rotation of the robot upon reaching the room’s entrance point, which means the robot is facing the correct 180° with the circle either in view or just outside of it’s line of sight almost in view (n.b. the horizontal field of view of the turtlebot is 62.2°).

The ‘Image_Converter’ node sends the image topic to the ‘Circle_Finder’ node in the form of an img_msg. This message is then converted to a cv2 image. The cvtColor function is used to change the image from BGR colour space to HSV. Two masks are generated by filtering out all colours except those that fall into the predefined red and green range. Each mask is then applied individually to the image to create two new images - one with only pixels that fall within the green range and another with pixels that only fall within the red range.

The masked images are then used as parameters, respectively, for two different functions - ‘findGreenCircle()’ and ‘findRedCircle()’. These two functions try to process the image by converting the image to grayscale, applying canny edge detection, and then blurring the returning edges. A Hough Circle Transform (Yuen et al., 1990) is then applied to the blurred image so that any circles within the image can be recognised. If a circle is identified then the relevant topic (red_circle_topic or green_circle_topic) publishes true, otherwise the topics publish false.

Navigating the Green Room

[Tom Marshall]
The ‘navigating_green_room’ node subscribes to the main room navigation, rectangle identification (discussed in the next section of report) and the turtlebot’s bumper sensor. The node will only start execution after the main_room_navigation finishes execution and informs the node that it is now in the centre of the green room. Once execution begins, the robot will rotate, stopping once it either detects the cluedo character, or until it performs a full rotation.
The node also contains a publisher to mobile_base’s velocity and the turtle_bot_main_room _moving_topic. The second publisher is used to prevent CircleFinder from publishing Twist messages, which interfere with the execution of this node.

If the character is seen during rotation. A secondary rotation function is used to continue rotating until the image is no longer seen, then perform a counter-rotation to centre the character. The function then uses a Twist message to move a small distance towards the portrait. This process is repeated until the bot is in an appropriate position to take the image of the cluedo character.

In the case that the character is not visible from the starting position, python’s random library is used to determine an angle (0-359). The turtle performs a rotation of this angle and takes a step in that direction. If the turtle collides with an object or wall, it moves back to its previous location and performs another random movement.
After each random movement, the turtle performs the original full rotation, looking for the character, and follows the execution path accordingly.

Rectangle Identification

[Isaiah Fergile-Leybourne]
The ‘Rectangle_Identification’ node subscribes to the image converter node. The image message that is received is converted to a CV image and then masked so that only values between (80,0, 0) and (180, 48.25,25.5) in HSV space are present. This reliably returns most pixels in the rectangular border of the cluedo character without including pixels from the surrounding environment in the test worlds.

If more than 50 pixels are found an rectangle is said to be found and it is defined as ((min_in_y, min_in_x), (max_in_y, max_in_x)) where each variable is the min or max of the x or y value of all pixels that occur in the masked image. The two points represent the upper left and lower right corners of the bounding rectangle.

If we are viewing the image from an angle then the rectangle will be skewed forming a parallelogram. The method above takes the bounding rectangle of the parallelogram which is enough for our needs since we are only interested in if a rectangle exists and how close it is to the edge.

Character Identification

[Isaiah Fergile-Leybourne]

The image from the image_converter node was masked four times to create four separate images. The mask for each image was based on the predominant colour for each of the cluedo characters. For each masked image, the number of non-black pixels were counted and if they were above a threshold of 300 pixels then the character was defined as seen. Based on which masked image returned more than 300 non-black pixels the character could be identified: yellow for Colonel Mustard, blue for Mrs Peacock, purple for Professor Plum, and red for Miss Scarlett.

For an image of the character to be captured and their name to be written to a text file, both the character and the bounded rectangle must be present. This gives two layers of verification that the image actually contains a character and not just a lone rectangle or coloured obstacle.

Timer

[Isaiah Fergile-Leybourne]
A timer was introduced so that the duration of time that passed from starting the program could be measured. This was then used for further debugging and testing to indicate how quickly the overall implementation and any fixes performed.

Launch File

[Isaiah Fergile-Leybourne]
A launch file was created so that all files could be run simultaneously using one command. It promotes ease of use of the application.

Git

[Isaiah Fergile-Leybourne]
The version control software Git was used throughout the entirety of this project. Git allowed us to manage the project as a whole and to integrate our work together. In total, 26 issues were created. Initially, these issues were based on how we decided to subdivide our problem. Additional issues were also created to address problems that arose during the development of the project. Using Git, we could also work on separate issues simultaneously without interfering with each other individual's workflow. Each member created a branch that addressed specific issues assigned to them for that week, then at the end of the week a merge request was created for each complete branch so that they could be merged into the main working branch.

Demonstration

[Isaiah Fergile-Leybourne]
A video of our program executing can be found online at https://youtu.be/Dn7E6FPT1tU.

Testing

Creation of Additional World Files

[Isaiah Fergile-Leybourne]

In total eight worlds were used to test the project. Four different world types were created with the character positions, entrance locations, entrance circle sizes, and obstacles positions all varied.

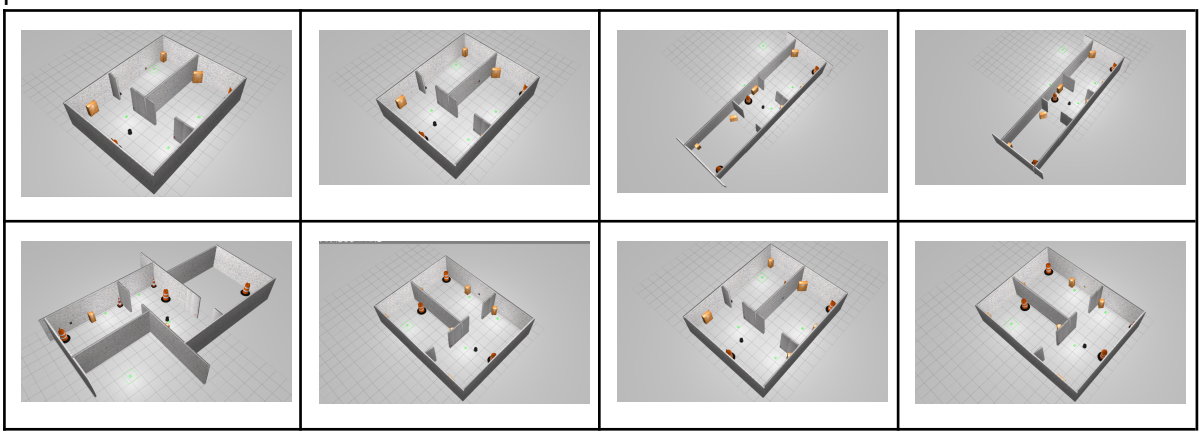
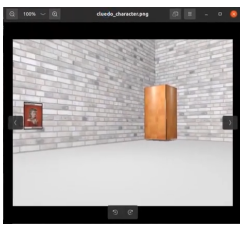
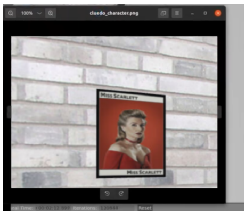
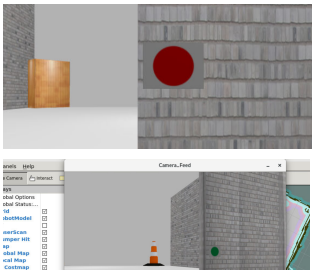
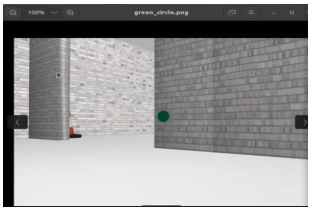
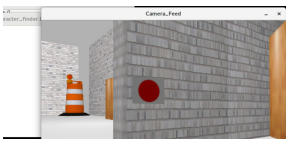
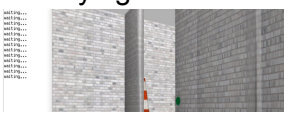


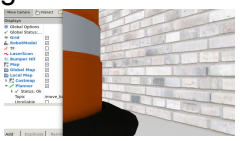
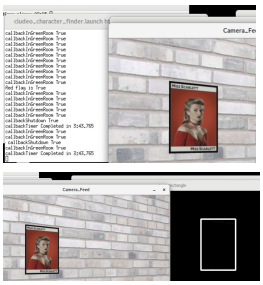
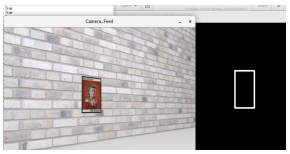
Figure 3: Eight Test Worlds

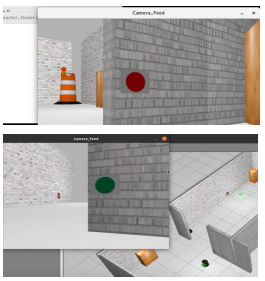
Results

[Yuvraj Mahida]
With the creation of additional world files with vast variation of parameters, testing of functionality and robustness of all aspects of the solution has been carried out. This section outlines the conducted testing and analysis of performance with concrete evidence to prove this.

Test Description	Test Outcome (Map/Environment 1 to 8)	Successful /Unsuccessful	Performance Evaluation	Evidence
Does the program fail to correctly identify the cluedo character?	Cluedo character has been correctly identified in all worlds except world 4 where the camera has not registered the character. This has been addressed by modifying the character colour thresholds.	Successful	To test the performance of this module, we created project7.world where all characters exist. The test outcome was that each character was identified correctly 100% of the time.	 callback[CluedoCharacter] successful returns true for correct character in all cases
Does the program fail to capture an image of the cluedo character?	The program successfully captures a snapshot of the cluedo character in all worlds. In each case where the character has been visible, the turtlebot performs a counter-rotation to have a full view of the character.	Successful	In terms of performance, the cluedo character image is captured correctly in all new environments. Generally, the image is captured entirely and saved to a png file. In some cases, performance can be improved by using angle and distance adjustment to centre	

	The counter-rotation in each case has positioned the turtlebot where the entire cluedo character is visible. In the case where the character is not visible from the starting position in the green room, the turtlebot rotates by an undetermined angle and moves to a near position until the character has been located. The snapshot is then taken successfully.		the camera feed.	 Figure shows correct image capturing of cluedo character to png file.
Does the program correctly locate the circles on room entrances?	Program correctly navigates the main room so the camera recognises the red and green circles on room entrances. The turtlebot performs the correct actions on locating the circles on each world.	Successful	In terms of performance, we have created maps where the size and positions of the green and red circles vary. The program identifies both circles and the correct flags are determined to be true.	
Does the program fail to capture image of the green circle	The program correctly locates the green circle and captures a snapshot of the camera image of the green circle correctly in all 8 worlds. The green circle is completely contained within the image saved in all cases.	Successful	In terms of performance, the turtlebot positions in view of the green circle before capturing the image. In all new environments, the green circle is captured entirely in feed but in some cases, the angle of the capture could be improved.	 Figure shows successful capture of green circle saved to png file
Does the program locate the Cluedo character efficiently and in a timely manner?	In all worlds the program accurately navigates the turtlebot to the green room without colliding with any obstacles or entering the red room beforehand. Once the turtlebot enters the centre of the green room, it rotates to locate the cluedo character. In most cases, the cluedo character was identified on initial rotation which proves the efficiency of the solution. In other cases, the robot navigates towards an undetermined	Successful	To test the performance of the module, the turtlebot has been localised using the roslaunch keyboard_teleop command to a position where an obstacle restricts view of the turtlebot camera in the green room (worst case). In these cases, the turtlebot rotates to a certain angle, if the obstacle is collided with, the turtlebot returns back to the starting position, where the process is repeated using a different angle which moves the turtlebot in view of the cluedo character. The design of the code is efficient as after one full	 Figure shows turtlebot identifying red circle  Figure shows turtlebot searching for green circle  Figure shows turtlebot has moved to centre of

	angle.		rotation, the angle changes, meaning no unnecessary rotations are performed again. Additionally, the program accounts for any occurring collisions with obstacles and can manage obstacles as required. Therefore the module performs the task effectively in a robust but timely manner.	green room with flag in greenroom set to True  Figure shows turtlebot camera feed where the turtlebot is localised facing a corner between two obstacles (grey wall and cone) with no view of character  Figure shows cluedo character identified and task completed in 3:43:765 < 5 minutes which includes time to localise robot.
Does the program accurately detect a rectangle representi ng the cluedo character?	The program correctly publishes the rectangle is true when a rectangle is detected as required by our design specification. Thresholds have been adjusted to reduce noise and to ensure background rectangles in the walls are not detected.	Successful	This module performs exactly as expected. When the robot rotates in the green room, the rectangle of the cluedo character is detected precisely from any range within the room..	 Figure shows rectangle identification identifies the cluedo character in a range of positions in different environments.
Does the program fail to navigate the turtlebot into the green room?	The robot enters the green room in all worlds except world 4 where the green circle was not captured. And the red circle is captured after further navigation through the map. After debugging, in all cases, the robot enters the correct room. This has been addressed and the input_points file has been updated. The test now passed for all 8	Successful	To test performance, we have created environments where the main room size and number of entrances varies. The program performs accurately when the red circle and green circle are switched. The time taken to navigate the main room and reach the entrance of the green room has been calculated to be on average 96 seconds for the 8 new environments tested	 Figure shows callbackInGreenRoom flag set to True  Figure shows the location of the turtlebot at the centre of the green room

	worlds			where the Cluedo character is placed.
Does the program accurately and efficiently locate the red and green circles?	The program navigates the robot in the main room to locate the green and red circles in a timely manner. In all tests, the robot did not navigate into the room with the red circle on the entrance.	Successful	In terms of performance, the program locates the red circle, and navigates around the room to find a next entrance. The turtlebot navigates around the main room until a green circle is detected. In all worlds the robot repositions to ensure the green circle is in full view as required.The robot searches for and locates the green circle in a timely manner.	 Figure shows main navigation effectively identifies all circles on room at any position on room entrance
Does the timer calculate the task time correctly?	The program accurately measures the time taken from executing the launch file to correctly identifying the cluedo character. The 'I can see it' flag determines when the cluedo character has been seen and identified. When the correct cluedo character name is matched to the character, the program displays the time taken to complete all tasks.	Successful	In terms of efficiency, the program completes the task accurately within the desired 5 minute time setting. There was one instance where the program took 5:43:765 which was over the maximum time limit. This was due to the mapping of the world being incorrect and also due to the robot being localised to the furthest point from the red circle and required a second pass of the room to identify the red circle. We have prioritised accuracy of the task first meaning will not enter the red room and cost us penalties. An up to date model of the room will be created using 3D sensors to resolve this.	Timer is accurate and starts when the cluedo_character_finder launch file is roslaunched, and ends precisely when the program identifies the correct cluedo character.  Figure shows timer function works as expected.
Does the integrated launch perform all tasks accurately and efficiently	The launch file has been configured to launch programs simultaneously. The launch file works as expected	Successful	The launch file successfully launches the files that are required to perform the task. In all worlds, the task has been completed efficiently	 Figure shows when testing of our solution in the example world provided, all tasks were completed in 39.29 seconds which proves the efficiency of our program

Evaluation

[Yuvraj Mahida]

Strengths

1. Robustness of solution - The solution performs accurately in the example environment and in a total of 8 new environments.
2. Modularity design and implementation - The main task has been divided into specific modules which perform an individual task. This allows for a more effective and rigorous testing process. Additionally, adding a subscriber for each node and outputting the required information enables other parts of the solution to listen/access the information of another relevant node's publisher without replicating functionality. For example, the navigating module subscribes to the circle_finder_node as this efficiently checks if a circle exists in the camera feed and determines whether the flags for red and green circles signal true or false. As described, the behaviour resulting from this action has been developed in a separate module.
3. Use of planning and search methods - As many modules can be implemented using the methods and knowledge acquired from the labs, we evaluated the efficiency of all methods during the design stage and creatively implemented ways to improve these. For instance, Isaiah was able to creatively implement rectangle_identfication to work efficiently from skewed angles, which has reduced the time required for navigation in many test cases.

Weaknesses

1. One area of improvement discussed during the testing process is navigation of the green room is implemented using random angle generation. Although this works efficiently for the world size provided, more efficient methods are available. We can explore more intelligent path planning techniques to determine the next movement of the robot based on previously explored areas or the information acquired from the camera feed, including registering locations of items in the map.
2. Therefore one scenario the solution may not be as efficient and will take longer is in a larger room with a high number of large obstacles such as walls, where the cluedo character is obscured or harder initially to view/identify. The solution will correctly locate and identify the cluedo character but navigation times may exceed the allocated time limit in worst case scenarios.

Conclusion

[Yuvraj Mahida]

To conclude, we believe this project has been a great success overall and we present to you a fully functioning solution adhering strictly to the specification as required. At each milestone of the project, we have continually reconsidered our design and implementation decisions for efficiency and robustness. Our video demonstration shows a full run through of our solution completing the task on the example world provided.

References

1) Yuen, H.K., Princen, J., Illingworth, J. and Kittler, J., 1990. Comparative study of Hough Transform methods for circle finding. *Image and Vision Computing*, [online] 8(1), pp.71-77. Available at: <http://www.bmva.org/bmvc/1989/avc-89-029.pdf> [Accessed 5 April 2021].