I Will Walk You There
The Intelligent RoboKart Final Report

Prepared by:

Hou, Chuyi Li, En Xu Shentu, Chengnan Chen, Ryan

Robotics Institute, University of Toronto

Contents

1 Project Overview

The demand for fast-paced and technology-driven shopping experiences has become a growing trend in customer behaviour since the COVID-19 pandemic. As a result of this phenomenon, the retail industry, namely grocery stores, have suffered significant revenue losses due to customer satisfaction declines. To address this exacerbated issue, we propose an intelligent robotic shopping cart, or RoboKart, that can provide a complete robotic shopping experience for supermarket and grocery customers. This is achieved through the addition customer following, shopping guide, and auto-checkout features to the old-fashioned shopping cart.

The rest of the report is organized as follows. We start by providing a brief background to identify and motivate the main issue that is resolved through this robotic solution. Furthermore, the main design objectives regarding the proposed solution are outlined. In this project, we focus on the shopping guide feature of the RoboKart and demonstrate its functionality in the *Gazebo* simulation environment. An overview of the prototype design is provided based on the five key modules that the RoboKart is composed of, which includes planning, perception, mapping, localization, and user interface. Specifically, the planning module of the prototype is rigorously evaluated through a three-phase evaluation process, where the RoboKart is commanded to navigate to desired locations in an empty store, a static store, and a "real" dynamic store. Through this procedure, the RoboKart prototype is able to successfully navigate through all three store environments. Finally, some challenges encountered during the design process are highlighted and some helpful advice are given to future students taking on a similar design task.

2 Background

As online shopping becomes more established and prevalent due to the COVID-19 pandemic, supermarkets and the retail industry as a whole has been severely disrupted and enervated. Consumers have grown accustomed to the ease and convenience of online shopping and e-commerce, which experienced significant growth over the past few years. This shift towards fast-paced and technology-driven shopping experiences has led to a decrease in customer satisfaction within the retail industry. The American Customer Satisfaction Index (ACSI) reports that 86% of all retailers in the industry have experienced customer satisfaction declines after the global pandemic [1]. This issue is particularly prevalent in the grocery store and supermarket sector, where customer satisfaction has dropped by 2.6% from 2019 to 2020 with no signs of recovery. The increasing frustrations from the consumer side has also been reflected in revenue loss from the industry as a whole. Currently, 77% of consumers are less likely to shop at a retail store again if they encounter a bad experience such as long wait in checkout lines [2]. This accumulates to a total of $37.7 billion loss in potential sales each year [3]. Grocery stores specifically are experiencing as much as a 5.5% drop in quarterly same-store sales [4].

To combat this alarming issue, we are proposing a robotic solution to provide a more intuitive and efficient grocery shopping experience. This strategy also aligns with the direction that the industry will be taking in the coming years. As reported in a survey conducted by RetailWire, 77% of large retailers have identified that it is important to create a robotics-oriented strategy to resolve the previously mentioned problem [5]. Furthermore, 47% of these retailers are planning to take action and get involved with an in-store robotics project within the next 18 months. Therefore, by solving a major consumer-oriented problem backed by significant interest from the industry, we are confident that the robotic solution will have a real positive impact on the future of grocery stores and supermarkets.

The robotic solution that we would like to present is RoboKart, an intelligent robotic shopping cart. Equipped with customer following, shopping guide, and auto-checkout capabilities, RoboKart aims to increase customer shopping efficiency by providing a complete robotic shopping experience. There are a few existing products or products in development that we have identified as potential competitors in this space. However, none of these products offer a complete and all-encompassing solution similar to the proposed design. For example, JD has developed a customer-following shopping cart that can track and follow grocery shopping customers based on signal emitted from wristbands worn by the customers [6]. Despite a relatively reliable method to eliminate the need for customers to physically push the shopping cart, this product does not include any product location navigation features in addition to the basic customer tracking functionality. Similarly, Caper has designed a vision-based system that can detect products placed in a shopping cart and provide auto-checkout functionalities based on known product information [7]. While this solution provides an efficient system in reducing the time needed to complete purchases, it does not address the time lost from manually searching for items from aisle to aisle. Therefore, we intend to combine the customer following and auto-checkout features from these existing solutions with a complete product location navigation system to produce RoboKart, the ultimate robotic shopping assistant.

3 Design Objectives

RoboKart, as an intelligent alternative to existing shopping carts, is designed with the goals of being safe, user-friendly, and marketable. Besides the basic functionalities of a wheeled shopping cart, RoboKart should function as a shopping assistant to guide customers to items they want to find or their desired location. RoboKart must do so in a safe manner without hurting any persons or animals and without causing damage to items or the environment it operates in. In addition, the user interface should be intuitive and easy for customers to learn and use so that more customers are willing to use RoboKart instead of a traditional cart. Ideally, RoboKart should also be cost-effective for it to be marketable to medium or large-sized retail stores that we are targeting.

The following subsections will outline the specific design objectives established to achieve these goals, as well as relevant constraints. Considering the limited time and resources allocated to the project and the complexity of an intelligent shopping cart system, we choose to limit our scope and focus on the planning and collision avoidance subsystems. The other subsystems and their objectives are discussed but with less depth. In addition, we provide a discussion on some non-technical concerns relevant to our system, along with relevant objectives.

3.1 Design Constraints

RoboKart is set to operate in a medium to large-sized retail store, with results in the following constraints:

1. The design must be able to handle dynamic environments with pedestrian traffic
   (a) The design must be able to detect obstacles and people within 2 meters
   (b) The design must update its detection faster than 10 Hz
2. Path planning must be complete and optimal
3. The design must have enough power to operate continuously for at least 3 hours between charging

Table 1: List of objectives for RoboKart and its functional components

- Design should be safe
  - Design should not cause damage to its users, people or animals nearby, and its environment
  - Design should be able to sense static and dynamic obstacles nearby and avoid them
- Design should be efficient in its guiding process
- User interactive should be easy and intuitive
  - The design should understand general product types, specific names, as well as common locations in the store, such as the checkout counter or the washroom
  - The design should remain in close proximity to the user at all times
  - The design's motion should be comfortable to follow
- Design should minimize the cost of production
  - Software algorithms, such as planning, should be computationally inexpensive
  - Hardware components, especially for sensing and actuation, should be cost-effective
- Overall system should be easy to maintain

3.2 Non-Technical Concerns

Besides the objectives and constraints mentioned above, there are topics that do not have direct technical implications in this project. Nonetheless, they are important topics to recognize and be mindful of during the design process.

3.2.1 Accessibility

As an assistive device for all shoppers, accessibility is essential for RoboKart. The device should support people with special needs. For example, the user interface design should pay special attention to people with visual or hearing impairments by providing visual or audio cues to the user when needed. Under the premise of safety, RoboKart should adapt to the behavior of all types of customers whether they are in a hurry and walk very fast or have mobility problems and walk very slow.

3.2.2 Job Displacement

With the new wave of automation driven by robotics and COVID-19, jobs with higher physical proximity, such as on-site customer interaction in this project, face greater disruption [8]. Similar to automated teller machines (ATMs) or airport kiosks, devices like RoboKart could push more disruption to the offline retailing industry like self-checkout devices already are. Specific implications of RoboKart to the job market is out of the scope of this design proposal, but is certainly a topic worth attention.

3.3 Design Reflection

Through the development of the prototype, we encountered the most difficulties trying to meet the objective of avoiding both static and dynamic obstacles under limited computation resources. Specifically, the challenge arises from the planning module dealing with different behaviours of the two types of obstacles. For static obstacles, the planner seeks to keep its location in the map for the entire run to avoid being blocked by the same obstacles multiple times. For dynamic obstacles, on the other hand, the planner seeks to only keep track of their current locations to avoid blocking paths that are actually feasible. In an ideal situation with lots of computation resources, object segmentation can be used to label static and dynamic obstacles and handle them differently downstream. However, such approaches usually require GPU resources, which conflicts with the objective of minimizing the production cost.

4 Design Prototype Overview

We propose a new mobile shopping assistant (RoboKart) that combines the existing customer following [6], automatic checkout [7], and item locating functions [9] with a customer guide function designed by us. Hence, our primarily focus is designing an autonomous shopping cart that can lead the customer to the desired item location. The mechanical design of RoboKart is presented in Figure 1. The handle is integrated with a user interface screen, POS machine for payment, a barcode scanner for scanning packaged items, and an ultra-wideband tracker for localization in the store. Under the shopping basket, a weight sensor is installed to weigh unpackaged items such as fruits. On the bottom of RoboKart, a differential drive system is used for actuation, along with a battery pack to provide power and a 2D LiDAR sensor for obstacle detection. We developed the prototype in simulation to demonstrate its capabilities of guiding customers around the store and avoiding obstacles. The simulations are done using the *Gazebo* package because we can make use of its 3D modeling function to simulate the grocery store, obstacles, pedestrians, and RoboKart. We implemented software algorithms in *Python*, which operates the mobile robot model developed based on the Turtlebot3 package through the Robot Operating System (ROS).

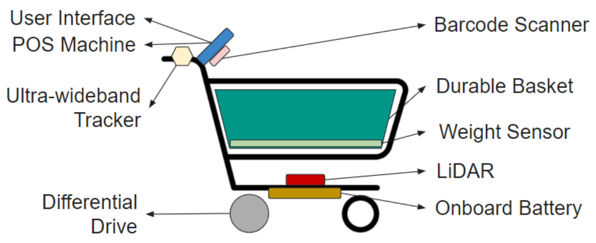


Figure 1: RoboKart's mechanical design

At a high level, the RoboKart prototype (with a focus on shopping guide feature) contains planning, perception, mapping, localization, and user interface modules. The communication between these modules and with the *Gazebo* simulation environment are illustrated in Figure 2. The process starts with the user selecting target locations to go in the user interface module. This information is passed to the global planner along with the store map from the mapping module and cart pose from the localization module. The global planner finds feasible waypoints for the local planner to follow, which also receives obstacle information from the perception module to determine the optimal actuation commands. This actuation commands is passed to the mobile robot model and simulated in *Gazebo*. The local planner also communicates back to the global planner in case the current route is not feasible, and the perception module can update the map when static blockages are detected; these optional information flow are highlighted by the orange dotted lines in Figure 2. In the following subsections, we will describe the detailed implementation of these modules.

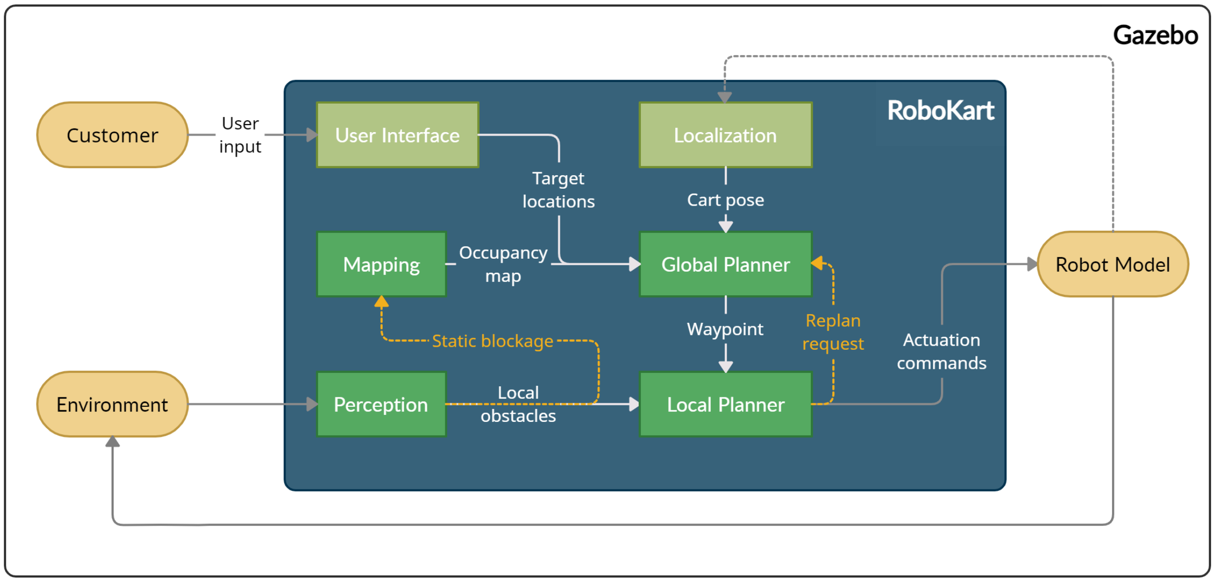


Figure 2: RoboKart's major functional components and information flow

4.1 Planning

We divide the planning module into two parts, namely the global and local planner. The global planner is responsible for generating waypoints from the current position of RoboKart to the desired location in the store. Every waypoint is about 3 meters apart. On the other hand, the local planner generates control inputs that drive RoboKart from one waypoint to the next, avoiding obstacles along the way. The two planners cooperate and communicate with each other seamlessly when in action to ensure RoboKart drives to the desired location safely and efficiently. The details about the two planners are presented as follows.

4.1.1 Global Planner

The global planner takes as input a bird's eye view (BEV) greyscale store layout for occupancy checking, a starting position, and a goal location and generates a path represented by a list of milestone waypoints. First, we take a store layout diagram and discretize it into grids as shown in Figure 3. Meanwhile, we identify the corresponding grid of the starting and desired positions. Note that the starting position is the current pose of the robot received from the localization module, and the desired location is the nearest empty cell of the grocery item that the customer wants to buy. We then adopt the A* Algorithm [10] to find an optimal path from start to goal given the grid map. The exploration of A* is prioritized based on the cost $f = g + h$ where g is the cost-to-come and h is the heuristic of cost-to-go. We define a unit cost of 1 for travelling between each adjacent cell and a cost of $\sqrt{2}$ for travelling diagonally. In order for the planner to generate optimal path, a consistent and admissible heuristic is needed. We adopt Manhattan distance as our estimated cost-to-go. Lastly, we publish the found path using a list of waypoints with an example shown in Figure 4.

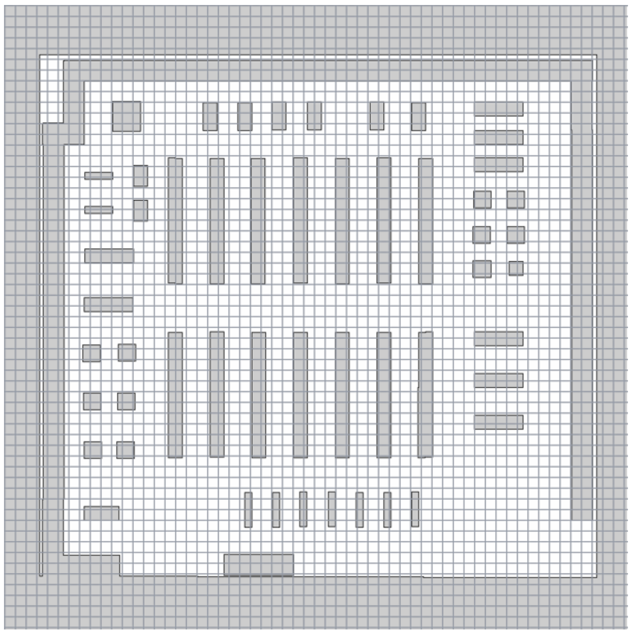


Figure 3: Sample grid map input: a discretized store map

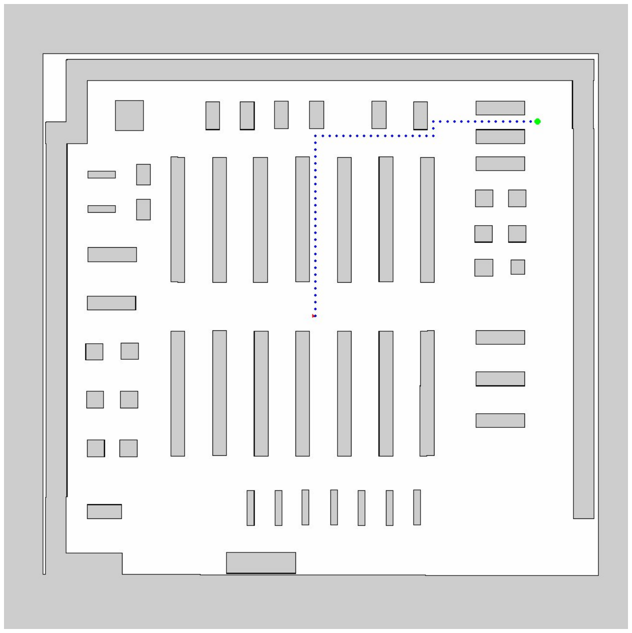


Figure 4: Sample path from global planner. The starting point is denoted by the red triangle and the goal point is represented by the green circle.

We pick A* Algorithm for path planning over other alternatives such as Dijkstra's Algorithm [11], Breadth-first Search (BFS) and Depth-first Search (DFS) for several reasons. For a grid-based search problem with unity cost, Dijkstra's Algorithm works similar to BFS where it needs to explore all the surrounding neighbours to reach the goal which could be time consuming. On the other hand, A* Algorithm uses a heuristic for cost-to-go which tries to guide the search towards the goal location. Even though the Manhattan heuristic is unaware of the obstacles, it mostly works better than a heuristic of 0 (i.e. Dijkstra and BFS). Additionally, DFS algorithm is also a complete planner but the path found is usually not optimal, meaning the robot may spend extra time and distance travelling to the same goal location. Therefore, we find A* Algorithm suitable as the global planner of RoboKart.

4.1.2 Local Planner

Given a predefined list of linear velocities v and angular velocities ω, we follow Equations 1, 2 and 3 to simulate a trajectory for every option pair.

$$x_k^v = \frac{v}{\omega}\sin(\omega \cdot k\Delta t) \tag{1}$$

$$y_k^v = \frac{v}{\omega}(1 - \cos(\omega \cdot k\Delta t)) \tag{2}$$

$$\theta_k^v = \omega \cdot k\Delta t \tag{3}$$

where $k = [0, K]$ is each simulated step up to K, and $\{x_k^v, y_k^v, \theta_k^v\}$ denotes the position and orientation of the kth point of the trajectory in the vehicle frame. Specifically, our linear velocities choices are $V : \{-0.1, 0.1, 0.15, 0.23\}$m/s and angular velocities choices are $\Omega : \{-1.5, -1.4, ..., 1.4, 1.5\}$rad/s. A pairwise combination of V and Ω yields a total of 120 options. We choose to simulate trajectories 4.0 seconds ahead with an interval Δt of 0.5s which corresponds to $K = 8$.

Then, we transform the computed trajectory points from the vehicle frame to the inertial frame as follows,

$$\begin{bmatrix} x_k^i \\ y_k^i \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta^i & -\sin\theta^i & x^i \\ \sin\theta^i & \cos\theta^i & y^i \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_k^v \\ y_k^v \\ 1 \end{bmatrix} \tag{4}$$

$$\theta_k^i = \theta_k^v + \theta^i \tag{5}$$

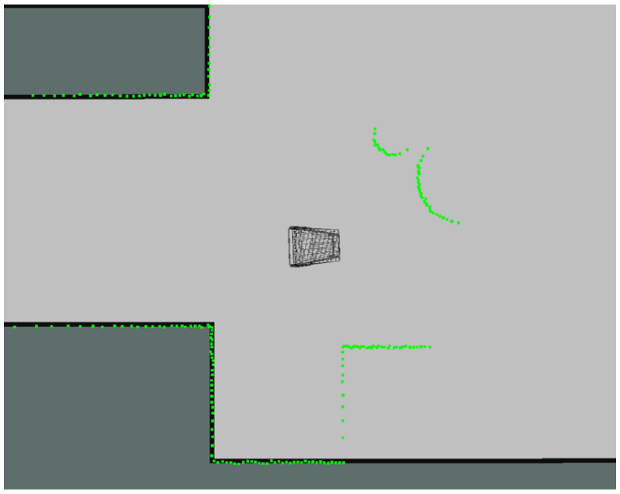
where $\{x^i, y^i, \theta^i\}$ is the current pose of the robot in inertial frame, and $\{x_k^i, y_k^i, \theta_k^i\}$ denotes the trajectory points in inertial frame. We then convert them to map indices for collision checking and reject possible trajectories if obstacles are found in the route. In particular, we merge the original occupancy grid map from the store and the mapped environment from our perception module with a logical OR operation. We then dilate the resulting map to make obstacles "thicker" to account for RoboKart radius. Collision checking is simply indexing on the dilated map which is quite computationally efficient.

If there are no valid options available after collision checking, which is likely the case where our preplanned global path is completely blocked by an obstacle, we make necessary change to the occupancy map and ask the global planner to replan a path with the same goal location but an updated occupancy grid.

Normally, we still have a list of control options valid after collision checking. We compute a cost function C for each collision-free trajectory as described below and pick the one with the lowest cost.

$$C = \frac{1}{K}\sum_{k=1}^{K}\sqrt{(x_k^i - x_g^i)^2 + (y_k^i - y_g^i)^2} - \beta|v| \tag{6}$$

where $\{x_g^i y_g^i\}$ is the position of the goal in inertial frame and β is the weight on the second linear velocity term. Essentially, we prioritize the trajectories based on the average distance from each point in trajectory to the goal location and add another term with linear velocity to reward the ones attempting a safe but slightly more aggressive move to possibly go around an obstacle rather


(a) LiDAR scan visualization in RViz

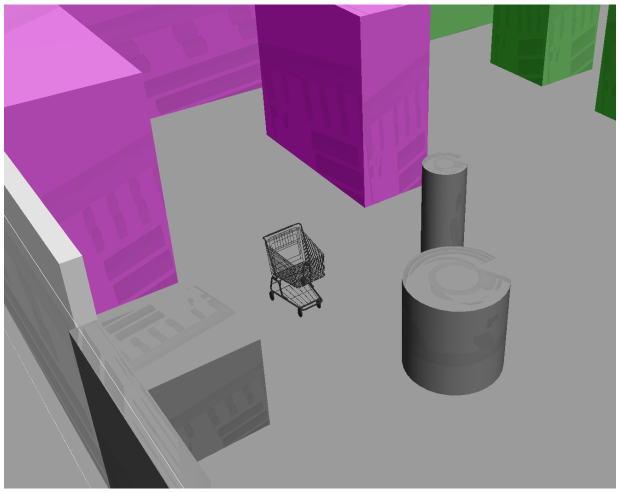

(b) Environment in *Gazebo*

Figure 5: LiDAR points outlining the shape of an obstacle detected by the RoboKart.

than trying to be "timid" and stay at where it is. We then publish the control input with the lowest cost C. Meanwhile, we move on to the next intermediate goal if the current milestone is reached.

We have also considered other local planner alternatives such as sampling-based (e.g. RRT [12]) and variational planners. Sampling-based planner usually sample the control inputs randomly to quickly explore the workspace, and collision-checking is performed as new points are added to the explored space. They could often operate relatively fast, but generate poor-quality paths due to the randomness. Variational planners usually optimize trajectory based on a cost function which contains penalties for collision avoidance and robot dynamics; however, they run slow and less likely to converge to a feasible solution. Thus, trajectory rollout (also known as the lattice planner) is a better design choice when trading off path quality and runtime.

4.2 Perception

The perception module is responsible for sensing and detecting obstacles while RoboKart is in operation. The 2D LiDAR sensor is chosen for this purpose due to its accuracy and efficiency in sensing the surrounding environment of the RoboKart. In this application, contact-based sensing is undesirable and dangerous because collision between humans and the RoboKart can result in potential injuries. Furthermore, ultrasonic sensors are excluded from our consideration due to their limited detection range and material-dependent sensing accuracy. In the case of cameras, it is significantly more difficult to extract depth and range information from images compared to LiDAR laser scans. Therefore, LiDAR is selected as the optimal sensor to adopt for the perception module. As shown in Figure 5, the 2D LiDAR sensor can accurately detect the outline of an obstacle visualized through LiDAR points in the simulation environment.

4.3 Mapping

The mapping module is utilized to update the locations of obstacles detected by the perception module in an occupancy grid map. To begin, all cells in the occupancy grid are initialized to a prior likelihood of being occupied of 0.5. Then, all LiDAR laser scans from the perception module are looped over to obtain the endpoint of each scan. The log-odds of the cell that contains the endpoint is updated as follows,

$$l(c_j|y^t, x^t) = l(c_j|y^{t-1}, x^{t-1}) + \alpha \tag{7}$$

where α is a tunable parameter, c_j is the jth cell in the map, y^t is the measurement from LiDAR at time t, and x^t is the pose of the robot at time t. Lastly, probabilities are recovered from the log-odds using the following equation

$$p(c_j|y^t, x^t) = \frac{\exp(l(c_j|y^t, x^t))}{1 + \exp(l(c_j|y^t, x^t)} \tag{8}$$

Given this map which contains the locations of the currently detected obstacles, a logical OR operation is performed between the obtained occupancy map and the original occupancy map built from the store layout. This update process produces a new map which contains both the store environment and locations of detected obstacle. An example of the updated occupancy grid map is presented in Figure 6, which is generated with the environment shown in 7.

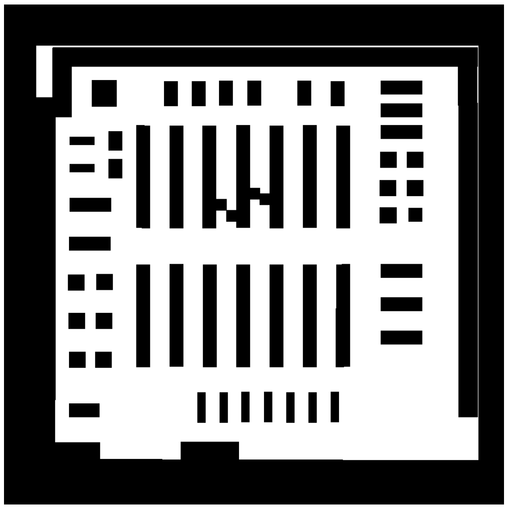


Figure 6: Updated occupancy grid map containing the locations of the detected obstacles.

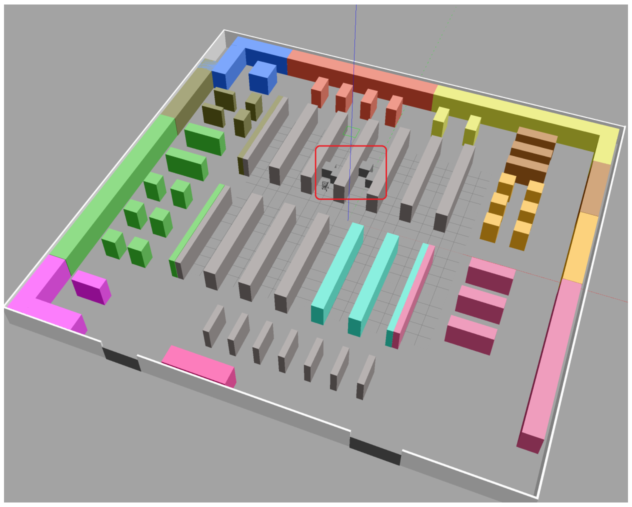


Figure 7: Corresponding environment with obstacles (in red box).

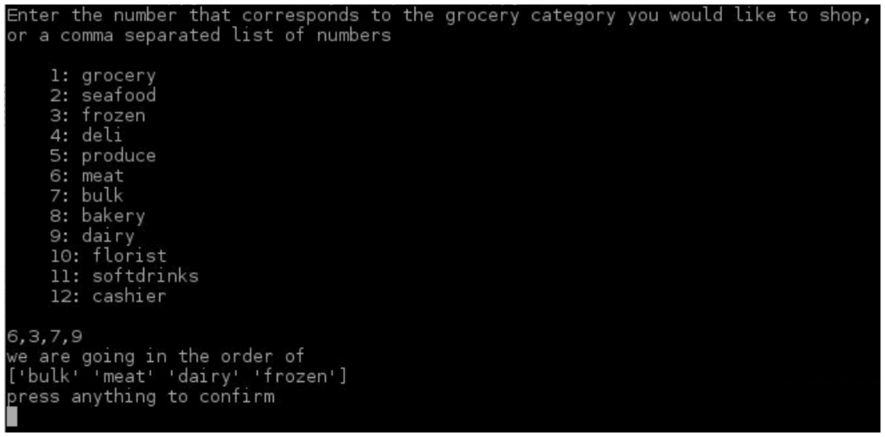
For our prototype, the mapping module only considers the laser endpoint rather than performing ray-tracing to determine all cells in the field of view of the scan. This is designed to improve the overall efficiency of RoboKart because the operation of ray-tracing and updating all cells along the laser scan can take quite a long time to compute. Thus, to reduce the computation time required to complete the mapping process, only laser scan endpoints from the perception module are utilized in the mapping module.

4.4 Localization

The localization module is used to determine the position of the RoboKart in the constructed grocery store map. Since a precise location of the RoboKart is required for both the planning and mapping modules, it was quite challenging for us to deploy a reliable localization method for our prototype. Localization using wheel odometry was not considered in this application because errors can accumulate as the robot is motion due to dead reckoning. Additionally, feature-based visual odometry cannot be performed based on our design choice of using LiDAR for the perception module. Therefore, to facilitate the performance of the other modules, the ground truth pose information of the RoboKart is used in the localization module for the current prototype.

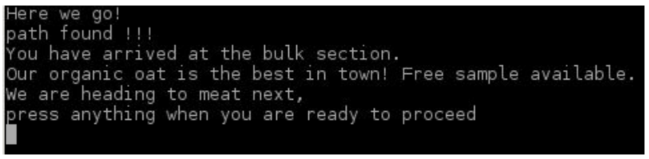
4.5 User Interface

The user interface of our prototype is demonstrated via terminal prompts and prints. In other words, the user will provide input (e.g., using keyboard) to the RoboKart and the RoboKart will talk back (e.g., by printing text messages in the terminal) to the user in the initialization state, when it reaches a food section, and when existing. Examples of such interactions are shown in Figures 8, 9, and 10 below.



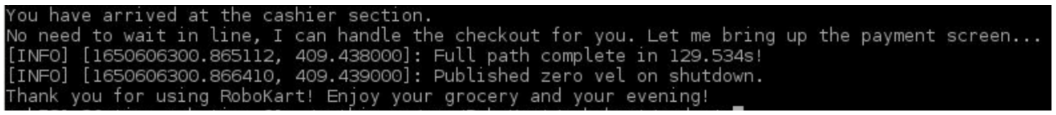
```
Enter the number that corresponds to the grocery category you would like to shop,
or a comma separated list of numbers

    1: grocery
    2: seafood
    3: frozen
    4: deli
    5: produce
    6: meat
    7: bulk
    8: bakery
    9: dairy
    10: florist
    11: softdrinks
    12: cashier

6,3,7,9
we are going in the order of
['bulk' 'meat' 'dairy' 'frozen']
press anything to confirm
```

Figure 8: RoboKart asks for a list of food categories at the initialization state.



```
Here we go!
path found !!!
You have arrived at the bulk section.
Our organic oat is the best in town! Free sample available.
We are heading to meat next,
press anything when you are ready to proceed
```

Figure 9: RoboKart informs the user when a milestone is reached, prints advertisement for the current food section, and waits for the user's confirmation to proceed.



```
You have arrived at the cashier section.
No need to wait in line, I can handle the checkout for you. Let me bring up the payment screen...
[INFO] [1650606300.865112, 409.438000]: Full path complete in 129.534s!
[INFO] [1650606300.866410, 409.439000]: Published zero vel on shutdown.
Thank you for using RoboKart! Enjoy your grocery and your evening!
```

Figure 10: RoboKart finishes and exists after it reaches the cashier and the user's payment is confirmed.

4.6 Deploy RoboKart in the Real World

Our simulation environment is one-to-one in size to real-world grocery stores. Our planning and perception modules can be utilized in the real world environment with very minor modifications (i.e., setting up different parameters for the local planner, and use different floor plans, etc). That being stated, there are four points to consider before deploying RoboKart into a real-world environment.

- **Localization**
  The work-around that we used to solve the localization problem in the simulation is making use of the ground truth location of the RoboKart from *Gazebo*. However, a SLAM algorithm is necessitated to localize the RoboKart in a real-world environment where the true location is never perfectly known. We have tested the AMCL [13] from the ROS navigation package in the simulation environment. RoboKart was able to navigate to the assigned goal location even with static obstacles were placed on the path. Therefore, we consider adopting the AMCL when we scale up the RoboKart.

- **Keep track of individual item's location**
  To demonstrate our prototype, we gave a general coordinate to every food sections rather than specific item locations. When RoboKart is placed in a real grocery store, we will enable its individual item's location tracking by integrating with the store inventory logistics. Then, RoboKart will be able to lead the customer to the precise location of the item within the food section.

- **A real user interface**
  We used the terminal as the user interface for the purpose of demonstrating the human-robot interaction aspect of RoboKart. In the real world, however, we will attach a touch screen that will serve as the user interface. To improve the user experience, more visual design and other user applications may be incorporated and integrated on the touch screen.

- **Accessibility and safety**
  In the simulation environment, the maximum linear velocity is capped by the maximum linear velocity of turtlebot3 waffle Pi, which is 0.26m/s. In reality, accessibility and safety certainly require more attention. Some customers may demand a lower speed so that they can catch up with RoboKart and some may demand shorter travelling time. Therefore, RoboKart must take the customer's demand into account when planning the trips and choosing the velocities. In addition, safety is another concern because shopping carts are usually heavy moving objects whose velocity must be carefully controlled. We will study the collision impact versus the momentum of the shopping cart to ensure only safe velocity commends will be sent to RoboKart.

5 Verification, Validation, and Evaluation

In this section, we outline how we determine if the design meets the needs of the clients and users. In particular, we break down the verification, validation, and evaluation of the design according to each of its component. We assume the actuation, localization, and tracking subsystems are thoroughly verified and validated. Meanwhile, we assume the store keeps track of where each item is (similar to IKEA [9]), and we have access to the precise location of all desired items in store. As a result, we focus our attention in evaluating the perception and planning modules. Specifically, we evaluate them in a simulation environment built by *Gazebo* as depicted in Figure 11.

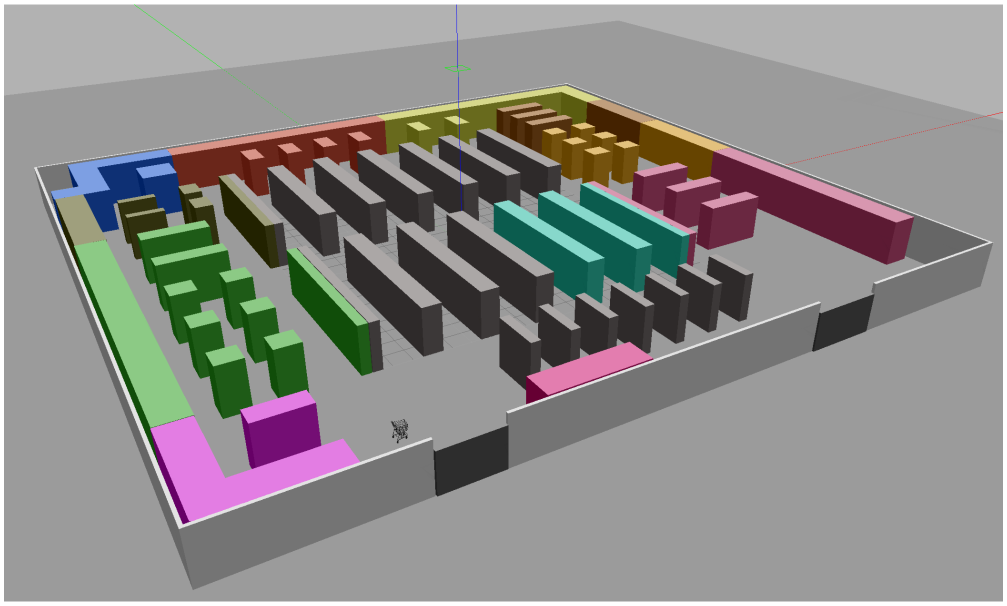


Figure 11: Simulated environment built in *Gazebo*. Colours represent different categories in the store (e.g. blue for seafood).

5.1 Perception Subsystem

We randomly place obstacles around RoboKart which it is not aware of from the given occupancy map of the store. From the new occupancy grid map built from the LiDAR sensor readings, we evaluate whether it is able to correctly detect and localize the obstacles. We have run 20 tests including edge cases for small and large size obstacles as well as placing them close and far away from the robot. RoboKart is able to successfully detect and localize obstacles in all the tests. We acknowledge the result of this "nearly perfect" perception module is partly due to the precise localization information of the robot in a simulated environment and many challenges could have possibly been faced in a real-world scenario.

5.2 Planning Subsystem

The store shelves are static obstacles in the map which the RoboKart is aware in advance. It is only allowed to move in the barrier-free region in the map. We divide the evaluation of the planning module into three phases and include metrics in each phase as follows. Note that the desired performance on each metric is capitalized and bolded.

5.2.1 Phase I: An Empty Store

To begin, we evaluate the RoboKart performance in a customer-free store which will be simulated by an environment with pre-known barriers as shelves. We randomly select two locations in the map where the first one is the starting location of the RoboKart (i.e. where it is currently situated in the store) and the second one is the goal location (i.e. where the desired item is in the store). Then, the RoboKart is expected to plan a route from the starting position to the goal location. The metrics include the following,

1. Binary (**Y**/n): if a path is found from start to goal
2. Binary (y/**N**): if the path found collides with any of the barriers (store shelves)
3. Binary (**Y**/n): if the path found is optimal (shortest)
4. Seconds ($< 1s$): time taken to find the path

We have run a total of 20 test cases and all cases are successfully completed without collisions. Meanwhile, all paths found are optimal. A simple completed test case is shown in Fig. 12.

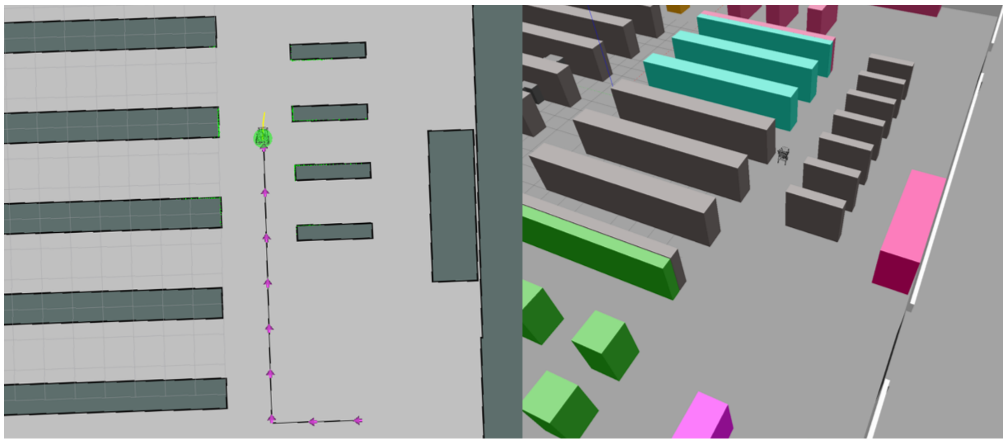


Figure 12: An example of completed waypoints (left) with empty store (right).

5.2.2 Phase II: A Static Store

In the second phase, we build on top of the simulation environment mentioned above and add randomly generated obstacles in the scene which the RoboKart is unaware until it travels near them. These obstacles simulate the static instances in the aisles of a store which could include stationary customers, storage boxes, and fallen items from the shelves. The local planner is expected to adjust its pre-planned path and react to the obstacles accordingly. The additional metrics include,

1. Binary (y/**N**): if the path collides with any of the obstacles (stationary customers)
2. Ratio (0 to **1**): (time taken to reach the destination if the obstacles are all pre-known) / (actual time taken to reach the destination)

We have run a total of 30 tests in Phase II and most of them are extreme cases. One sample of such is shown in Figure 13. For instance, we place two obstacles beside each other on the preplanned global path such that there is very limited space for RoboKart to move in between but sufficiently large space for it to go around and proceed. It turns out that RoboKart takes a more conservative approach to replan its global path to move around the obstacles, arriving at goal safely but a bit later than expected. RoboKart gets an average of 0.8526 score on the second metric mentioned above, largely due to the replanning involved for many of the test cases. If all of the obstacles are pre-known, the global planner will find a path that is collision-free to travel to the goal location. However, in our test cases, many of the aisles are completely blocked by obstacles which the robot gets to know and replan its route only if it travels near it. Therefore, there is some extra time and distance travelled trying to move back and go around. It is worth mentioning that RoboKart completes 29/30 test cases without collision. The only failure case is when it is surrounded by obstacles and the robot decides to rotate itself without linear motion rather than attempting to squeeze through obstacles.

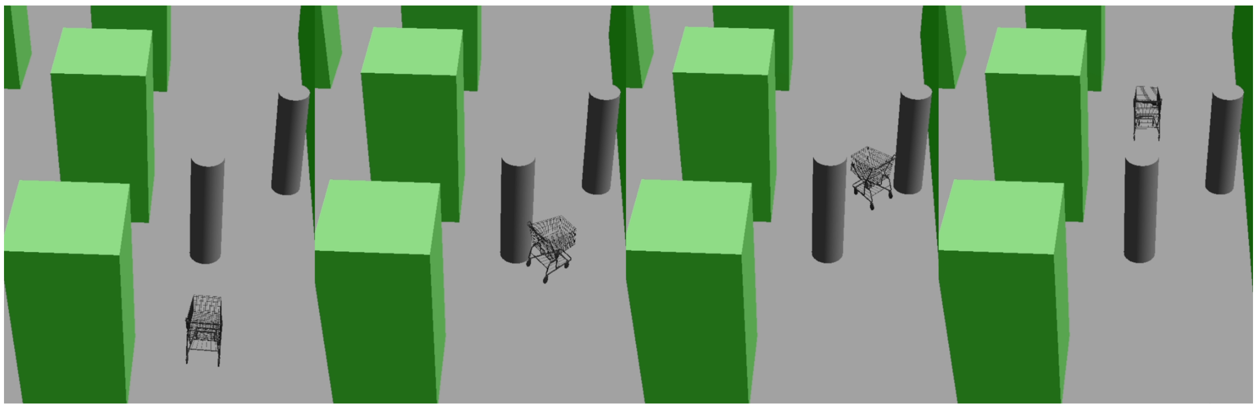


Figure 13: An example of simulated test case with static obstacles.

5.2.3 Phase III: A "Real" Store

In the third phase, the simulated environment is highly analogous to the store where we add dynamic agents. We built a customized script to spawn dynamic obstacles in the *Gazebo* environment. The number of obstacles and their initial location can be specified. One sample of the simulated environment with randomly generated dynamic instances is shown in Figure 14. Each dynamic agent with a random velocity simulates the customers in store. The local planner of RoboKart is expected to avoid colliding with any moving and non-moving obstacles in the environment. We use the same metrics mentioned in Phase I and II. Specifically, we spawn 20 dynamic obstacles with random velocity around RoboKart. We have run 10 test cases and RoboKart is able to reach its destination in 9/10 cases tested without collisions. The only case that it collides is one of the dynamic obstacles runs into RoboKart!

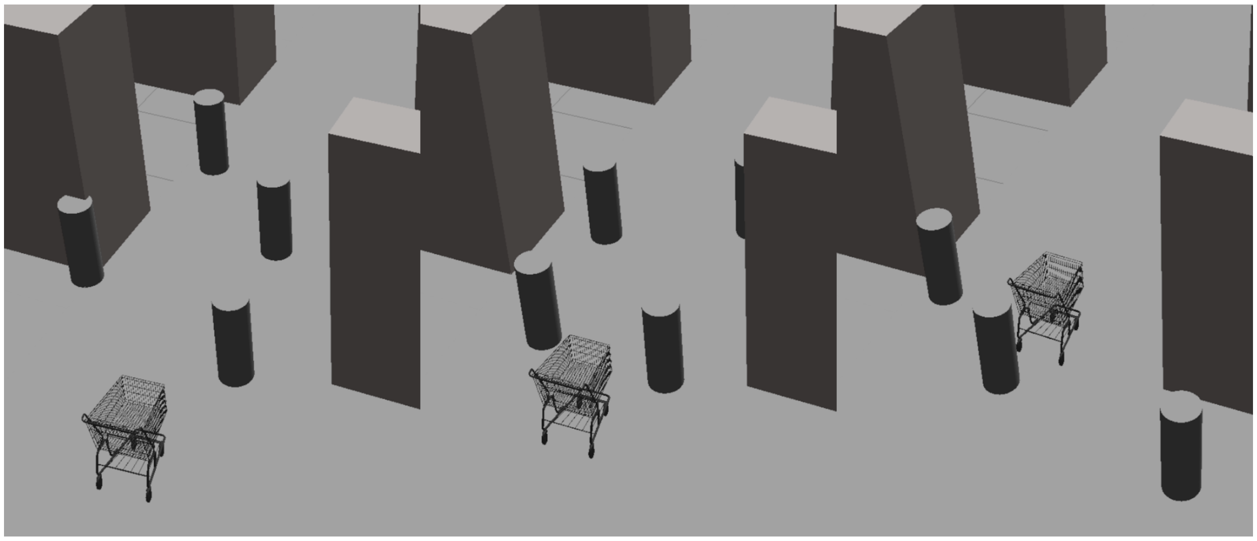


Figure 14: An example of simulated environment with crowded dynamic obstacles.

6 Lessons Learned

Just as other design projects, we have encountered many difficulties during the design process and have taken many detours. This section will highlight the challenges that we have experienced and provide advice on how to deal with them if they ever happen again.

6.1 Trial & Error and Theoretical Background

To allow the local planner to perform adequately, for instance, making the RoboKart go around the obstacles, we spent an enormous amount of work tweaking the cost function (i.e., the heuristic). We experimented with distance-to-obstacle, velocity penalty, and heading error in addition to distance-to-goal. To tune the heuristic, we were mostly using trial & error to fine-tune the heuristic by experimenting with different weights and costs combinations. The RoboKart will sometimes behave as expected, but it will also fail in some edge cases, or it will function in edge cases but fail in regular situations (i.e., going straight with no obstacle in front). Although our prototype performed adequately during the demonstration, the local planner may still be insufficiently robust because there may be other edge cases that we did not evaluate.

In this case, since trial & error is time-consuming and the robustness of the outcome may be questionable, we recommend tweaking the heuristic using methods that provide theoretical guarantees. The significance of heuristics in addressing dynamic programming problems needs no more emphasis. Suboptimal solutions to the trajectory rollout problem can be produced by a good heuristic. To obtain such a heuristic, we suggest using reinforcement learning methods rather than trial & error. Given enough training episodes and simulation efforts (which may be extensive), the result should be robust enough to handle practically any instance, and presumably less unpleasant than manually trying out all the weights and costs combinations.

6.2 Debugging Efficiently

Prototypes are unlikely to run successfully the first time. Hence, debugging is an essential skill to have. While we are all familiar with the concept of debugging, what is more vital in terms of working in fast-paced development environments is debugging efficiently.

Before we talk about tips for debugging large coding project, we recommend using the debugger appropriate for the programming language used. PyCharm, for example, is a Python script IDE with a great debugger.

Moving on to the debugging tips, we recommend modulating the code before any coding takes place. Meanwhile, thoroughly test each module of the code before merging everything together. This aids in locating the troublesome section of the code and concentrating on troubleshoot that module. Another suggestion for debugging ROS is to ensure that the code executes inside the time limit of the control rate. To put it another way, the loop time should be at least as quick as the control rate. For example, if the control rate is 5Hz, which means that control is sent to the robot every 0.2 seconds, and the loop time is longer than 0.2 seconds, the robot may act strangely.

6.3 ROS Environment

Setting up a customized system environment is always tedious and painful, especially running ROS and *Gazebo* simulations on a non-Ubuntu operating system. We built a Docker-based ROS virtualization. The difficulty with this option is that owing to dependency issues, we were unable to install any other ROS packages. We were unable to install ROS map_server and hence could not publish any map-related information. We then used VMWare to construct another virtualization. In comparison to the Docker, this option necessitates additional steps and higher hardware requirements (e.g., CPUs, GPU, and RAM). We were able to install ROS and the necessary packages from scratch. The main issue with this choice is that it has a high hardware demand, which causes the system to operate quite slowly if the machine is very outdated.

Ideally, to work on projects like this, we recommend the team to own a machine with native Ubuntu OS or request access to the lab's computers to save the pain of setting up the environment. Nevertheless, it would only add to one's expertise dealing with these technological challenges if one chooses the painful way. Therefore, take our suggestions with a grain of salt when weighing alternatives.

6.4 Agile Development

Communication among the team members, as well as with mentors and clients, is critical to project success, and client and team satisfactions. We believe we could have done better if we had spent more time making continuous progress throughout the project duration, which would have provided us with more opportunities for frequent feedback. As a result, we believe that by adopting an agile development process, teams may work towards better outcomes.

The agile development process enables the team to 1) swiftly exchange design values, 2) reduce development risks, 3) evolve continuously, and 4) increase client satisfaction. Agile is exactly a excellent example of a mitigation strategy since it allows for better flexibility in various elements of the project to continually adjust the scope of requirements, modifications, and priorities, etc.

Acknowledgement

We would like to thank Professor Jonathan Kelly for the constructive feedback and fruitful discussion, helping our team to frame the project. We also owe a huge thank you to our mentor Michael for the continuous guidance and support.

References

[1] R. Redman, “Customer satisfaction fell at supermarkets in 2020,” Oct 2021. [Online]. Available: https://www.supermarketnews.com/issues-trends/customer-satisfaction-fell-supermarkets-2020

[2] “Retailers losing billions in revenue due to long lines,” Sep 2021. [Online]. Available: https://www.parcelpending.com/blog/retailers-losing-billions-revenue-due-long-lines/

[3] S. McDonald, “Study: Retailers are losing $37.7 billion every year, thanks to long checkout lines,” Aug 2018. [Online]. Available: https://footwearnews.com/2018/business/retail/long-checkout-lines-retailers-revenue-loss-1202554602/

[4] “Grocery stores have seen a significant sales decline as people tire of eating at home,” Jun 2021. [Online]. Available: https://thecounter.org/grocery-stores-sales-decline-as-people-tire-of-eating-at-home/

[5] J. Skorupa, “Robots in retail: Examining the autonomous opportunity,” Apr 2021. [Online]. Available: https://retailwire.com/resources/robots-in-retail-examining-the-autonomous-opportunity/

[6] “Smart shopping carts at 7fresh,” Nov 2019. [Online]. Available: http://www.retail-innovation.com/smart-shopping-carts-at-7fresh

[7] J. Constine, “Meet caper, the ai self-checkout shopping cart,” Jan 2019. [Online]. Available: https://techcrunch.com/2019/01/10/caper-shopping-cart/

[8] S. Lund, A. Madgavkar, J. Manyika, S. Smit, K. Ellingrud, and O. Robinson, “The future of work after covid-19,” Sep 2021. [Online]. Available: https://www.mckinsey.com/featured-insights/future-of-work/the-future-of-work-after-covid-19

[9] “How to check product stock availability.” [Online]. Available: https://www.ikea.com/ca/en/customer-service/stock-availability/

[10] P. E. Hart, N. J. Nilsson, and B. Raphael, “A formal basis for the heuristic determination of minimum cost paths,” *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[11] E. W. Dijkstra, “A note on two problems in connexion with graphs,” *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[12] S. M. LaValle, “Rapidly-exploring random trees : a new tool for path planning,” *The annual research report*, 1998.

[13] F. Dellaert, D. Fox, W. Burgard, and S. Thrun, “Monte carlo localization for mobile robots,” in *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, vol. 2, 1999, pp. 1322–1328 vol.2.