

Building a navigation system for a shopping assistant robot from off-the-shelf components

Kenny Schlegel, Peer Neubert, and Peter Protzel

Chemnitz University of Technology, Chemnitz, Germany
`kenny.schlegel@etit.tu-chemnitz.de`

Abstract. A primary goal of developing robots is to relieve people from hard work or help them in difficult situations. An example for such a situation is a shopping assistant robot that supports people who need help in their daily life. Such a mobile robot has to be able to deal with dynamic environments (moving people) and navigate safely and efficiently. In the present paper, we provide a system description of our navigation strategy for an assistant robot that is developed for autonomous operation in a supermarket. Creating an appropriate experimental environment can be quite challenging, and access to the application place can be limited (e.g., a real supermarket). Therefore we complement our real robot with a digital twin and describe our approach to create a suitable simulation of a real-world supermarket. Further, we discuss how off-the-shelf software can be used to implement a three-stage navigation strategy (planing a route with a TSP solver, global path planning, and incorporating dynamic obstacles in an optimization-based TEB approach) that is suitable for environments with dynamic obstacles. The paper presents our approaches to create a 3D map from 2D floor plans, as well as the preprocessing of the sensor data for usage in the TEB planner. Finally, we provide our hands-on experience with implementing a complex state-machine using the graphical RAFCON framework.

Keywords: mobile robots · dynamic environments · navigation.

1 Introduction

As a result of the technical progress, robots are going to be more widely used not only in industrial fields but also in people's daily life. They can be designed as mobile platforms to make people's lives easier. One application scenario is a shopping assistant for supermarkets. In the context of ambient assisted living, such robots can be helpful for older or disabled people, but also for other consumers shopping can become more convenient. For example, the robot provides product information or their position in the market, works as a guide, or as a shopping cart, which follows a customer. However, a major goal of this development is autonomous shopping: with a list of items, the robot collects them efficiently and bring them to the customer or checkout.

Supported by the Federal Ministry of Education and Research, Germany.

Moving people, shopping carts, and narrow shelves make autonomous navigation within a supermarket quite challenging. The requirements for the robot are high: it should be fast, efficient, and safe. Compared to applications in autonomous logistic centers, a supermarket is less constraint and especially navigation in the vicinity of moving people in an efficient way is difficult. It requires an extensive perception of the environment and sophisticated planning algorithms. But such an unconstrained system offers the opportunity to work in unprepared environments and has a wider applicability. Beside robot application like Home-care, a shopping assistant robot must be fast while ensuring safety, since costumers usually have limited time. This leads to a difficult trade-off between speed and safety. Furthermore, in order to fulfill complex tasks, a shopping assistant robot has to have a large set of skills (autonomous shopping, leading, following, etc.) - the coordination and ordered execution of these skills, as well as monitoring and coordinating the various hardware and software modules are essential. In particular, the exception handling requires a robust robot control architecture.

Although robots designed for such complex tasks are typically customized and vary in a lot of details, the design decisions and challenges when building and programming such a robot are quite similar. In this paper, we present our experiences in building the navigation module for such a shopping assistant robot in order to facilitate the development of other robots for similar tasks. This work is part of an ongoing larger project to create a fully functional shopping assistant robot that is capable of navigating in crowded environments and autonomously collect a list of shopping items. This paper presents the following aspects of our shopping robot¹:

- Sec. 2.1 provides a short presentation of the custom made robot and its sensor layout designed for the complete autonomous shopping task.
- Since navigating a larger robot in the vicinity of humans is dangerous, extensive testing and evaluation of the navigation algorithms is mandatory. Sec. 2.2 describes how we complement the real robot with a digital twin in a simulated supermarket environment. This simulation can later also be used for visualization purposes for customers.
- To circumvent the (still existing) challenges of SLAM, we present a semi-automatic procedure to create 3D maps from a priori known 2D floor plans in Sec. 3.1. These maps are used to create the simulation, as well as for navigation of the real robot.
- The core navigation capabilities result from a combination of three off-the-shelf components: we use a state-of-the-art optimization-based time elastic band planer in combination with a TSP solver and a global A* planner to approach navigation tasks in supermarket environments (see sec. 3.1).
- Sec. 3.2 presents hands-on experiences with the recently published RAFCON [2] framework to create, run, and monitor state-machines (in particular, hardware-sensor monitoring).

¹ Videos and other supplementary material can be found on our website <https://www.tu-chemnitz.de/etit/proaut/shopping-robot>

The purpose of the paper is to provide an example of a navigation system to support researchers and developers who want to design a robot for navigation in dynamic environments, especially those with humans. The list of similar examples of robots is not very long. A historical milestone and the first mobile robot that could navigate autonomously was Shakey the robot [14]. Although we still use a variant of Shakey’s A* planner in our system, since then, technology and software have been improved, and further applications became possible. There were mobile robots in the field of human interactions, like the RHINO robot [3], which works as an interactive tour guide in a museum or the LINO robot [10] as an example of a domestic user interface robot. Based on these early-stage systems and more recent developments, applications of service robots in more complex environments became possible. For instance, the SPENCER robot [21] can support people at a large airport. Such a system is similar to our shopping assistant robot since both operate in dynamic environments with moving people. SPENCER uses multimodal people tracking with a laser scanner and RGB-D cameras to estimate people’s motion and accordingly adapt the planned paths based on an RRT* planner. Also related is the STRANDS [9] project that deals with long-term autonomy in dynamic environments. Cheng [4] describes another system in a supermarket environment that can grasp items and put them into the bag. They used an A* algorithm to compute waypoints to navigate the robot. However, since the main task is focused on grasping, the navigation strategy does not provide specific dealing with dynamic obstacles.

All these presented references have individual requirements and different implementations. Although our system shares some properties with the existing approaches, there are significant differences that are supposed to provide additional insights and guidance for developers of new systems. In particular, we see value in the parallel usage of 3D maps created from floor plans in simulation and real-world navigation, as well as the integration and experiences with the recently presented TEB planner as well as the RAFCON framework. Although we build on off-the-shelf components, they require modifications and extensions to work in combination to approach the challenging task of navigation in supermarket environments.

2 System Components

In the following two sections, we will describe the system components, starting with the hardware platform, followed by our simulation environment.

2.1 Robot Platform

The complete robot system prototype is visualized in figure 1. It shows the real system on the left and its digital twin for our simulation on the right (described in section 2.2). The robot is based on an omnidirectional wheel drive. This is an important design decision since it significantly simplifies acting in a narrow and crowded environment like a supermarket, in particular since picking items from shelves requires small motions parallel to the shelf. The omnidirectional wheel drive allows us to perform them without additional rotations of the platform. The size of the platform is limited to the dimensions of a regular shopping cart. For perception, we use the following sensors:

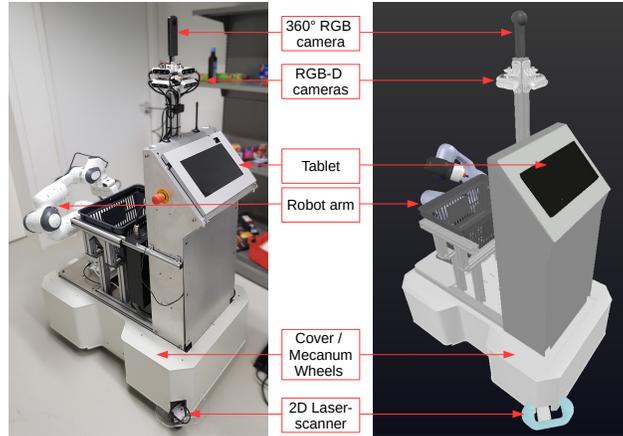


Fig. 1. left: the real robot system, right: the digital twin in simulation environment.

- Two 2D laser-scanners (Hokuyo UTM-30LX-EW), which are located on diagonal corners of the case to get a 360° field of view. They are used for localization and collision avoidance. They are mounted very low (6cm over the ground) in order to perceive the bottom panel of the supermarket shelves.
- One inertial measurement unit (IMU MTI-3-8A7G6-DK) located in the center of the robot to improve the wheel odometry based on wheel encoders.
- In total, there are eight RGB-D cameras (Realsense D435), arranged in two horizontal rings of four cameras to allow 360 degree perception and a large vertical field of view.

The main exteroceptive sensors for navigation and collision avoidance are the laser scanner and the lower ring of RGB-D cameras. The second RGB-D camera ring is primarily used for interaction with potential customers by recognizing intentions with an eye gaze tracking system described in [13]. Although it is part of this shopping robot, this interaction system is beyond the scope of this paper. Eight RGB-D cameras produce high data traffic on the hardware. Grunnet-Jepsen et al. [8] provide a system description with up to four cameras. However, our experience shows that it is possible to use simultaneously eight D435 cameras with a PCIe extension for USB3.0 (DELOCK 89297 PCIe card mounted in an Aprotech GOLUB 5000 i7 industry PC) if the resolution and frequency are set to a mid-level (480x270 pixel depth resolution and 15 fps).

Furthermore, on top of the RGB-D camera rings, there is a 360 degree RGB camera to detect people and their skeleton in images. In addition, the robot system also has a tablet to interact with users, which is located in an ergonomic position, and a 'Panda' robot arm from the company 'Franka Emika' to grasp objects and put them into the basket. However, the components that are not related to navigation are not further discussed in the present paper.

2.2 Simulation environment

Since it provides a more accessible test environment in contrast to a real supermarket, we use a simulation environment to develop and evaluate the algorithms

parallel to the real robot platform. We decided to use V-Rep [17] as simulation software² based on the publication [15] that compares three different simulation frameworks - V-Rep has the overall best evaluation. V-Rep has the advantages of a large object library, for example people who can be static or dynamic (well suited for supermarket environments) and the simple ROS compatibility with an appropriate plug-in. To create the 3D model of the simulated supermarket, we evaluated two approaches:

1. Using 3D point cloud SLAM We did a 3D measurement of a real supermarket with another robot [11] that is equipped with the 360 degrees laser scanner from [20] to produce a real representation of the market. After getting several 3D scans of different places in the supermarket, we fused these to one map with the ICP registration algorithm and aligned RGB images with the 3D point cloud to obtain a colored point cloud representation of the market. Afterward, we created a mesh from the points to use it in the simulation. Figure 2 shows the result of the point cloud and the constructed mesh (for visualization, the image contains both - from left to right is going from the point cloud to the mesh (shown in gray tones)).

2. Creating 3D maps from available 2D floor plans The target shopping assistant robot is developed in cooperation with a supermarket company. Thus, we are in the comfortable position to have floor plans available that also include a layout of the larger furniture (e.g., the shelves). If such robots are deployed in real supermarkets or other public places for practical usage, this will presumably also be based on corporations with the owners of these places; thus, the availability of floor plans is a reasonable assumption.

To create a 3D model with primitive geometric objects from the 2D floor plan, we conduct the following steps:

1. We use standard image processing tools to transform the floor plan to predefined color code for the different semantic objects and categories (e.g., the floor has the color white, all shelves of a particular type have another color value, and so on). This requires some manual configuration and supervision.
2. Based on our previous work [16] we can create a 3D model automatically from the 2D color-coded image (basically, each area in the 2D image with a specific color becomes a 3D object with the corresponding predefined height).
3. Finally, texturing all these primitive geometric objects with supermarket images creates the environment visualized in figure 3.

When creating the simulation environment based on the mesh obtained from 3D SLAM in the real supermarket, we faced three challenges: First, error correction and loop closure detection in our point-cloud SLAM required some careful human intervention. Second, the created mesh needs a high effort to smooth some irregularities and reducing the complexity of the surface. The third was the high resolution of the created mesh, which made the simulation very slow and increased the computation time for rendering and sensing. Although all three challenges can be addressed with more sophisticated algorithms and more manual intervention, we found the alternative approach based on the 2D floor

² version 3.5; latest version (CoppeliaSim) does not work with the used V-Rep plug-in



Fig. 2. The result of the supermarket pointcloud and its resulting mesh (left to right is going from points to surface).



Fig. 3. The simulation environment of the supermarket with V-REP.

plans to be preferable: The 3D representation based on simple geometric objects allows a sufficiently detailed representation of the static structure of the market while it is still possible to simulate with low computational effort. The high-resolution 3D model provides a very detailed and realistic appearance of the real world but is not suitable for a simulation with V-Rep. For instance, the simple 3D map has with 2500 vertices only 0.02% of the size compared to the high-resolution mesh visualized in fig 2. Even if the detailed mesh is reduced in size, the simulation with the simple 3D model is still resource-hungry. For instance, simulating with a frequency of 10Hz on an i7-8550U CPU is possible in real-time but requires a complete CPU kernel. The scripts of the sensors (laser and camera data-processing) need the most CPU power.

3 Software Components

The presentation of the software architecture is again divided into two sections: 3.1 provides information about the navigation software and 3.2 the concept of our state control (state machine). The implementation uses ROS in version Kinetic.

3.1 Navigation

Where I am? (Localization) As described in section 2.2, we create a map from available floor plans for simulation. We use the same map to create an occupancy grid map for the localization of the robot in the supermarket. In combination with the well known Monte Carlo localization from [7] and the available ROS implementation³ we are able to localize the robot using the 360 degrees 2D laser scans. The repetitive structure of supermarkets makes the initial global localization particularly challenging. The traversal of a relatively long route might be required to solve localization ambiguities due to visual aliasing. We circumvent this problem by assuming a known start position of the robot. In practice, this could be, e.g., the power charger station. Additionally, we improve the motion estimation of the wheel odometry by combination with IMU measurements in a Kalman filter based on an available ROS-implementation⁴. We did experiments in a real supermarket (with the robot from [11]), which showed that it is possible

³ Brian P. Gerkey, AMCL, ROS Wiki, <http://wiki.ros.org/amcl>

⁴ Tom Moore, robot_localization, ROS Wiki, http://wiki.ros.org/robot_localization

to hold the localization even if the initial floor map is sometimes unrelated to the real measurements (in particular, there were unknown and temporary shelves in the market).

Where do I need to go? (Planning the route) In the autonomous shopping use-case, the user selects several items on the tablet and it sends a list of articles and their positions to the robot. Then, the robot has to find the shortest route to collect all items as quickly as possible. This optimization problem is called 'Traveling Salesman Problem' (TSP) and is intensively studied in particular in the field of operations research. An approximate solution can be achieved with the nearest neighbor method described in [5]. It is a deterministic heuristic procedure that starts at one point and selects the next item based on the smallest distance to all other open items. This greedy algorithm produces only an approximated solution and not necessarily an optimal order of all items. Besides the nearest neighbor method, there are also other solvers of the TSP. For instance, an optimal TSP solver called CONCORDE⁵, which outputs the item order with the shortest entire path. We use this solver in combination with a ROS package⁶ for our supermarket robot system. Since the computing time for exactly solving the TSP can increase exponentially, it is important to know when it gets too time-consuming and how are the difference of the lengths of the entire paths compared to the approximate solver. Experiments in our supermarket environment with 20 items show that the exact Concorde TSP solver needs roughly 500 ms more than the approximate TSP solver (nearest neighbor) with less than 1 ms. A significant growth of computing time with the Concorde solver starts at round about 60 to 80 items. Additionally, it has to be noticed that the most time-consuming step is constructing the distance matrix. An A* planner calculates distances between all possible points and saves these in a matrix. This matrix can be precomputed concerning all item positions in the market and is given while solving the TSP problem to save computing time. However, with a view on the average number of items that are bought in the supermarket per stay, we recognize that the computing time of the solver is negligible - only an average of 10 items are purchased per visit⁷. Finally, a comparison of the entire path lengths indicates that between 10 and 60 items, the Concorde solver calculates a path that is 15% shorter than the path from the nearest neighbor approach. Based on these significant differences and the relatively short computing time of the exact solver within the range of typical human behaviors, the exact TSP solver can be used for creating a global order in the market. One example of the route planning with 16 items can be seen in figure 4 on the left image - it is the first step of navigation planning.

What is the fastest way to the next item? (Path and motion planning) If we have the correct order of all items, we have to navigate between the

⁵ Applegate, D. et al., "Concorde tsp solver". <http://math.uwaterloo.ca/tsp/concorde>.

⁶ Richard Bormann, ipa_building_navigation, http://wiki.ros.org/ipa/_building/_navigation.

⁷ Statistic from our cooperation partner of the supermarket

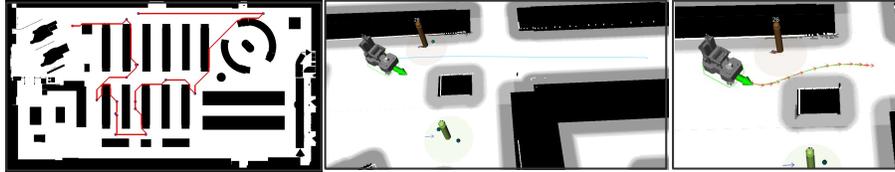


Fig. 4. Visualization of the three steps of navigation - left: calculate the route with TSP solver to collect all items; middle: result the global plan to the next item with A*; right: motion planning with TEB. Pillars are recognized people.

subgoals. The whole path and motion planning of our system is based on the ROS navigation stack⁸. We use an A* planner on the grid map from the given floorplan of the supermarket (enhanced with current sensor data) to get the global path from the current position to the next item according to the results of the TSP (visualized in the middle image in figure 4 with a blue line as global path). After having our global path from the current position to the next item, we need a planner that reacts to unknown obstacles (static objects which are not in floor plan) and dynamic objects like people. A local planner can realize such behavior - it uses the global plan and adopts it regarding currently seen obstacles. In the literature are existing ready-to-use algorithms; for example, the dynamic windows approach (DWA) planner [6]. It is also part of the ROS navigation stack. The DWA planner is a sample-based trajectory generation algorithm that samples a set of velocities for translation and rotation (concerning the dynamic constraints of the robot). These hypotheses are evaluated with the so-called 'trajectory rollout' that simulates a path for all possible velocities and compares them with the current environment for potential collisions. It results in a score, and the best trajectory for the next time step can be chosen. A disadvantage of this planner is that it cannot predict a more complex path to avoid collisions of dynamic obstacles in the future - it is rather a reactive planner with a constant velocity model.

A more suitable planner to incorporate dynamic obstacles is the more recent *TEB* (*time elastic band*) planner from [19] and [18] with its ROS implementation⁹. It is an online optimization algorithm based on a hypergraph (factor graph) and optimizes towards a minimum execution time. All nodes within the graph are waypoints (sampled from the solution of the global A* planner), and all edges represent the penalization functions (e.g., close to an obstacle). The current implementation provides an extension to incorporate dynamic obstacles and adapt the trajectory with the optimization of the factor graph. It enables long term planning and creates a path that avoids any collisions (static or dynamic) efficiently. In addition, the implementation of the TEB planner also includes the calculation of the velocities (path execution). Therefore, we chose the TEB planner for our local navigation within the supermarket.

⁸ Michael Ferguson, navigation, ROS Wiki, <http://wiki.ros.org/navigation>.

⁹ Christoph Roesmann, *teb_local_planner*, http://wiki.ros.org/teb_local_planner.

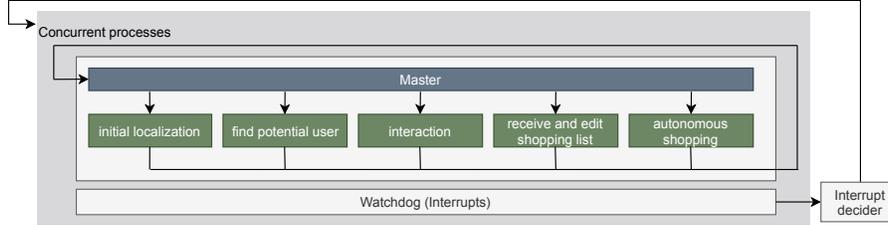


Fig. 5. High-level concept of state control within the statemachine.

However, it is essential to preprocess and provide the map and sensor data in a suitable way. The TEB planner uses two inputs of obstacles: one *static* and one *dynamic*. A local costmap, which is a gridmap with occupancy probabilities, represents the environment for static navigation. In contrast, a list of object descriptions contains all dynamic obstacles with properties like position, velocity, and appearance (shape). We developed a pipeline to separate our environment perceptions, which is based on the laser scanner and the RGB-D cameras. The first step is to detect and locate humans in our environment. As described in section 2.1, we are planning to use a 360 degrees RGB camera to detect people around the robot. Since it is work in progress, we use a simple laser scanner detector from [12] that can locate people in a 2D laser scanner (similar to the approach of [21] and [9]). In combination with a Kalman filter, we can use it as a basic human tracker. Another way is to use the ground truth positions from the simulation environment - it provides an error-free evaluating of the navigation algorithms. Once we have all the positions of the people in the environment, we can go to the next step: create the **dynamic obstacle** description. Based on the information about the position and velocity of the laser tracker, we used these to create the description for the TEB planner.

The third step of our preprocessing pipeline is constructing the appropriate **static costmap**. It should contain no obstacles, which were already described in the obstacle message (including the people). To do so, we have to remove the points assigned to humans from the laser scan and RGB-D point cloud (we denote the input point cloud as P). We did this by grouping the point cloud P with an euclidean clustering. Afterward, we calculated for all clusters the centers and compared them to the given people positions. If one center is closer than a predefined threshold, it is marked as assigned and removed from the point cloud P - the result is a non-person point cloud P_{nP} . Notice, such a threshold-based method can be sensitive to the selected value (hyperparameter) and can remove clusters that are not related to a person (e.g., a person is standing close to a wall). To prevent such behavior with the point-cloud clustering, we set a maximum number of points per cluster - thus, merging of people and non-people objects becomes less likely, and more importantly, less harmful (since only small additional set of points is removed).

Now, the last step is to create a costmap from point cloud P_{nP} . For that, we use a so-called 'Nearfieldmap' from [22], which accumulates the input (our P_{nP}) over a specific time interval in a discretized 3D representation (voxel) -

it prevents outliers from camera noise. We recognized that it would be more suitable to apply the clustering separately to the laser and the cameras, and fuse these two data into the nearfieldmap. It ensures a better parameterization of the sensor data probability (the laser is more reliably than the cameras). After a projection to the 2D plane, we obtain a 2D costmap with all non-person obstacles and can use it as the input of the TEB planner. Besides, methods to directly remove the people from the 2D costmap did not work because regions of the people are sometimes connected with other non-people objects (through projection from 3D to 2D). It leads to removing objects like walls or other obstacles if the person is too close to it.

With such a preprocessing pipeline, it is possible to use the TEB planner for navigation in a dynamic environment with people. Figure 4 visualizes this third step on the right side with a local path shown in green (robot avoids the person). Further results and visualizations can be found on our web-page¹.

3.2 State control

After describing the robot and its appropriate simulation environment and creating a navigation strategy, we want to provide our hands-on experience concerning state machine design. Such a task control software is necessary to control the order of all tasks and subtasks (e.g., localization, global planning, or reaction on inputs by the customer). One well-known framework to create a state machine within ROS is SMACH [1]. It is based on python scripts that execute specific tasks and return values indicating success or failure. One disadvantage of SMACH is the lack of tools to reduce the complexity of the state machine. Debugging in case of errors is hard and strongly benefits from a visualization of the states. Even though SMACH provides a graph visualization of all states and their connections, it cannot be used to intervene when the machine is running, and editing of the code is only possible in the concerning python script, not in the visual graph.

Another recently published framework for efficient and transparent programming of a state machine is RAFCON [2]. It is a graphical tool to construct hierarchical tasks and allows real-time intervention and monitoring. All states contain a Python script that represents the executed code if the state is active. Based on the simple and clear designing as well as the monitoring while execution, we decide to use RAFCON to create our state machine.

Figure 5 visualizes the high-level concept of the entire system and how it is divided into several parts. The head of all tasks is the Master state that decides the order of given subtasks which are:

1. initial localization (robot moves to localize with MCL, see 3.1)
2. find potential user (work in progress; currently the nearest person will be selected as a potential user - the robot rotate to it)
3. interaction (the robot starts an interaction with a selected person by the tablet)
4. receive and edit shopping list (the user enters a shopping list for autonomous shopping, and the robot saves it for navigation)
5. autonomous shopping (the robot plans the route and executes it)

Since the project of the shopping assistant robot is not finished yet, the number of possible states will be increased (e.g., following a person) and adopted (interaction state gets more modalities like verbal communication). We want to emphasize that each state, like the autonomous shopping, consists of multiple subtasks summarized into one hierarchical state. Beside the Master and its tasks, the state machine has a concurrent “Watchdog” state that monitors all processes and reacts to interrupts or errors in the system. Additionally, user interaction on the robot mounted tablet could trigger interrupts (e.g., a customer press the stop button) and the state machine has to react on it. If such an interrupt arises, the following state ‘Interrupt decider’ creates a specific code and sends it to the Master that can react (for instance, repeat a state, go to a specific state, etc.). Another important class of interrupts comes from our sensor monitoring software. Basically, this is implemented as a ROS node that regularly checks the sensors concerning their connection, driver and data. If something goes wrong, the node sends an interrupt, and the state machine decides an appropriate reaction; for example restart the driver of the sensor. This monitoring node turned out to be of very high value in a complex robotic system with many sensors and a large number of different soft- and hardware components.

The combination of the presented high-level concept and the RAFCON framework allowed to create a complex state machine with monitoring of the hardware and reacting on signals from the tablet interaction. Based on our experience with both, SMACH and RAFCON, we find that RAFCON provides a better solution for complex task control than SMACH, in particular since RAFCON provides a user-friendly GUI with several options for better debugging (step by step execution, go backwards, set breakpoints, etc.).

4 Conclusion and discussion

The paper provided an overview of the navigation strategy of our shopping assistant robot. We started with describing the system in both the real hardware platform and the simulation environment. We discussed how an available floor plan can be used to create a suitable simulation environment and discussed why we preferred this over a SLAM based approach. Such a simulation benefits from easier access to the test environment and ensures repeatable test scenarios (e.g., people are always moving on the same path and react consistently). After describing the hardware components and the simulation, we presented the software components, particularly the three-stage navigation process that consists of planning the most efficient route with a TSP solver, generate a global path to the next subgoal and plan the motion with the local TEB planner afterward. Using the TEB planner in an environment with people as dynamic obstacles was not described in the literature yet and required a specific preprocessing of the obstacle perception. With our presented concept, we can use the TEB planner in the presence of humans. However, future work includes an extension to the ‘freezing robot’ problem in dense human crowds. The current system is a promising basis for addressing this problem in the ongoing research project. For example, the factor graph representation in the TEB planner provides a general

framework to include additional information about predicted human motions and their uncertainty in the planning process.

Finally, we presented why and how we use RAFCON to implement the high-level concept of handling multiple states (tasks) and monitoring of our sensor hardware as well as the interrupts, e.g., triggered by user interaction on the tablet. The implementation with RAFCON showed beneficial properties like user-friendly GUI, intuitive debugging and rapid development.

References

1. Bohren, J., Cousins, S.: The SMACH High-Level Executive [ROS News]. *IEEE Robotics Automation Magazine* **17**(4), 18–20 (2010)
2. Brunner, S.G., et al.: RAFCON: A graphical tool for engineering complex, robotic tasks. *IEEE International Conf. on Intelligent Robots and Systems* (2016)
3. Burgard, W., et al.: Experiences with an interactive museum tour-guide robot. *Artificial Intelligence* (1999)
4. Cheng, C.H., et al.: Design and implementation of prototype service robot for shopping in a supermarket. In: *ARIS* (2018)
5. Domschke, W., Drexl, A., Klein, R., Scholl, A.: *Einführung in Operations Research*. Springer Gabler, 9 edn. (2015)
6. Fox, D., Burgard, W., Thrun, S.: The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine* **4**(1), 23–33 (1997)
7. Fox, D., et al.: Monte Carlo Localization: efficient position estimation for mobile robots. In: *National Conf. on Artificial Intelligence* (1999)
8. Grunnet-Jepsen, A., et al.: Using the Intel  RealSense TM Depth cameras D4xx in Multi-Camera Configurations. Tech. rep. (2018)
9. Hawes, N., et al.: The STRANDS Project: Long-Term Autonomy in Everyday Environments. *IEEE Robotics and Automation Magazine* **24**(3), 146–156 (2017)
10. Kröse, B.J., et al.: Lino, the user-interface robot. In: *European Symposium on Ambient Intelligence*. pp. 264–274. Springer (2003)
11. Lange, S., et al.: Two autonomous robots for the dlr spacebot cup -lessons learned from 60 minutes on the moon. *International Symposium on Robotics, ISR* (2016)
12. Leigh, A., et al.: Person tracking and following with 2D laser scanners. *Proceedings - IEEE International Conf. on Robotics and Automation* (2015)
13. Lorenz, O., Thomas, U.: Real Time Eye Gaze Tracking System using CNN-based Facial Features for Human Attention Measurement. In: *VISIGRAPP* (2019)
14. Nilsson, N.J.: Shakey the Robot. SRI INTERNATIONAL MENLO PARK CA (1984), <https://www.sri.com/work/publications/shakey-robot>
15. Pitonakova, L., et al.: Feature and performance comparison of the v-rep, gazebo and argos robot simulators. In: *Towards Autonomous Robotic Systems* (2018)
16. Poschmann, J., et al.: Synthesized semantic views for mobile robot localization. *European Conf. on Mobile Robots, ECMR* (2017)
17. Rohmer, E., et al.: Coppeliassim (formerly v-rep): a versatile and scalable robot simulation framework. In: *Intl. Conf. on Intelligent Robots and Systems (IROS)* (2013)
18. Rösmann, C., et al.: Trajectory modification considering dynamic constraints of autonomous robots. *Robotik 2012* pp. 74–79 (2012)
19. Rösmann, C., et al.: Efficient trajectory optimization using a sparse model. *2013 European Conf. on Mobile Robots* pp. 138–143 (2013)

20. Schubert, S., et al.: How to build and customize a high-resolution 3D laserscanner using off-the-shelf components. In: Towards Autonomous Robotic Systems (TAROS) (2016)
21. Triebel, R., et al.: SPENCER: A socially aware service robot for passenger guidance and help in busy airports. Springer International Publishing (2016)
22. Weissig, P., Protzel, P.: Properties of timebased local OctoMaps. In: Workshop on State Estimation and Terrain Perception for All Terrain Mobile Robots held in conjunction with IROS (2016)