Autonomously Localizing A BWI Bot Using Surrounding Objects

Nevyn Duarte
njatord@gmail.com
njd778

Shreyas Konana
skonana@gmail.com
spk486

Joseph Moyalan
jmoyalan6266@gmail.com
jtm3986

Ishan Phadke
ishanphadke@gmail.com
iap392

***Abstract*—This project aims to autonomously localize a robot, in an effort to decrease the chance of human error in initializing the bot upon startup. Using a variation of frontier exploration, our goal is to use Augmented Reality tags, also called alvar markers, to localize the robot within an indoor space. We used a Kinect sensor to identify each AR marker, retrieved the location of the marker from our database mapping AR tag IDs to coordinates for the positional data, and reversed the transformation to find the robot's position. We find that localizing the robot using objects is more accurate, but takes significantly more time than 2D pose estimate. In general, localization using surrounding objects is more accurate and requires less human involvement on startup than 2D pose estimate, but takes more time to work on its own since it needs to move around to detect objects.**

I. Introduction

One of the biggest challenge we face in the field of robotics today is making a robot function autonomously, without the need for human input or help. Currently, the robot uses a 2D pose estimate program and a navigation goal, which uses the Monte Carlo Localization (MCL) algorithm, to estimate its location in the GDC with respect to the elevators. However, manually trying to estimate a robots location on a map and setting a navigation goal can lead to significant errors, and the method of reading in room numbers fails to work in large and open areas without many doors.

Our goal in this project is to solve this issue by using the frontier exploration package to allow the robot to explore its surroundings, and look for alvar markers that store positional data. We hope that these alvar makers will eventually be replaced with actual objects, such as computers, microwaves, and refrigerators, that will allow the BWI bots to approximate its location in a building without any human input. After an object is found, PoseNet will be used to find the orientation of the objects, and a location can be determined using the orientation and distance calculated by this process.

In order to make a truly autonomous robot, software must be able to function correctly from startup, with minimal to no human assistance. The original Monte Carlo Localization algorithm relies too much on human approximations, which can be a burden at times and not to mention inefficient. Being able to automatically position a robot on a map based on surroundings is key in the effort to make a fully autonomous robot.

II. Background

Robot localization can be defined as the process of ascertaining the location of a robot with respect to its environment. The current localization system the BWI bots are using is known as Monte Carlo Localization, or particle filter localization. This algorithm localizes robots using a particle filter which represents the distribution of likely states with each particle representing one possible state. Given a map of the robots surroundings, the algorithm estimates the current position and orientation as it moves around. It does this by generating several random guesses of where its going to be next. The robot discards these guesses that are inconsistent with its observation of the environment, and generates more guesses close to what appear consistent. Towards the end, most particles should converge to the actual location of the robot.

The representation of the state of a robot using this method typically consists of a tuple for position x, y, z and orientation . Additionally, a probability density function is used to estimate the robots current state. If a particular state space contains several particles, there is a higher likelihood the robot will be there. The current states probability distribution relies exclusively on the previous state. On startup, the robot has no information at all on its current pose. This is where the problem with the current system arises. If a poor location estimate is provided by the user of the robot, the next state will naturally be inconsistent, and this snowballs all the way to the end. Since all future estimations are based on this rough human approximation, a poor estimate will most likely result in particles being very different from the actual location of the robot.

Another method for solving this problem is frontier exploration. This package essentially aims to explore its current environment until it builds a complete map of its surroundings, or to locate itself on a pre-existing map given to it by the robot. It uses the ROS implementation of a costmap to keep track of obstacles and walls encountered during its wanderings. This costmap is populated with one of 3 values that indicate either open space, occupied space, or a buffer zone that the center of the robot should not enter. Then the algorithm essentially navigates to unknown areas by separating its surroundings into neighborhoods and mapping a path through the open space to the closest unexplored region. As the algorithm populates the costmap, it compares the costmap to the building map that

is passed to it. Finally, once enough of the space has been explored, the algorithm is able to figure out where the robot is in the building. However, this algorithm is not very suitable for our purposes. Instead of blindly exploring space, we want to be able to maximize our chances of finding an alvar marker or object.

III. Approach

Stage 1: Intelligent Frontier Exploration Using AR Tags

Our group is planning to break this problem down into a couple of different stages to simplify our complex solution. The first stage would consist of exploring the robots surroundings to find alvar markers placed strategically around the AI Lab. Similar to the frontier exploration package in ROS, our algorithm would explore the space around it; however, its primary goal would not be mapping the surrounding area, but to make an educated guess on the position of an alvar marker, which will model where real world objects are most likely to be located.

To complete this task, we intend to write our own implementation of the current frontier exploration package to make it more focused on making intelligent decisions on which unexplored spaces to explore. For example, if the algorithm detects two open spaces, one through a large opening and the other through a small opening, a door, it would ideally go through the larger opening. In most cases, a larger opening hints at a larger space, so choosing the larger opening would maximize its chances of finding an alvar marker, since the objects that the alvar markers are modeling would be more likely to be present in larger spaces. Another possible addition to our implementation of the intelligent exploration algorithm would be contingent on the assumption that objects are more likely to be along a wall, rather than in open space. The algorithm would ideally follow a wall if other options have been exhausted, and will be more likely to hit an alvar marker. These markers are tied to different positional information, stored in a database, in the coordinate frame with the origin at the elevators. In theory, we would expect the robot to be able to autonomously find an alvar marker and be able to recognize it within a relatively small time frame and without human assistance.

Stage 2: Object Recognition

The second stage would involve building a database of permanent, visible objects and its positional information, and then using YOLO to distinguish these objects while running our implementation of frontier exploration. To train YOLO to identify the specific objects in the BWI lab and the GDC, we plan on taking pictures of them from different positions and then training YOLO to differentiate these objects from each other. For example, while YOLO can currently detect a microwave or chair without training, we will use pictures to train the algorithm to differentiate between different chairs or microwaves because each object will have a distinct position. The robot would use the same method as the previous stage, running the intelligent exploration algorithm and detecting objects. The challenge of this stage is effectively using and integrating YOLO with our frontier exploration-based algorithm, which would eliminate the need to have alvar markers model unique real world objects. However, due to time constraints, we do not expect to be able to finish this stage.

Stage 3: Determining Orientation of the Object

Stage 3.1: Determining Orientation of the Object Using Triangulation: In this approach of the third stage, we will be attempting to triangulate the robots location given that it has recognized three or more markers. When it recognizes the first object, the algorithm would then calculate possible positions on a circle with the radius of the distance between the robot and the object. It would then continue accepting nav goals from the modified frontier exploration algorithm. Upon recognizing the second object, it would then calculate possible positions on the circle around the second object, and eliminating all but 2 possible positions that overlap between the two circles. The same would be done for the third object, allowing the robot to narrow its position down to one possible location. However, the issue with this approach is that the time required to find 3 distinct objects might be significantly more time consuming than using 2D pose estimate and navigation goal.

Stage 3.2: Determining Orientation of the Object Using PoseNet: An alternate third stage would be to build off of YOLO and determine the orientation of a given object using PoseNet. For each aforementioned object, we would need to train a PoseNet which would give the robot the orientation of the object in relation to itself. In comparison to our previous approach to find position of the robot using triangulation, this method would be around three or more times as efficient due to the fact that we require the robot only find one object. The PoseNet would determine the orientation of the object that YOLO has detected, which would then allow the algorithm to easily reverse the transformation and localize the robot.

IV. Experimental Setup / Evaluation

Our initial goal was to integrate the frontier exploration package with the BWI system and run frontier exploration until the Kinect identifies an alvar marker. However, using frontier exploration with the BWI bots proved to be more difficult than we expected due to major differences in formatting between BWI and the frontier exploration packages launch files. Additionally, frontier exploration was setup to work out of the box on outdoor robots called Huskies and changing the package to work with the BWI bots sensors and other hardware would be time-consuming and requires an intimate knowledge of all the robots startup code and hardware.

Along with this, we had a lot of difficulty running the demo due to differences in the launch files mentioned above. We couldnt run the demo on the 3rd floor of the GDC or rather anything than the demo launch map due to the fact that the BWI bots use a different mapping system than frontier exploration. Due to these complications, we decided to write our own makeshift implementation of frontier exploration to

make the robot explore the surrounding area on startup. While frontier exploration systematically explores the entire area around the robot using the input from the Kinect, our variation will create random navigation goals in a decreasing radius around circle surrounding the robot and will rely on the robots sensors to ensure that it doesnt hit walls or objects. As soon as an alvar marker is found, the robot will be able to read the markers ID and relate each marker to the matching positional data stored in a text file which the program has put into a map. Then, the transformation from the robot to the marker will be rotated using a rigid transformation around the z axis.This new rotated vector will then be added to the existing vector that was stored using the alvar marker id. At this point, we would publish this pose to the topic that amcl subscribes to, allowing the robot to know exactly where it is with respect to the elevators. Unfortunately, given time constraints and unforeseen problems with getting the frontier exploration package to run on the BWI robot, we were not able to get to the point of replacing alvar markers with real objects that YOLO would be able to identify.

V. Testing a scientific principle

The scientific principle we are testing in this project is the concept of autonomous localization. To test this principle, we simply ran several different trials with alvar markers in different locations. We expect the robot to move around in different directions until an alvar marker is seen, and as soon as a marker is found, the robot should stop and immediately know its location with respect to the elevators.

Fig. 1. Image of the BWI bot attempting to find an AR tag

VI. Testing system performance

The main goal of our project was to improve the accuracy of the localization of the robot because 2D pose estimate can be inaccurate since it uses human input. To test our systems performance, we wanted to compare 2D pose estimate to our localization method using factors such as time it takes to localize, accuracy of the robots location, and precision of its location after multiple iterations of the localization. If we were able to implement PoseNet to identify objects, we would also compare the precision, accuracy, and time it takes to localize using the surrounding objects to the latter results with AR tag localization. Due to the problems we had with the robot identifying the AR tags, we decided to measure the performance of our system by recording the number of times the exploration program actually identified the AR tag near it and the number of times it failed and continued exploring or went into rotate recovery. Additionally if the alvar marker was successfully spotted, we noted how long the robot took to find the marker on average using our random search method.

VII. Results

We recorded the following results after testing our system:

TABLE I
AR Tag Localization Experiment Results

	AR Tag Found	**Time to Find Tag (in seconds)**
Test 1	Yes	32 s
Test 2	No	-
Test 3	Yes	73 s
Test 4	Yes	44 s
Test 5	No	-
Test 6	Yes	25 s
Test 7	No	-
Test 8	Yes	17 s
Test 9	No	-
Test 10	No	-

In our experiment, we tested the robot in different rooms with the AR tags in different locations on walls and tables and found that on average, the BWI bot found the AR tag 50.0% of the time. Moreover, when the marker was successfully found, the robot took around 38.2 seconds on average to find it. The standard deviation of this time was roughly 21.833 seconds, indicating very large variation. This is possibly due to the fact that there is no real way to know where the robot will go first, and could go the opposite direction before making its way back.

The main goal of the experiment was to get the robot to autonomously move around until it could recognize an AR marker. We successfully implemented a program to get the robot to move around randomly and search for an alvar marker, however, as we ran the tests we got very inconsistent data. We populated an AR marker and stored the positional data of it (in relation to the elevators) in a file with the key as the header of the AR tag. However, we had lots of difficulty recognizing

the AR marker during the robots random movement. On some occasions, the robot would recognize the AR marker and stop, but the other times it would look straight at it but not recognize it. Due to time constraints, we could not successfully get the robot to always recognize the markers. Although we faced this issue, we were able to reverse the transformation from the marker to the robot which should have given us the robots correct location if it could always successfully recognize AR markers.

VIII. Discussion

Our results show that our project, in its current state, is not precise in identifying AR Tags in an attempt to autonomously localize the robot. Additionally, if we effectively implemented our plan with object identification using PoseNet and continued to identify objects at the same speed at which we are finding and reading AR tags, localizing using surrounding objects would not take that long on startup. As opposed to 2D pose recipient which we estimate takes around 20 seconds to run and use on startup, object-based localization would take around 30 seconds on average, or only around 10 seconds longer than 2D pose recipient. Additionally, if properly implemented our method would also be much more accurate than 2D pose recipient and doesn't require human input or estimation. We must emphasize, however, that we don't know how much longer identifying objects for localization will take than AR tag-based localization and thus, the emphasis of the impact of our project is about the accuracy and autonomy of the localization. The large standard deviation in the amount of time it took the robot to find the AR tag is due to the fact that we tested the robot in very different locations in which the AR tag was harder to find in places with certain objects in the way and limited space and easier in others. Whenever we tested in very narrow hallways, such as the area outside the BWI lab, we saw some very lengthy times. On the other hand, testing in wide open areas, such as the space in the GDC basement, gave us some relatively shorter times.

There were several places to improve on our approach to make it more efficient. One of the biggest improvements we believe we could make to our solution is by improving the random movement of the robot. Instead of completely random motion until it finds the marker, the robot would be able to access the costmap and thus make educated decisions on where to go. This would be implemented through the use of creating a plug in which would allow us to access the costmap that AMCL builds. Another way to use a costmap is to build one ourselves. This would require giving the costmap the proper topics to access the transforms and sensors that allow it to propagate itself. Using a costmap, we would ideally send navigation goals that explore areas most likely to contain alvar markers, such as large open spaces and along walls. These locations could easily be discerned by traversing through the costmap to discover sections of long straight occupied spaces or the largest cohesive section of open space.

Our initial idea to get the position of the robot from the AR marker was by continuously sending the robot to the offset flip of the AR marker and storing the location in the AR marker. The robot would ideally recognize the AR marker as it randomly moves around and be able to move its current position. This idea was based on the assumption that the robot would go to the same goal regardless of where it saw the AR marker from. However, when we tested it the robot went to different locations and had different orientations each time. Due to this, we chose to store the current position of the AR marker on the coordinate frame with the elevators as (0,0,0) and to find the current position of the robot by reversing the transformation from the robot to the marker. Another improvement we could make would be improving the accuracy of the locations stored in our alvar markers. To find the position of the AR marker in relation to the elevators, we found the current position of the robot and essentially did the same process as we did to find the robot's position in relation to the alvar marker. We found this to be accurate since the kinect gave us approximately the same distance each time we ran a trial.

However, we believe that our current implementation is a step forward in the right direction. The goal of autonomous robotics is a robot which is capable of functioning without human input on startup. Since the joint 2D pose estimate and Monte Carlo localization system required a human to kickstart the process, we believe that our perceived version of the package with all the stages and improvements completed will be superior.

IX. Conclusion

Ultimately, our final project aimed to autonomously localize a robot, in an attempt to reduce the chance of significant human error during startup and allow the robot to be near independent of human input. Although our initial expectation of modifying the frontier exploration package itself was not met, we were still able to successfully implement our own algorithm to randomly move and search for alvar markers that are tied to positional information. In other words, immediately upon startup, there is no need for a manual 2D pose estimate nor a navigation goal. In addition, we successfully populated alvar markers with its positional information, all with respect to the elevators of the GDC third floor. This positional information was stored in a text file that would be accessed later on by our program if an alvar marker was found. If the alvar marker was successfully seen by the robot, the location of the robot itself would be calculated by reversing the transformation of the robot to the marker, using a rigid transformation to rotate the vector, and adding it to the the marker's displacement from the elevators.

References

[1] A. Topiwala, P. Inani, and A. Kathpal, "Frontier based exploration for autonomous robot," *CoRR*, vol. abs/1806.03581, 2018. [Online]. Available: http://arxiv.org/abs/1806.03581

[2] Visual localization using ar tags as landmarks. [Online]. Available: http://github.com/JGOOSH/visual$_{a}r_{l}ocalization$

[3] F. D. S. T. Dieter Fox, Wolfram Burgardy, "Monte carlo localization: Efficient position estimation for mobile robots," Rome Labs, NY, Tech. Rep. DAAE07-98-C-L032, Feb. 2017.

[4] I. The MathWorks. Monte carlo localization algorithm. [Online]. Available: https://www.mathworks.com/help/robotics/ug/monte-carlo-loc alization-algorithm.html

[1] [2] [3] [4]