A Multi Algorithmic Approach and Evaluation to Finding Treasure Autonomously in an Environment

[Hasan Shariff - 2455269]

***Abstract*—This paper aims to detail the simulation of a robot whose focus is to find treasure in a given environment, using odometry, occupancy mapping, visualisation, and localisation. This project has been completed as a proof of concept for the necessary algorithms and the WeBots simulation software as a suitable environment for robotics. The robot used was created following a WeBots tutorial so that it can act as a valid test subject to prove the validity of this project. Subsequently, we were able to fully test the initial and goal states of the robot, where the goal states are the locations of the treasure. To further test this hypothesis, we used two different search algorithms that varied in success rates. The results obtained by experimental testing indicates that the algorithms used require modifications to improve the runtime and overall accuracy.**

I. Introduction

Autonomous robotics revolves around location and navigation. There are many different algorithmic approaches to the problem of estimating the position and orientation of a robot. Such approaches include odometry, which can be derived using known or found landmarks. Within this project the first algorithm has no prior knowledge of anything. Whereas, the second algorithm uses clues as known landmarks to assist it. Moreover, we can explore the occupancy grid mapping methodology in which the robot can interact coherently with its world [1], with respect to its global environment.

As part of this research project situated in autonomous robotics, it is necessary that all robot systems are fully tested before deployment. As we do not have a physical robot, we must select a suitable development environment for our robot. The environment must allow us to demonstrate our proof of concept of treasure hunting through the use of odometry and occupancy grid mapping. There are numerous potential solutions available to us; however, WeBots presents itself as the best candidate for this. WeBots allows us to capitalise on its low processing power and ease of use [2]. The scene tree allows us to implement nodes such as sensors, GPS, compass, and a camera.

To simplify navigation and reduce complexity we will use a two-dimensional environment. This will ease navigation for the simulated robot. This approach aims to significantly yield better results in comparison to a three-dimensional environment. Odometry and occupancy grid mapping are key and core fundamentals of this project given their unique functionality and industry standard methods for localization.

This paper will primarily explore related works relevant to this project from a multitude of authors. It will then, detail the algorithms and frameworks that were necessary to developing this project in WeBots. All of the most prevalent algorithms will be explained in detail and why they are necessary to this project. Finally this paper will culminate in experimental results which then leads onto the conclusions and limitations discovered during the development process.

II. Related Works

A. *Treasure hunting - related paper.*

Chenghui Cai's paper on "Information-Driven Sensor Path Planning And The Treasure Hunt Problem" aims to utilise a robot within a given environment as a test subject while also enabling certain abilites. These abilities include sensors which draw similarities to the sensors I have implemented. Cai briefly mentions a filed of view being implemented which insinuates that a camera was used which shares common ground with my implementation. Cai then explains using a connectivity graph while relevant, it differs from my implementation of an occupancy grid map. These papers while having the same premise and goal differ as Cai's paper utilities a decision tree formulating a sensing strategy while my paper simple records and updates the results. Cai's paper provides valid scope for improvement and inspiration.

B. *Odometry - related paper.*

K.S.Chong and L.Kleeman present a paper on "Accurate Odometry and Error Modeling for a Mobile Robot" [4] where they focus their work on developing a low cost odometry system with a high accuracy. They focus on utilising dead-reckoning however, I have focused on pure odometry which is a subtype. Both papers do focus on accurate measurement of odometry of a wheeled robot in a given environment.

C. *Occupancy Grid Mapping - related paper.*

D.Meyer-Delius et al's paper on "Occupancy Grid Models for Robot Mapping in Changing Environments" [5] explores real time updates to a grid map through the use of a probabilistic grid-based approach which is further supplemented with the use of a Hidden Markov Model. While D.Meyer-Delius focuses on dynamic environments my implementation solely focuses on a static environment. Both papers have heavy emphasis on grid-based mapping which is a core concept. D.Meyer-Delius et al also takes their experiment one step further by implementing their hypothesis on a real robot however, we were limited to using a simulated robot.

III. Algorithms and Frameworks

A. WeBots Test Environment

The primary focus is evaluating how well a robot can find pieces of treasure in a given environment, we need a suitable simulated robot to carry this test out. The robot developed in the WeBots tutorial page acts as an experimental subject for this. The test environment was enhanced to a 5 meter by 5 meter grid. This was done as we adapted the speed and size of the robot for such an environment. We also implemented 4 solid walls around the test environment so that the robot could effectively avoid such obstacles. The larger environment also simulates a real-life scenario where, the robot must navigate through an environment to find treasure.

Within the WeBots framework it implements a "scene tree", in which we are able to instantiate objects and properties for the robot and its surrounding components. The scene tree determines a node as an instance of a predetermined class with a set of properties [6].

As the robot needed the ability to navigate itself, redirect itself, visualize items, recognise items and also the ability to move itself, sensors were added to the robot. Firstly, we gave the robot the ability to move through adding HingeJoint parameters along with positional sensors, in which we instantiated a rotational motor such as a wheel. This was done for all 4 wheels. The robot was also given two distance sensors so that it can effectively identify objects. The range on these sensors was predefined and never changed. The camera node gave the robot the ability to see which was a crucial element for this project as the robot needed to find the treasure. The camera does also allow us to use the "getRecognitionObject()" function, this was utilised in the second approach and not the first in order to evaluate its significance. The GPS was also important as it helped the robot use its current location to navigate towards a new cell and to update the occupancy grid. The compass allowed us to record and measure the robot's bearing along any given axis. As this project was completed in a two-dimensional environment and odometry and occupancy mapping work best on grids the need for a grid to be overlayed was paramount. This was done through converting the current world co-ordinates to grid co-ordinates. This gave the robot ease of movement and assisted with measurements.

In the test environment, different coloured solid boxes were placed throughout the grid. All 5 pieces of treasure were represented with a blue box which remained in the same unique position throughout the running of both algorithms. In the second algorithm we introduced clues where green represented a right turn 90 degrees and white represented a left turn 90 degrees. The first algorithm only had the treasure whereas the second had both clues and treasure. The clues were introduced so that we can compare and evaluate the efficient of the robot finding the treasure. No treasure nor obstacle can be found twice which is a key component of the occupancy grid.

B. Odometry

Odometry is the prediction of the position of the robot after it has moved in any given direction with respect to the wheel revolutions. The robot was allocated 4 unique position sensors which, are connected to the rotational motors attributed to each wheel. These sensors enabled us to retrieve the encoder data in order to calculate the odometry. In WeBots with position sensors the encoder computation is done for us. The output is the rotation value which is returned in radians. Using this we can then compute the distance traveled.

Firstly we compute the distance traveled otherwise known as the linear movement, which is the average distance of each wheel. This can be seen in the following equation where ΔS is the distance traveled.

$$\Delta s = \frac{\Delta s_l + \Delta s_r}{2}$$

.

To then calculate the change in rotation, a combination of geometry and differential drive kinematics is applied to compute the angular velocity. Angular velocity can be easily calculated as the difference between the right and left distances divided by the overall distance between the wheels

$$\Delta\theta = \frac{\Delta s_r - \Delta s_l}{2L}$$

Once we have both Δs and also $\Delta\theta$ we can now calculate the change in position with respect to its global co-ordinates. Using trigonometry and if we make the assumption that the motion is a small value which can be approximated to Δs we can now compute the new co-ordinates of (x, y):

$$\Delta x = \Delta s \cos\left(\theta + \frac{\Delta\theta}{2}\right)$$

$$\Delta y = \Delta s \sin\left(\theta + \frac{\Delta\theta}{2}\right)$$

Odometry while it seems mathematically sound is still susceptible to certain extenuating circumstances which are beyond the control of the robot. Such circumstances include uneven floor contact which can lead to friction and even slipping. In addition the wheels may not have the same contact with the floor.

C. Occupancy Grid Mapping

Occupancy grid maps represents the static world environment by using a 1-dimensional grid. There are some key assumptions to be made when establishing an occupancy grid. Firstly we must assume that the probability of each cell's occupancy is independent from the other cells. Next we must know the starting position of the robot. While this implementation does not update the grid with the robot position it does track it however. Finally we assume that the map or grid used is static for the whole iteration.

By assigning the robot with many different sensor properties it enables us to use the multi-sensor data to formulate the occupancy grid. As this is a project situated within autonomous

robotics, such robot systems require the need to display sensory information which has been implemented [7]. In this project we first convert the world co-ordinates to grid co-ordinates while maintaining the assumption that a cell is empty. The occupancy grid map retrieves sensory data and updates the grid in the console with unique letters to represent different findings. As the robot navigates the environment the grid is automatically updated.

One of the limitations specifically with the implementation of occupancy grid mapping within this paper is that the Bayes rule was not implemented. The Bayes rule would allow us to get the posterior probability for each cell. As we currently do not take probability into account when plotting the grid the accuracy may be ever so slightly reduced. One way we mitigated against this was implementing several checks to ensure that no cell can be visited twice. The current implementation returns a more stochastic result where the grid does aim to prevent the robot from revisiting the same cell. The Bayes rule would have prevented missing the finer nuances of probability updates.

D. ***Robot Properties***

1) ***Navigation****:* As mentioned, there are two search algorithms that we are evaluating within this paper. The first search algorithm defines a target cell within the frontier and the robot navigates towards it. This process is repeated until all the treasure is found. The second algorithm defines clues where the robot moves naturally and if it finds a clue then follows its instruction to find the treasure. The difference between the two will be explored more, further into the paper. However, the primary properties remain the same. Both algorithms implement basic movement, obstacle avoidance, cardinal direction tracking and movement properties.

Within the Robot class definition, movement constants are defined such as base speed, turn steps, angle threshold, the simulation time step amongst other constants. Along with these the cardinal directions are defined in a compass format, this is to ensure the robots cardinal direction with respect to its bearing stays accurate and constant. Should the robot encounter an obstacle then the ”avoidance_phase” is triggered. This is a string that determines in what phase the robot currently is in. This is fundamental as each phase has different movement and general properties. On the assumption that the robot then encounters an obstacle then the robot will turn a predefined number of steps in order to avoid said obstacle. On the other hand should the robot not detect anything then it will move in a forward direction while maintaining base speed.

For the first algorithm, the robot will assume that the entire grid is empty where it has no prior knowledge of the grid. The robot will select a target cell from the array of frontier cells. Once selected the robot adjusts its bearing to face the cell and moves towards it. As it reaches the cell it will update the occupancy grid with information from the given cell, this process is then repeated to then find the treasure.

In the second algorithm again the same assumption is made about the grid being empty and having no prior knowledge of the grid. The difference now is that the robot will start from a given position and move forward till it recognises a clue. Once it does then the robot applies it's intelligence and follows the instruction provided by the clue. The robot then travels till it finds treasure. Post finding the treasure the robot will then continue on its path to then finding another clue subsequently leading to another piece of treasure. This is repeated till all treasure is found.

2) ***Computer Vision****:* Computer Vision within autonomous robotics is pivotal as it enables robots to navigate complex environments accurately. By processing visual data the robot translates the data into actions enabling the robot to preform a variety of tasks. As we have given the simulated robot a further degree of intelligence it now has the ability to perceive and understand its surroundings visually via the camera node which was equipped to it via the scene tree in WeBots.

The robot will be able to process real-time data from the camera. This means that the robot will be able to accurately classify, define and recognise objects within the environment.

To accurately test and evaluate the efficacy of the two treasure finding algorithms we equipped the first algorithm with a very simple object recognition system. The robot can process the image from the camera and draw information from the location data. This will enable to robot to either recognise an obstacle and trigger the avoidance method. Or it will be able to accurately recognise the treasure.

In contrast, the second algorithm was fully enabled to utilise as much of the computer vision aspect as possible. We removed any location data related to the treasure and purely focused on recognising objects. WeBots allows us to use the ”getRecognitionObject()” method where the camera processes the image. The robot is set up to specifically look for blue objects or to follow clues which it sees. A colour threshold system is implemented with a tolerance of 0.1. Once the image is captured it converts it to a numpy array and then focuses on the central region of interest. The effectivness of adding clues can be seen in the results demonstrating the run time.

3) ***Use of Sensor Data****:* In this project sensor data is a fundamental aspect for navigation and environment perception. Both robots are equipped with the same sensors. The sensors ensure that the robot will be able to adjust to changes and make decisions based on real time data with precision.

In order to calculate odometry the robot was equipped with wheel encoders otherwise known as position sensors. The position sensors faultlessly track the robot's displacement and rotation. However, the position sensors are also used in conjunction with the GPS and the Compass. The GPS provides the absolute position with regard to the world co-ordinates. The Compass provides absolute orientation information so that we know which direction the robot is facing. Within our implementation to precisely calculate the position of the robot the metrics from the sensors must be fused together to provide a more reliable position estimation.

The robot is also equipped with 2 distance sensors which are both used for obstacle detection. Once the distance sen-

sor recognises an object it will either trigger the avoidance movement logic.

IV. Experimental Results

A. Testing

1) ***Test 1****:* Successful/sub-optimal run of algorithm 1 - where the robot aims to find the treasure by selecting a random target cell.

2) ***Test 2****:* Unsuccessful run of algorithm 1 - the robot starts in a different position however proceeds to get stuck.

3) ***Test 3****:* Successful run of algorithm 2 - where the robot starts in a certain position and finds all clues and treasure.

4) ***Test 4****:* Sub-optimal run of algorithm 2 - the robot starts in a different position however now the clues don't align with the robot's position so the sequence is disrupted and takes 3 times as long to find the treasure.

B. Evaluation of Results

Due to the nature of the experiment within a closed environment the robot will eventually find all of the treasure. Failing or sub-optimal cases are identified as ones where the run time exceeds 60s. Now Test 1 has deemed as successful but it should be noted that result is beyond sub-optimal.

The tests produce a wide variety of results and its is clear that while the algorithms work, there is room for further development. Test 1 does find all of the treasure however, it finds the treasure in a substantial amount of time averaging over 600s. This does indicate that while the navigation does work there is definitely a more optimal way to complete this task. This more optimal way was seen in Test 3 where we introduced the second searching algorithm. This search algorithm utilised more computer vision aspects as well as a further degree of intelligence by following clues. The clues subsequently led to treasure which drastically decreased the robots run-time in comparison to Test 1. The run-time which was the main metric measured in this project fell to around 49s. By following the clues and giving the robot more guidance we saw substantially better results.

On the other hand, Test 2 was the failing case. For Test 2 I decided to move the robot to a slightly different position to evaluate how it dealt with an unfamiliar location. While the initial navigation and exploration was technically sound and showed a level of intelligence, the robot did get stuck. The robot was equipped with obstacle avoidance logic and a predefined number of turn steps to execute once the robot did encounter an obstacle. However, the robot approached the obstacle from a peculiar angle in which it did not trigger any obstacle avoidance logic nor treasure finding logic. This angle caused the robot to exhibit undefined behavior by attempting to drive over the block. Such angles were not accounted for hence the poor behavior, leading to a more advanced avoidance system needing to be implemented in the future.

In contrast, Test 4 did find all 5 pieces of treasure, however I have deemed it as a sub-optimal case. As algorithm 2 should reduce the overall runtime of the robot finding the treasure. This extended run-tine was caused by the robot being in a new position with a different navigation strategy. The run time shows this as it measured nearly 3 times more than Test 2 at 157s. The idea of the second algorithm was to keep the runtime under 60s. Test 4 also performed sub-optimally because as the robot was in a new position it would have encountered the clues in a completely different sequence meaning that it would take longer anyway. The robot did get stuck in repeated avoidance behavior which is something that needs to be developed. Algorithm 2 should be developed more so that it can handle different starting positions better rather than having a perfect use case. Another further improvement is that we could give the robot a global path planning algorithm.

V. Conclusions

A. Conclusion

We have presented a simulated robot that successfully integrates localisation and navigation capabilities. These are supplemented by odometry and occupancy grids while providing visualisation of object and treasure detection within a given environment. This implementation has fulfilled most of the objectives which were outlined in the introduction. The robot demonstrates its capability of locating and finding treasure in a given scenario. The robot is able to complete the set out objective where the performance is measured by a run time metric which varies across different use cases. The experimental results demonstrate the validity of the two algorithmic approaches however there remains scope for improvement.

B. Limitations

My implementation of the algorithms and the discussed systems exhibits a variety of constraints and limitations that affect the robot's ability to optimally function in a less structured environment such as undefined clue paths or exceeding the expected run time. The way the first algorithm works is very random and unsystematic, this leads to inconsistent performance and inefficient search patterns. The second algorithm while it provides more structure and better results it does not attain the most optimal outcome for use cases the clue path was not made for. Both algorithms struggle with dynamic path recalculation and rely on prepositioned items. These limitations identify the need for more robust path finding algorithms.

C. Future work for improvement

If I was to continue to build upon the foundations laid out within this project I would firstly ensure that posterior probability was implemented when applying occupancy grids. While the odometry data was calculated with the correct equations and methods the results were skewed this was due to the data not being reset after each iteration. I would also develop a more structured navigation system where the robot would optimise paths towards the treasure to reduce runtime. This could be done via several different algorithms such as the A star or even through more refined use of computer vision. Another potential improvement is implementing an extended Kalman Filter for more accurate position estimation. I would also implement robot learning algorithms such as SLAM.

References

[1] A. Elfes, “Using occupancy grids for mobile robot perception and navigation,” Computer, vol. 22, no. 6, pp. 46–57, 1989.

[2] Angel Ayala, Francisco Cruz, Diego Campos, Rodrigo Rubio, Bruno Fernandes, Richard Dazely ”A Comparison of Humanoid Robot Simulators: A Quantative Approach” pp. 4-6, 2020.

[3] Information-Driven Sensor Path Planning And The Treasure Hunt Problem - Chenghui Cai

[4] Accurate Odometry and Error Modelling for a Mobile Robot - Kok Seng Chong, Lindsay Kleeman Intelligent Robotics Research Centre (IRRC) Department of Electrical and Computer Systems Engineering Monash University, Clayton Victoria 3 168, Australia

[5] Occupancy Grid Models for Robot Mapping in Changing Environments - Daniel Meyer-Delius, Maximilian Beinhofer and Wolfram Burgard KUKA Laboratories GmbH D-86165 Augsburg, Germany, University of Freiburg D-79110 Freiburg, Germany

[6] Simulation of e-puck path planning in Webots - Germán A. Vargas, Oscar G. Rubiano, Ricardo A. Castillo, Oscar F. Avilés, Mauricio F. Mauledoux

[7] A. Elfes, ”Occupancy Grids: A Stochastic Spatial Representation for Active Robot Perception”

VI. GitHub Links

A.

This is the https clone link. https://git.cs.bham.ac.uk/intel-rob-2024-25/hxs321.git

B.

This is the web url. https://git.cs.bham.ac.uk/intel-rob-2024-25/hxs321