**TurtleServer**

RBE3002 Final Project

Worcester Polytechnic Institute

Ransom Mowris, Lillian Walker, Alex Woodyard

For the last project in WPI’s Unified Robotics series, our team used the Robot Operating System (ROS) to program the Kobuki Turtlebot to autonomously serve refreshments to guests at a party. This required the integration of localization, mapping, and path-planning with higher-level robot behavior. We demonstrated our understanding of the fundamentals of mobile robotics by successfully serving donuts to both simulated and actual guests of a party.

## Table of Contents

Table of Contents 2

Introduction 3

System Requirements 3

Methodology 4

Subtasks 4

Room-Scanning Behavior 4

Recognizing Party Guests 4

Navigating to Party Guests 4

Prioritizing Guests 4

Setting a Navigation Goal 5

AMCL 5

Serving Party Guests 5

Results and Discussion 7

Scanning the room 7

Recognizing Party Guests 8

Navigating to Guests 9

Serving Guests 10

Conclusion 12

References 13

# Introduction

For the final project of the final class of the Unified Robotics series, we were tasked with the programming of an autonomous mobile robot[[1]](#footnote-1). More specifically, our project required the programming of a kobuki Turtlebot via ROS to autonomously serve refreshments to guests of a party. Accomplishing this task required the integration of mapping, navigation, and localization into an effective system.

## System Requirements

To approach the problem, we deconstructed it into a list of requirements. System requirements are the fundamental element of systems engineering and our understanding of this problem’s requirements determined our approach to its solution. We identified the following objectives as necessary for the completion of the project:

* The robot must identify and serve three groups of guests within ten minutes
* The robot must act autonomously
* The robot must not collide with obstacles or party guests
* To serve a guest, the robot must be within 60cm of their legs

To accomplish these goals, we had to utilize several aspects of robot navigation, some of which we had not experienced before. Localization, environment scanning, and path-planning were all important components of our system. This project was also a good introduction to creating a robot that would work around humans, a concept which wasn’t explored in any other unified robotics courses.

# Methodology

## Subtasks

We broke our system down into these subcomponents:

1 – Scan the room for obstacles

2 – Recognize party guests

3 – Plan navigation to party guests

4 - Serve the party guests with delicious donuts

Informed by our definition of these subtasks, we drafted a flow-chart for our code. This flow-chart can be seen in Appendix B. This helped us to recognize what functions we needed to write, how our program would progress during its execution, and areas where problems could arise.

## Room-Scanning Behavior

To create a map of the room, we had the robot drive in a pre-determined path while running a mapping algorithm. Our path involved driving in a square to visit the four corners of the room, then driving in a diagonal line across the room. Once the costmap was generated by AMCL\_demo (explained below), our code compares the cost map to a pre-recorded map of the room without anyone in it.

## Recognizing Party Guests

After the robot has scanned the room, the cost map includes clusters of occupied cells. Any clusters that are not seen in the pre-determined map of the room are assumed to be people. Clumps of occupied cells are consolidated into lists of points and the centroids of these clumps are calculated. Each centroid is wrapped up in a “Person” object, and the location of the centroid is recorded as the location of that person. Once the centroid of every clump has been calculated, this list of people is sent to our ServingBehavior class which takes care of serving every party guest.

## Navigating to Party Guests

Our mapping behavior generated a list of Person objects. To serve party guests in a sensible manner, this list required some modifications. First, the robot sorts the list according to proximity to the robot. Next, it generates navigation targets for each guest. Finally, the robot drives to the guests and offers delicious donuts. These components of our code are detailed in the subsections below.

### Prioritizing Guests

Guests are sorted based on distance from the robot and whether they have recently been served. To reduce the time spent driving around, nearby guests are prioritized over more distant guests. After three minutes, a guest is flagged as being hungry and the robot will revisit them soon.

### Setting a Navigation Goal

After our robot generates a list of people, it needs to plan to navigate between them. Since trying to drive to the centroid of a person would be an impossible goal, we needed to find a way to offset the robot’s desired position from each person. To do this, our code looked at a person’s position, and generated four candidate positions next to them: up, down, right, and left. Whichever point was closest to the robot’s current position was chosen.

### AMCL

To simplify localization, driving the robot to guests, and avoiding obstacles along the way, we decided to use the ROS package “Augmented Monte Carlo Localization” (AMCL)[[2]](#footnote-2) in our project. This package uses the Monte Carlo family of algorithms described by Thrun et al. in their *Robust Monte Carlo Localization for Mobile Robots*[[3]](#footnote-3)*.* This algorithm works by combining regular particle filter localization with a novel approach to traditional Monte Carlo Localization. While traditional MCL guesses a new pose with odometry, then uses sensor data to check this guess with prior data, AMCL combines this with an “inverted” implementation of the traditional algorithm. This version guesses the robot’s location via sensor data, then verifies it against the robot’s odometry readings (Thrun et al., 2001). According to Thrun et al., AMCL overcomes many of traditional MCL’s shortcomings while remaining efficient, versatile, and robust.

AMCL also helped us with navigation. The AMCL\_demo package automatically drives the robot to poses published to the */move\_base\_simple/goal* topic, avoiding occupied gridcells on the way. Although we could not find information about how the autonomous driving is planned, the robot’s tendency to almost always drive in an arc leads us to suspect that this uses tentacle path-planning. We handled all robot locomotion by sending poses to this topic.

## Serving Party Guests

Once the robot is within 50 cm of the centroid of a person, we consider the robot to be close enough to serve them. To notify the party guest of the robot’s arrival, we commanded the TurtleBot to play a sound by publishing a sound message to the *mobile\_base/commands/sound* topic. After playing a sound, the robot waits a few seconds so that the guest may grab their donut, then it flags that person as “visited,” notes the time of serving (this enables the robot to revisit that person after a few minutes), and moves to the next party guest.

# Results and Discussion

To evaluate our robot’s performance, we revisited our original list of subtasks. Our robot’s success in performing each subtask is outlined below.

## Scanning the room

*Was the robot able to localize quickly after starting our program?*

By choosing to start the robot near a unique feature (the center vertex of the tables) the robot was quickly able to localize itself in the map. As the robot searched the room for people, it was able to localize itself quite accurately especially as it began to sense objects that it had previously observed. Occasionally, the robot would become confused and lose its estimation of its position, but AMCL automatically corrects this by spinning the robot. Re-localizing usually only took a few seconds.

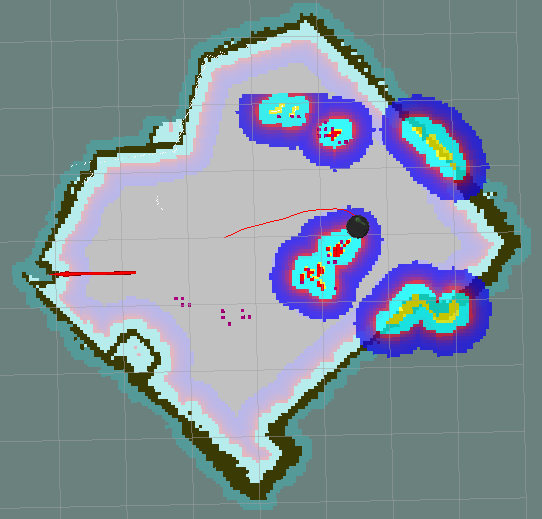
*Did the robot effectively scan the entire room?*

Figure 1 - RVIZ as our robot was mapping. Red dots represent obstacles and purple dots are centroids of obstacles. These clusters of points are consolidated into single points once the mapping phase finishes.

The robot was able to successfully scan the room by driving the room’s perimeter and a single diagonal between two corners. This allowed the robot to see the entire room and collect data about any discrepancies between the map of the empty room and the observed state of the room. A screenshot of the mapping phase of our robot’s operation can be found above, in **Figure 1**.

## Recognizing Party Guests

*How well did the robot find party guests?*

The robot does a good job of recognizing party guests. It sometimes over-estimates the number of people found in the room (it registers each leg separately person), but our serving behavior compensates for this quirk. When a person is served, our code flags any person within a certain distance (.6m) as having been served. Thanks to this distance check, our robot does not pitifully try to offer donuts to both legs of every party guest.

*How could our guest-recognition code be improved?*

We might want to consider scanning for guests while the robot is serving. This would require our guest recognition to be more robust, as when we tested this implementation, our robot was prone to revisit guests repeatedly, thinking it was discovering them for the first time.

## Navigating to Guests

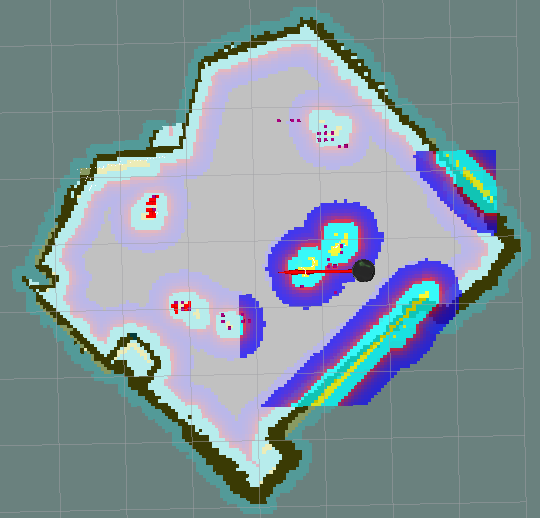
**

Figure 2 - RVIZ as our robot served guests. Purple cells have been consolidated into Person objects in the code.

*How well did AMCL work?*

Once the robot successfully identified the people in the room, it began to navigate to them based on how close they were and how long it has been since the robot had last visited them. **Figure 2**, above, shows data from this phase**.**  We were successful in navigating to each person by sending a pose to AMCL. AMCL was consistently able to navigate to people, except for a few exceptions when the robot thought it was trapped in a corner and could not find a valid path out of the corner. The AMCL package itself wasn’t perfect in its operation. The robot often needed to stop to localize itself by spinning around

*Did the robot bump into anything?*

Since the map of the room did not change after the initial scan of the room, the robot, thanks to AMCL’s guidance, did not bump into any obstacles. The AMCL code was able to successfully able to plan paths around the party guests who stood around the robot. In addition, since we inflated the obstacles significantly, the robot had a tendency to keep a safe distance away from obstacles in the first place.

We did have one slight issue with collisions. The AMCL demo for the Turtlebot generates a c-space for the dimensions of the Turtlebot chassis. Unfortunately, some of our class’s Turtlebots use laptops that are slightly wider than their chassis. This results in collisions when using the default AMCL package. To rectify this issue, we edited the *occdist\_scale* parameter in the AMCL package and re-compiled it.

## Serving Guests

**

Figure 3 - A TurtleBot driven by our code offers donuts to a hungry party guest

*Did the robot serve everyone?*

The robot was successful in serving all the guests of its donut party. As it drove to each guest, it waited until it was less than half a meter away from the center of the person before stopping and playing a sound. It stopped driving for a few seconds to enable the human(s) it was serving to retrieve their delicious goodies, then moved to the next closest person who had not been served. **Figure 3** above shows our robot as it served donuts. Once everyone was served, the robot started scanning the room again.

*How did people interact with the robot? Did it get too close or stay too far away?*

The robot interacted well with our party guests. It stopped within a reasonable distance of their legs and showed no favoritism for any particular guest. Guests commented on the utility of our robot’s playing of a sound when it served a person. If we hadn’t played a sound, our guests might have been confused as to whether the robot was offering food to them or stopping momentarily. Our robot also stopped for an adequate amount of time to serve donuts. None of our party guests needed to chase after our robot to receive their snack.

## Conclusion

This lab was extremely educational. Accomplishing this (seemingly simple) task of delivering food required extensive familiarity with the principles of autonomous navigation. To recognize people, we needed to intelligently parse information from sensor and map data. To know where they were in the room, we integrated a complex localization algorithm into our system. To avoid running into people, we needed a path-planning algorithm. Even figuring out where to drive the robot in order to serve a party guest (not too close, but not too far) required an extensive amount of code.

This lab also offered us our first experience working with a robot that would interact with humans. This discipline, dubbed “Human-Robot Interaction,” is one of the fastest-growing research areas in robotics. Our system was very simple and required only two human-oriented requirements (not running into people and needing to be within 60cm of a guest to successfully serve them), but we learned that human-facing robots require additional design considerations. For example, an industrial robot that works in close proximity to people needs to be very safe.

Our team would like to thank Professor Chernova, TA Dan Miller, and Robotics Grand Wizard Joe St. Germain for their contributions to this project. We couldn’t do this without you all!

# References

*Robust Monte Carlo Localization for Mobile Robots* - Sebastian Thrun, Dieter Fox, Wolfram Burgard, and Frank Dellaert April 2000

1. RBE3002 Lab 5 Assignment Document: http://goo.gl/vfMCS7 [↑](#footnote-ref-1)
2. <http://wiki.ros.org/amcl> [↑](#footnote-ref-2)
3. *Robust Monte Carlo Localization for Mobile Robots* - Sebastian Thrun, Dieter Fox, Wolfram Burgard, and Frank Dellaert April 2000 [↑](#footnote-ref-3)