Hochschule für Technik
und Wirtschaft Berlin
University of Applied Sciences

Titel der Bachelorarbeit oder Titel der Masterarbeit oder Titel der Diplomarbeit

Bachelorarbeit

Name des Studiengangs
Ingenieurinformatik

Fachbereich 2

vorgelegt von
Lukas Evers

Datum: 23.09.2022

Erstgutachter/in: Prof. Dr. Frank Burghardt
Zweitgutachter/in: Dr. Ahmed Hussein

Abstract

List of Figures

List of Tables

Contents

Chapter 1

Introduction

Research question: **How can behavior planning increase the robustness and autonomy of a robot running ROS2?**

Core Problem - too little autonomy in a standard autonomous mobile robot running ROS navigation(AMR)

Description of the Problem:

Despite the advancements in autonomous driving the robot regurlarly needs an operator in case smth unexpected happens.

This may include: unexpected collisions, a system restart is needed, battery is empty, the environment is too crowded and the robot gets stuck

The system is not able to react and handle unforeseen situations.

This can lead to safety issues during the operation of a robot. These safety issues can lead to uncontrolled driving maneuvers due to wrong sensor data or a wrong environment represantation.

Due to the need of the robot to require an operator for safe, the robot is not really autonomous despite it being labeled as such.

(Difference is defined by the SAE Levels for automonous driving in cars.)

Relevance, need for research:
It is important for the robot to possess the ability to navigate in an uncertain environment to guarantee the safety of the robot itself and all other actors in its environment, these can be other robots and humans.

In theory the robot can perform fully autonomous navigation including mapping and localization but as soon as something out of the ordinary happens the robots autonomy is not guaranteed to be reliable anymore. Default robot is not able to represent all possible fail states and does not detect failures on a systematic level, but only inside its navigation related subsystem. And even when failures inside this subsystems are discovered, the options to handle these problems are very limited and often do not deal with the problems in an autonomous way. The reliance on human-needed problem solving decreases the robots usefulness when tasked with real goals.

Having a robust and safe robot behaviour opens the door to many applications in a more diverse set of challenging environments. Usually a reinforcement learning approach with broad training data set is a great way to handle a multitude of unknown environments and situations, but one of the biggest problems is the lacking determinism in such systems.

In the robotic world exists a need for a deterministic system which can enhance and override the decision making process and navigation capabilities of the robot.

Planned solution:

This thesis will explore an approach how an extension of the current software architecture and behavior planning capabilities for an autonomous mobile robot that is running ROS2 can improve the autonomy of the robot.

Expected results:

The desired outcomes of these implemented extension will be an increase of robustness and decrease of required human actions during a number of scenarios. In these scenarios, failures and and problems are artificially induced to test the robots abilities to behave autonomously.

Research gap is not named so far

Chapter 2

State of the Art

2.1 Automated and Autonomous Vehicles

This thesis looks for a way to improve the autonomy of mobile robots. In order to define what qualifies as an improvement in the autonomous behaviour of a mobile robot, one needs to look at the defintion of automated and autonomy. The EFI defines the term autonomy in the context of robotics as a system which can act without human instructions and still solve complex tasks, make decisions, learn independently aswell as react to unforeseen circumstances [8]. The definition specifies the needed requirements to make a robot fully autonomous. The automotive industry defines vehicle autonomy in levels based on the human driver's need for supervision and possible intervention during the execution of complex driving tasks [1]. Figure xx depicts the different levels of autonomy of passenger cars on a scale between no automation to completely autonomous.

Both definitions entail the reliance on human supervision and guidance as the central part of what hinders a system or vehicle to become autonomous. By decreasing the instances a human intervention during the operation of a robot, one can increase the automation level towards full autonomy. But this has to be done by equipping the robot with robust and context-driven decision-making and planning capabilities, otherwise a robot which would never need a human to operate could easily be created. But this robot could not be considered autonomous as it would not able to solve complex tasks and make intelligent decisions.

Different levels of autonomy require different methodologies to achieve them.

SAE Levels EFI Gutachten Fachforum

2.2 Autonomous Driving Navigation Architectures

To gain a better understanding how behavior planning influences the autonomy of a robot, one can take a look at the latest and most used software architectures for autonomous driving on a functional level. Most of the current autonomous driving architectures are implementing a hierarchical structure that follows the "Sense - Think - Act" paradigm [9]. As shown in figure xx the sensory inputs, usually from multiple sensors, get processed to create an environment representation. Then the system calculates how to get from its current position to the destination and creates a path. A second planning calculation is triggered to compute the motion commands with the constraints of the system (e.g. size, turning radius, etc.). These motion commands then get converted to control the motors. The planning sequence can be further divided into global planning, behavior planning and local planning. Global planning,also named route planning, is responsible for outputting a path from start to goal. This is comparable to a person using Google Maps to find the shortest or fastest path to a far away city. In this analogy, the behavior planner is responsible to follow the traffic rules on the trip to the destination, e.g. stopping at red lights, giving right of way to other vehicles, following the speed limit. The local planning, also named motion planning, module takes the input from the behavior layer and has the task to let the vehicle drive in accordance to the executed behavior, so that it stays in its lane and comes to a stop at a red light [10].

Due to safety considerations, many autonomous driving architectures implement an additional system supervision layer which monitors the execution of the other system components. This supervisor checks

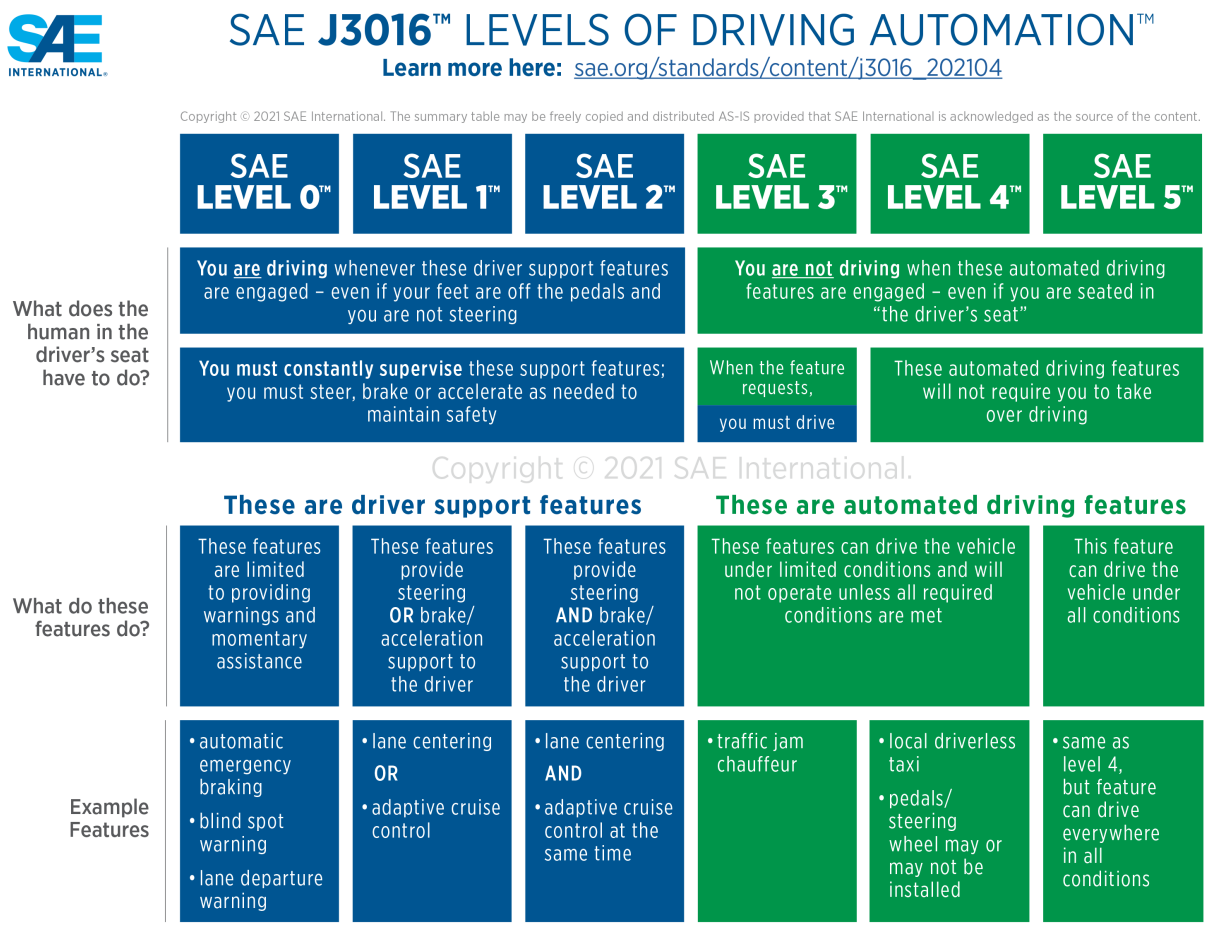

Figure 2.1: SAE Levels Of Driving Automation [1]

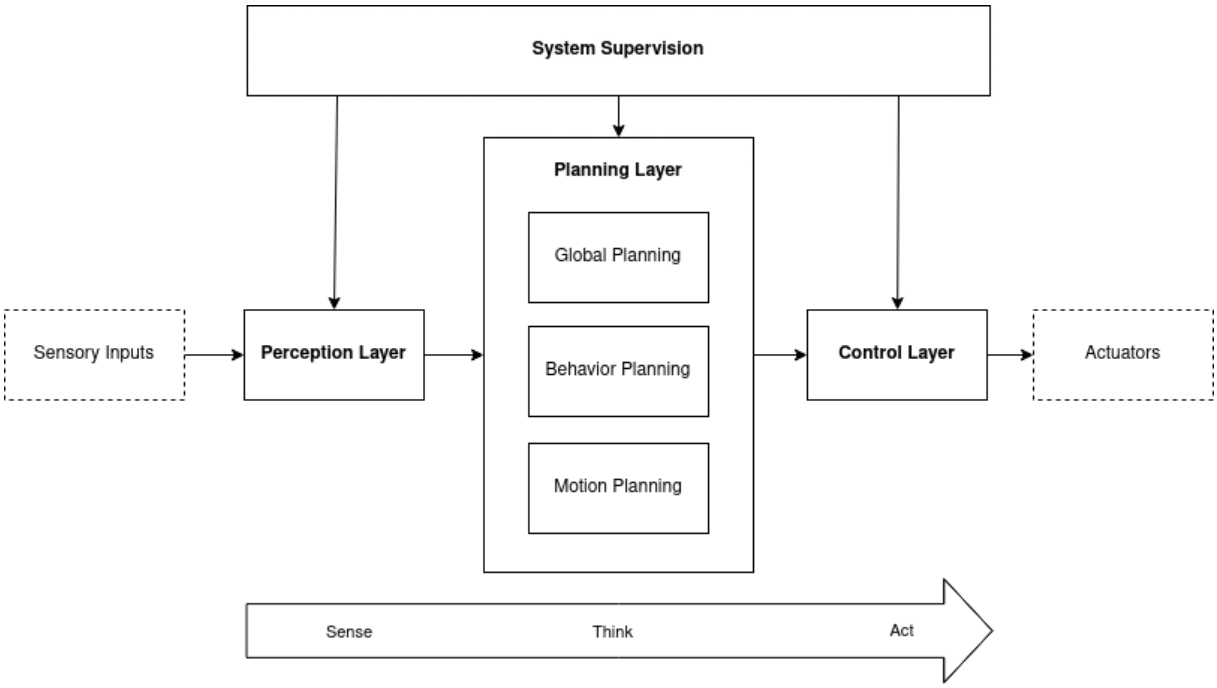

Figure 2.2: Common Autonomous Driving Architecture [2] [3]

the health of all software components. In case one or more components fail to pass the health check the supervisor is responsible for ensuring that the system does not continue the normal driving routine. The supervisor triggers measures to restore the normal functionality or if that is not possible bring the robot into a safe state (zimmermann2020).

Considering that with these architectures, vehicles can achieve an SAE level of autonomy up to level 4 (bacha 2008), a good starting point to increase autonomy in a robotic system would be to ensure that all of the functional modules in figure xx are implemented and available. On this basis higher levels of autonomy are achievable through the expansion of the behavior planning module inside of the planning layer.

bacha2008 reke 2020 wei2014 brooks1986 dhillon 2002 gonzales2016

2.3 Behavior Types

The behavior planning module contains a set of behaviors to operate in a environment. The task of the module is then to choose the best behavior to execute. In this sense a behavior is defined as a mapping of sensory inputs to a pattern of actions that in turn carry out a specific task [9]. A good behavior planning approach provides the robot with adequate behaviors for every possible scenario the robot is operating in. Different scenarios pose different challenges for the behavior planning module which has to decide on the best behavior option to execute in order to achieve a higher level goal.

2.3.1 Reactive

During the emergence of behavior based robots, the first type of behavior that was implemented in many robots was a simple reactive behavior. This behavior maps a sensory input directly to motor commands. In human behavior this type of behavior resembles reflexes, like the tapping on the kneecap which unwillingly results in motion in the knee joint. This behavior pattern is not following the typical "Sense, Think, Act" loop, but shortcuts directly from sensing to acting [11]. Reactive behaviors can be chained together to result in more advanced and goal-driven behaviors. Despite the possibility of more complex robot behaviors, sequential behaviors remain on the level of reactive behaviors due to the lack of a planning and decision-making cycle during their execution. The computational load of reactive behaviours is low and thereby fast as no decision and planning process is taking place, which makes them suited in scenarios where real-time safety is of concern. This property makes these kind of behaviours useful for system supervisor, where sometimes immediate reactions with low latency are required to ensure vehicle safety. But a system with a reactive behavior planning approach will always fail to meet the requirements for higher levels of autonomy due to the fact. This does not mean that reactive behaviors can not be part in highly autonomous systems as their quick response time to sensor inputs makes them valuable in improving vehicle safety and reliability.

2.3.2 Deliberative

Reactive behaviors suffer the drawback that they are not suitable enable complex behaviors. In addition, the creation of sequences of reactive behaviors that produce the target behavior is a complicated task [9]. Behaviors that involve a planning and decision making aspect are defined as deliberative. Unlike reactive behaviors, deliberative behaviors are following the Sense, Think, Act paradigm. They are not mapping sensory inputs directly into motor commands. Instead deliberative-type behaviors are deciding the best course of action before acting and only then proceed with the exution. The use of deliberative behaviors in the planning module is what enables a robot to meet the defintion of autonomy, e.g. making decisions and react to unforeseen circumstances. So in order to improve the autonomy of a robot, the focus should be on the creation of deliberative behaviors. A behaviour planner with a deliberative approach makes decision based on current sensor data, as well as previously processed data. By incorporating older sensor information into the decision making process, the deliberative behavior planner can make predictions about the changes that will happen in the environment. This allows the planner to proactively change the motion planning commands to better adapt to the environment. For example, a motion prediction of obstacles in highly dynamic environments would improve the motion planning considerably. This is due to a better understanding of how obstacles are interacting with the planned path of the robot in relation to the time at which the robot and obstacle are predicted to intersect paths. A purely reactive planner could never determine if an obstacle is destined to intersect the robots path in the future and would reroute constantly in order to accomplish the goal. A deliberative planner could determine if the

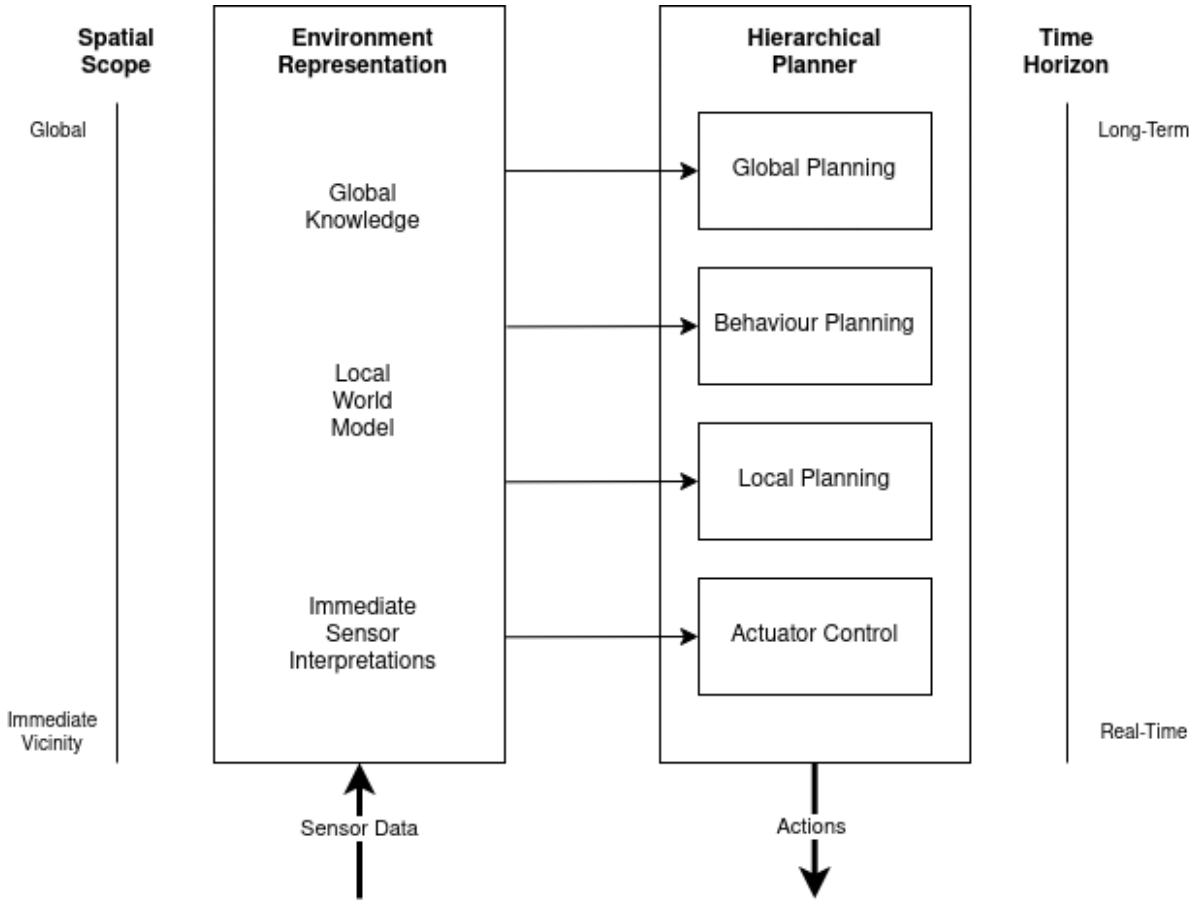

Figure 2.3: Hybrid Planning Architecture [4]

best course of action is to stay on the current path as the dynamic obstacle has already moved away by the time the robot reaches to intersection point. Other actions in scenario are possibly slowing down or speeding up briefly to avoid an obstacle and staying on the current path as this is calculated to be quicker than rerouting and taking a longer path. Generally, these cognitively more complex behaviours mimic human skills and thus make a system more autonomous.

Talk about machine learning as deliberative behavior -¿ true autonomy bc learning and unknown scenarios, programmers can not cover every scenario, so a well designed/trained ai is needed to make a system really autonomous, but -¿ missing determinism and unaccountability of machines, hence we need behaviors that are still hard coded and deterministic, these can be highly automated. traditional behaviour and ai generated behaviours can and must coexist in the system (adler2019)

silva2008 ingrand 2014 vasiolopolous 2022

2.3.3 Hybrid

To create reliable aswell as intelligent systems one can combine reactive and deliberative behaviours in a hybrid model. This behavior planning module is then able to quickly react to incoming sensor data and still exhibit high levels of automation.

Figure xx shows the areas where reactive and deliberative behaviours have their advantages and limitations. Deliberative planning does possess limitations during the execution of generated plans as the time horizon is long-term and not able to react to fast changes in the environment . When the system is not adressing these points, deliberative robots are very focussed on a small problem domain and are not robust [4]. The incorporation of reactive behaviors into the behavior planning combats this problem. In this multi-layer approach the higher level, more deliberative planners are able to override the reactive planners in order to allow the system to be more flexible and adaptible but still maintain fast reaction times. This architecture leads to more robust robots that are able to be deployed in a wider variety of

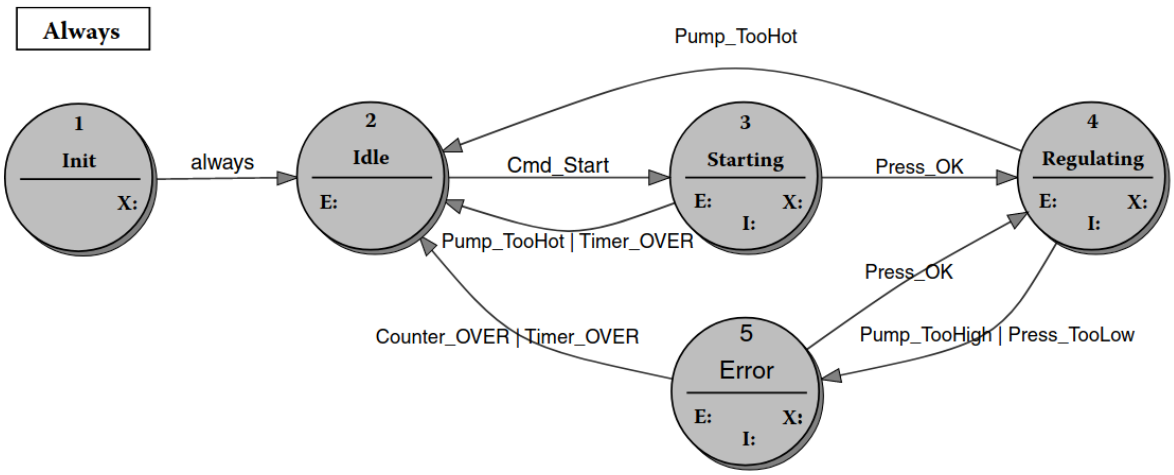

Figure 2.4: Example of a finite state machine transition diagram [5]

environments, thus leading to higher levels of autonomy.

2.4 Behavior Planning Approaches

Robots that use a hybrid, hierarchical behavior planning approach need a system which decides on a high level which behaviors are to be executed. This allows overriding simple reactive behaviors with more complex, deliberative behaviors when the system benefits from it.

The different solution for such behavior execution system all share the fact that they make use of states a robot can be in. Based upon a robots state and additional information about the environment, different strategies and behaviors are executed during the runtime of the system.

2.4.1 Finite State Machines

Finite State Machines (FSM) are commonly used in the environment of autonomous driving. They are used to describe behavior in a form of states, which in turn execute action. A finite state machine is defined by a list of possible states S, a starting state s0, a set of state transitions delta, a set of final states F and the input alphabet sigma. At any given time only a single state is selected and its containing actions are executed.

Figure xx depicts an example state machine with different states and examplatory transitions between them. Inside of the states the letters indicate the existence of actions. E stands for entry, I for Input and X for Exit. The arrows between the states illustrate the possible transition and transition conditions for every state. A state transition diagram is a good way to understand the system behavior but can not show the details of the modededeled behavior, like the actions [5].

Creating small state machines can easily and quickly be accomplished through the use of various feature-rich and performant libraries [12]. Finite state machines are often being used for comprehensive modeling of reactive behavior and sequential behaviors in autonomous driving functions [13]. These finite state machines make use of nested state machines inside of another state machine. This reduces the complexity of the state machine as the transition do not need to be modeled, because the start and final state of a nested state machine is treated as just one state inside of the larger state machine. Despite this possibility to simplify state machines, as a modeled behavior system grows larger to accomodate more complex behaviors into the state machine the effort to expand and maintain the state machine grows rapidly. This is due to the modeling effort of the transitions, transition conditions and transition events growing with each new state that gets introduced into the state machine as existing states need to be checked and updated accordingly [14].

The execution time of finite state machines can be fast enough that they are used to control the motors of a bipedal robot to enable the robot to balance and walk despite unexpected height variations

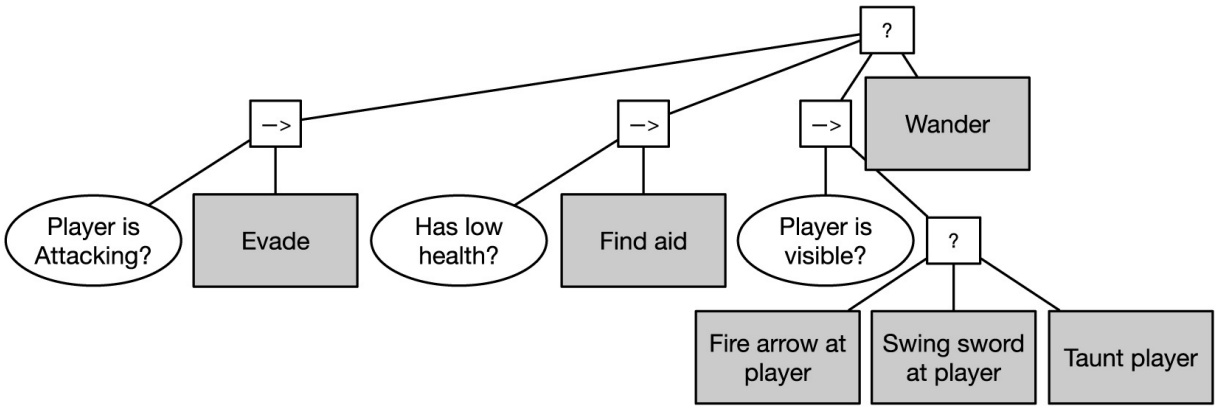

Figure 2.5: Example of a behavior tree [6]

between steps [15]. This property makes state machines suited very well suited for fast, reactive type behaviors and still allows a hierarchical approach to behavior planning.

allgeuer 2013 connor 2017 ziegler 2014

2.4.2 Behavior Trees

Behavior Trees (BT) are another way to model and control the behavior of autonomous systems. Behavior Trees first found big acceptance in the computer game industry where they are mainly used to model artificial intelligence for non-player characters [16]. Every tree consists of one root and many child, parent and leaf nodes. Leaf nodes are also called execution nodes, while non-leaf nodes are called control or control flow nodes. The execution of a behavior tree is done by ticking the root node of tree. This signal then travels down to the child node of the root node, where either control nodes are being ticked or action nodes are executed. Nodes return either "Success", "Running" or "Failure" which influences how the tick signal gets processed by the rest of the behavior tree. Control nodes can be of the type "Sequence", "Fallback" or "Condition". The condition node can not have child nodes and can only return success or failure. A sequence node can be compared to a logical "and" condition. This means that once a sequence gets ticked from the parent node, it will send the tick signal to every child node as long as they return "success" of "running", and only return "success" itself when every child node was successfully ticked. If any of the child nodes return "failure", the sequence node will stop ticking the remaining children and return "failure". On the other hand the fallback node is an equivalent of a logical "or" condition. The fallback node will tick its child nodes as long as they return "failure" or "running" and will return "success" if one of the ticked node returned "success". It will only return "failure" if all of the child nodes "returned success". Figure xx depicts an example of a BT with multiple levels and action nodes. The nodes with the arrow (-¿) symbol are sequence nodes, the question mark (?) symbol denotes fallback nodes. The condition node are depicted as white ovals. The action nodes are represented as gray rectangles in the BT example.

One of the core differences to Finite State Machine is that transitions between states are not distributed across all the states but they are organized in a hierarchical tree, where the leaves are representing the states [6]. While both FSM and BT are capable of producing the same behavior in robots, the fundamental shift in how the two system are created lead to significant advantages in the modularity, synthesis and analysis of the systems at hand. These effects are more significant with an increase in size of the system. BT offer a lot more flexibility when creating an advanced behavior layer for an autonomous system.

iovino2022 marzinotto 2014 colledanchise 2018

Chapter 3

Concept

3.1 Current State

This chapter briefly presents the current state of mobile navigation with ROS and its limitations. The goal of this chapter is to then identify the key areas in which standard mobile robots running ROS lack robustness and autonomy during autonomous mobile navigation. According to these key areas requirements are derived from them and prioritized with a risk analysis. Furthermore this chapter proposes a way how behavior planning can improve the current systems to meet the requirements.

3.1.1 Turtlebot3

A standard mobile research plattform that has an available ROS interface to control the robot is the Turtlebot3. The robot is well integrated in the ROS ecosystem and is established in the literature as a system to develop and integrate new methods on. The robot has two motors with attached wheel encoders to drive and steer, so it classifies as an differential drive robot. A third omnidirectional wheel is added for stability.

The manufacturer provides an open-source model for simulating the robot in physics-based simulators like Gazebo, as depicted in figure xx, which enables faster development and testing for the robot. The equipped sensors and easy simulation capabilities make the robot very well suited for the use of the Nav2 stack and further development for behavior planning.

The robots specifications are specified in table xx.

The turtlebot will be used for the analysis of the current state of mobile navigation. Furthermore, the simulated turtlebot will be the plattform on which the behavior planning will be implemented and tested.

3.1.2 ROS?!

write sth about that?

3.1.3 Navigation2

The ROS Navigation2 Stack (Nav2) is a combination of different packages that allows mobile robots to navigate from point A to point B. Navigation2 is the de-facto standard to achieve mobile navigation with a wide-range of supported robots. Supported robot types are holonomic, differential-drive, legged and ackerman (car-like). For the mobile robot to make use of the Nav2 stack, it has to be set up in a certain way to be able to generate plans and execute commands in the right way. Nav2 requires the mobile robot to possess a laserscan or pointcloud sensor, an odometry source (such as an IMU, wheel encoders) and a set of transformations to be available for planning and navigation. The stack contains tools that save and load occupancy maps, localize the robot, plan paths, execute the path, provide costmaps, build behaviors and execute recoveries [7].

These packages are often combined with a SLAM method to build or enlarge maps. The path planning and path execution capabilities, which correspond to the global and local planner as described in section 2.2, are further supported by ready-to-use planning plugins which use A* and Dijkstra algorithms for path planning and a dynamic window approach for path execution. The system sequencing is done

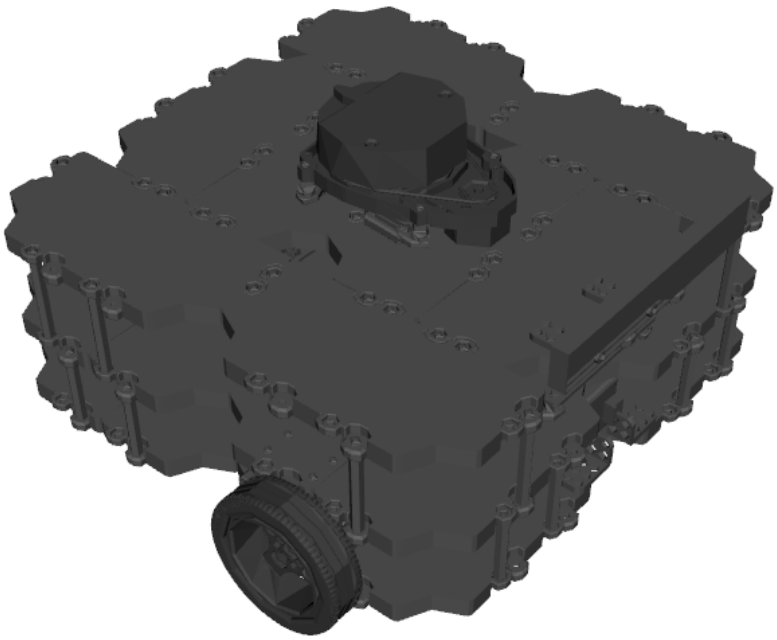

Figure 3.1: Turtlebot3 model in Gazebo simulator

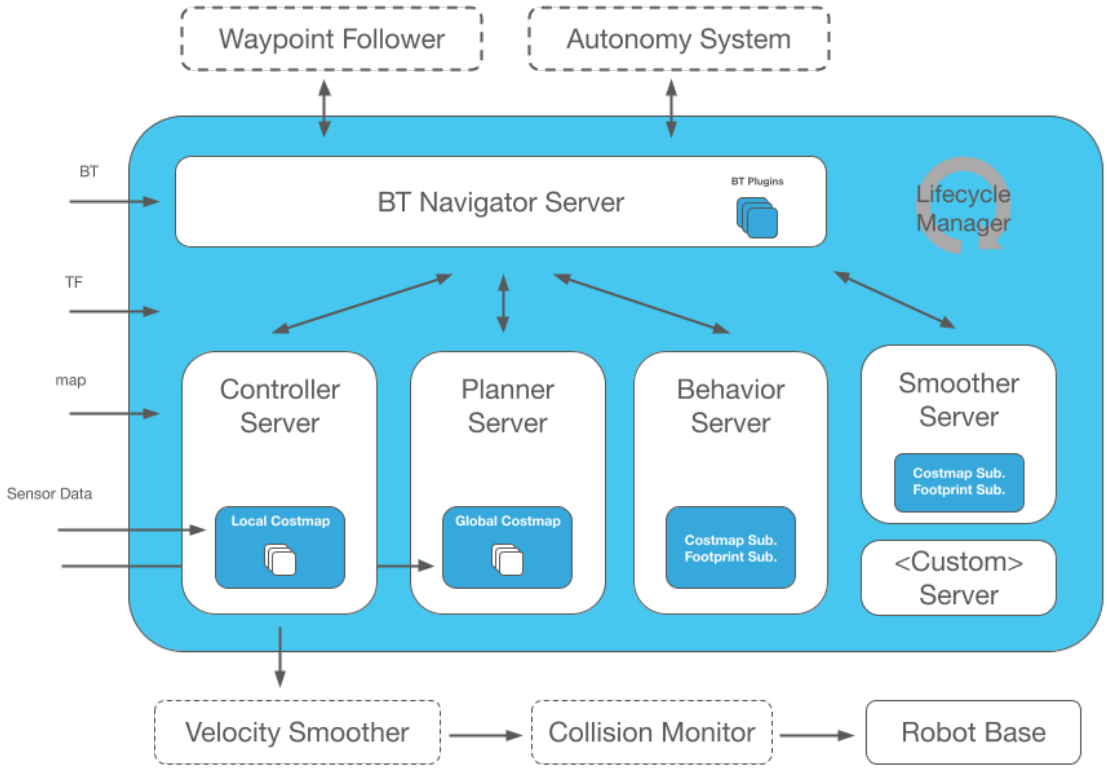

Figure 3.2: Navigation2 Architecture [7]

in a hierarchical structure as there is central behavior tree which calls asynchronous actions from the respective planners after another as seen in figure xx.

The behavior tree depicted in figure xx is the one that is the default Nav2 behvaior and has a set of recovery behaviors already implemented (spin, wait, backup, clear map). These behaviors are executed

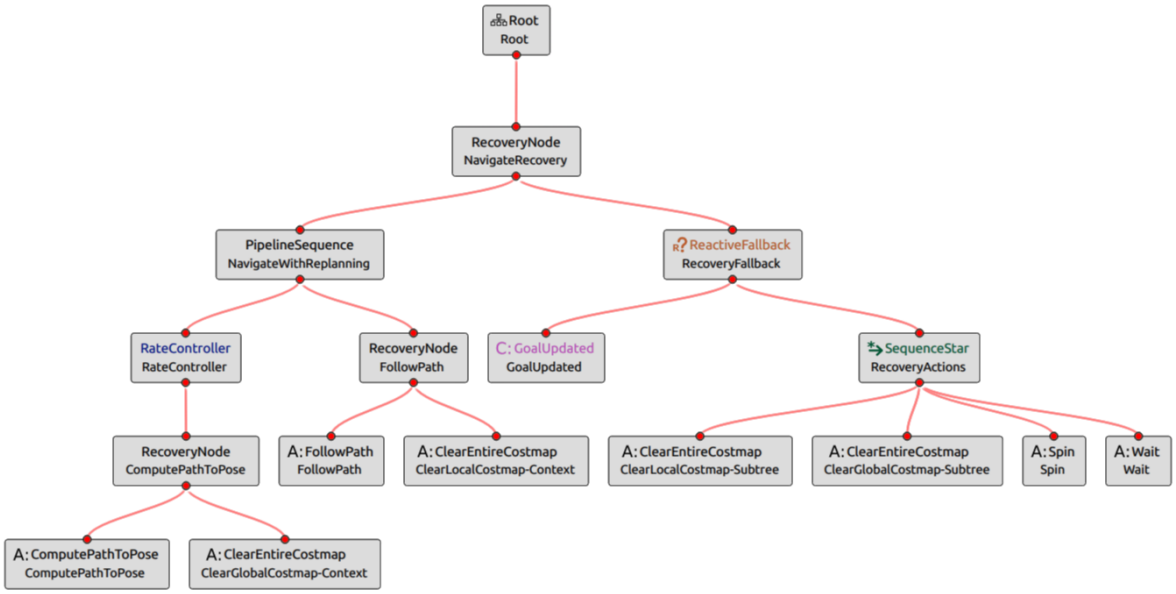

Figure 3.3: Navigation2 Behavior Tree

if the robot can not make progess towards the set goal. Nav2 makes use of a node lifecycle management system which is used to start, activate, deactivate and shutdown nodes in a controlled way. This allows the system degree to monitor the system's nodes startups, executions and failures.

With this quantity of functions and hierarchical architecture the ROS Navigation2 stack is a great basis to achieve high levels of autonomy in robots. Due to the expandability of the software with plugins the software can be modified and expanded to fit the users needs. When comparing the functionalities of the software stack to the proposed autonomous system architecture in figure xx (2.2), the things that Nav2 are missing are a system-wide supervision layer and and a more deliberative approach to behavior planning as mentioned in 2.3. The absence of these components leads to a limited set of circumstances in which the robot can perform navigation reliably. One of the core problems that behavior-based robots face is to generate a dependable environment representation as this is the basis for all of the latter steps in the planning part of the "Sense - Think - Act" cycle. Unforeseen events can severely limit the reliability of the environment representation without the robot being aware of it. Examples for such types of events that are currently not handled appropriatly are slipping wheels, orientation changes by external influences or undetected, more specifically undetectable obstacles. This can lead for example to erratic movement of the robot, as reactive behaviors and more deliberative behaviors compete for authority over the robots movement. By design, in order to allow more intelligent robot behaviors, the deliberative behavior can override commands from reactive behaviors. But the deliberative behavior's planning is based upon a wrong representation of reality which then leads to unsafe commands which in turn triggers reactive-type behaviors again to counteract the commands from the deliberative behavior. Possibly the reactive behaviors do not get activated at all and the robot is continuing to drive despite being unaware of the surroundings. This would be a situation in which a human operator would need to step in and restore the robot to full functionality by hand.

Add a closing statement about that suitability of the Nav stack for achieving autonomy

3.2 Risk Analysis For Requirement Prioritization

3.3 Requirements

How do I justify these requirements? Atm they are made arbitrarily with no real substance or reasoning.

Functional Requirement List

number .. Name .. Description .. Success Criteria -.. Priority freq1 Recovery The system can recover from crashes Ensure that the system can successfully reach goals despite previous crashes freq2 Sensor

Failure The system detects sensor failures Ensure that the system can restart sensors and decrease the speed during time sensor delivers limited information freq3 Maintain operability The robot executes commands as long as it is safe Ensure that the robot keeps driving if it is safe even when system functions are not working correctly freq5 Emergency Detection The system detects emergency. Ensure that the system can detect when the continuation on the calculated path is no longer safe (sensor failures, blockage) freq6 Emergency Stop The system can initiate emergeny stops Ensure that the system can override all commands and stop in case an emergency is detected freq7 Override Navigation2 The system can override navigation2 Ensure that the system's commands can always override the commands coming from navigation2. freq8 Reset Goals The system can reset and override goals set by the user so that the goal is reachable by planners. freq9 Determinism The outcome for a given set of inputs has to be deterministic, meaning that the behavior is always executed in the same way.

Table 3.1: Functional Requirements

Nr.	Name	Priority	Description
freq1	Recovery	High	The system can recover from crashes. Ensure that the system can successfully reach goals despite previous crashes
freq2	Sensor Failure	High	The system detects sensor failure.Ensure that the system can restart sensors and decrease the speed during time sensor delivers limited information
freq3	Maintain operability	High	The robot executes commands as long as it is safe Ensure that the robot keeps driving if it is safe even when system functions are not working correctly
freq4	Emergency Detection	High	The system detects emergency. Ensure that the system can detect when the continuation on the calculated path is no longer safe (sensor failures, blockage)
freq5	Emergency Stop	High	The system can initiate emergeny stops. Ensure that the system can override all commands and stop in case an emergency is detected
freq6	Override Navigation2	High	The system can override navigation2. Ensure that the system's commands can always override the commands coming from navigation2.
freq7	Reset Goals	Medium	The system can reset and override goals set by the user so that the goal is reachable by planners.
freq8	Determinism	High	The outcome for a given set of inputs has to be deterministic, meaning that the behavior is always executed in the same way.

Non-functional requirement list

number .. Name .. Description .. Success Criteria nfreq1 Single Point of Failure The system does not have a single point of failure When parts of the system fail the system maintains operability to a degree nfreq2 Performance The system's control loop guarantees fast reactions The average frequence with which the system checks the sensors/navigation2 is higher than 100Hz (10ms) nfreq3

3.4 Software

3.4.1 Behaviortree.CPP

The Navigation2 stack uses a behavior tree to coordinate the planning layer. The behavior tree that is used is an expansion of the Behaviortree.CPP (BT.CPP) library. This library offers a fast way to create, execute, monitor and edit behavior trees. Additionally to the types of control nodes that are mentioned in chapter 2.4.2 the library adds the concept of reactivity into the catalogue of control nodes. Reactive sequences and reactive fallbacks differ from normal ones in the handling of nodes that return the state "running". Instead of ticking the node again, the whole sequence restarts, which is very useful for continously checking a condition and executing an asynchronous action node, that gets halted when a condition is not met anymore.

Also, it offers Decorator Nodes which allow more control over the child node and its output. These Decorator Nodes can only have one child node each. A comprehensive list of the available nodes and descriptions is in table xx.

TABLE HERE

As a way to allow the tree nodes to communicate with each other, the library provides the developer with two possibilities. Either node can make use of the blackboard which is a dictionary (key/value) that all nodes of the tree can read and write. Or two nodes can be connected through ports which allows the direct communication between two nodes via a key/value.

Chapter 4

Implementation

4.1 System Architecture

4.2 Behavior Tree Structure

4.2.1 BT System Supervision

4.2.2 BT Advanced Behaviors

Collision Behavior

Battery Behavior

Path Planning Behavior

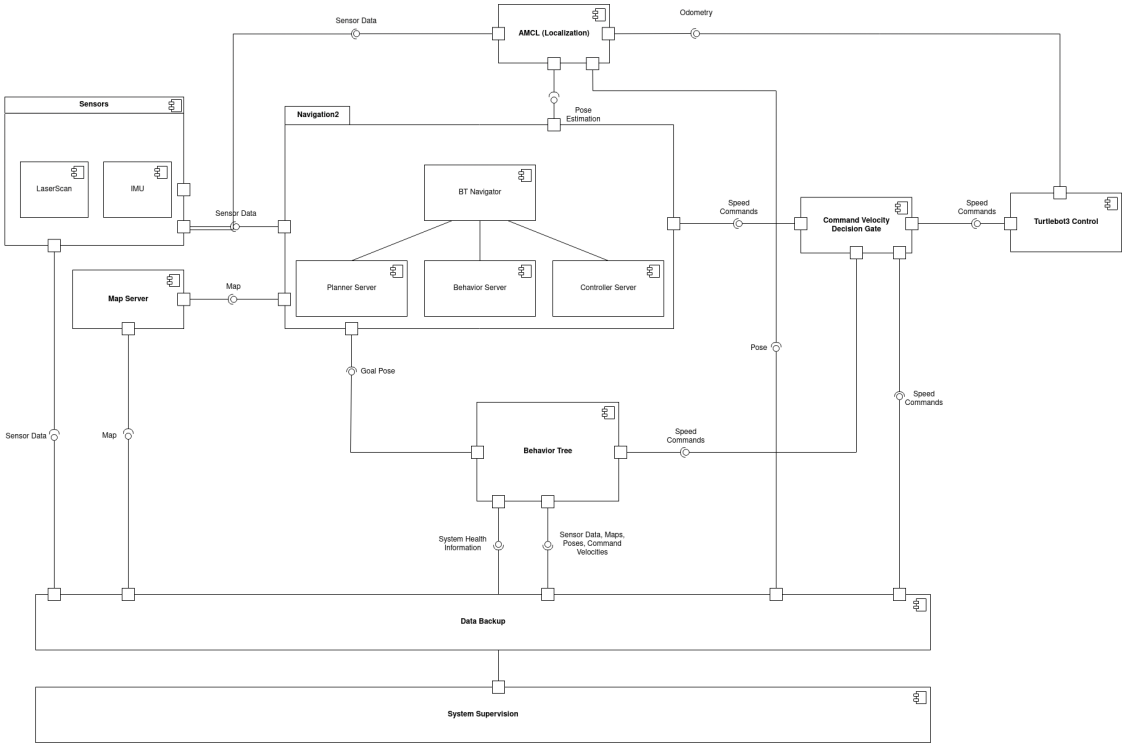

Figure 4.1: Turtlebot3 model in Gazebo simulator

Chapter 5

Evaluation

5.1 Scenarios

Chapter 6

Summary

6.1 Conclusion

6.2 Prospect

Bibliography

[1] SAEInternational, *Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles*, apr 2021.

[2] R. Brooks, "A robust layered control system for a mobile robot," *IEEE Journal on Robotics and Automation*, vol. 2, no. 1, pp. 14–23, 1986.

[3] G. Velasco-Hernandez, D. J. Yeong, J. Barry, and J. Walsh, "Autonomous driving architectures, perception and data fusion: A review," in *2020 IEEE 16th International Conference on Intelligent Computer Communication and Processing (ICCP)*, pp. 315–321, 2020.

[4] R. Arkin, R. Arkin, and R. Arkin, *Behavior-based Robotics*. Bradford book, MIT Press, 1998.

[5] F. Wagner, R. Schmuki, T. Wagner, and P. Wolstenholme, *Modeling Software with Finite State Machines: A Practical Approach*. 05 2006.

[6] M. Iovino, E. Scukins, J. Styrud, P. Ögren, and C. Smith, "A survey of behavior trees in robotics and ai," *Robotics and Autonomous Systems*, vol. 154, p. 104096, 2022.

[7] S. Macenski, F. Martin, R. White, and J. Ginés Clavero, "The marathon 2: A navigation system," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.

[8] E. F. und Innovation (EFI), "Gutachten zu forschung, innovation und technologischer leistungsfaehigkeit deutschlands 2018," tech. rep., EFI, Berlin, 2018.

[9] R. Murphy, R. Murphy, and R. Arkin, *Introduction to AI Robotics*. A Bradford book, MIT Press, 2000.

[10] M. Reke, D. Peter, J. Schulte-Tigges, S. Schiffer, A. Ferrein, T. Walter, and D. Matheis, "A self-driving car architecture in ros2," in *2020 International SAUPEC/RobMech/PRASA Conference*, pp. 1–6, IEEE, 2020.

[11] L. De Silva and H. Ekanayake, "Behavior-based robotics and the reactive paradigm a survey," in *2008 11th International Conference on Computer and Information Technology*, pp. 36–43, 2008.

[12] M. Foukarakis, A. Leonidis, M. Antona, and C. Stephanidis, "Combining finite state machine and decision-making tools for adaptable robot behavior," in *Universal Access in Human-Computer Interaction. Aging and Assistive Environments* (C. Stephanidis and M. Antona, eds.), (Cham), pp. 625–635, Springer International Publishing, 2014.

[13] J. Ziegler, P. Bender, M. Schreiber, H. Lategahn, T. Strauss, C. Stiller, T. Dang, U. Franke, N. Appenrodt, C. G. Keller, E. Kaus, R. G. Herrtwich, C. Rabe, D. Pfeiffer, F. Lindner, F. Stein, F. Erbs, M. Enzweiler, C. Knöppel, J. Hipp, M. Haueis, M. Trepte, C. Brenk, A. Tamke, M. Ghanaat, M. Braun, A. Joos, H. Fritz, H. Mock, M. Hein, and E. Zeeb, "Making bertha drive—an autonomous journey on a historic route," *IEEE Intelligent Transportation Systems Magazine*, vol. 6, no. 2, pp. 8–20, 2014.

[14] D. C. Conner and J. Willis, "Flexible navigation: Finite state machine-based integrated navigation and control for ros enabled robots," in *SoutheastCon 2017*, pp. 1–8, 2017.

[15] H.-W. Park, A. Ramezani, and J. W. Grizzle, "A finite-state machine for accommodating unexpected large ground-height variations in bipedal robot walking," *IEEE Transactions on Robotics*, vol. 29, no. 2, pp. 331–345, 2013.

[16] G. Florez-Puga, M. A. Gomez-Martin, P. P. Gomez-Martin, B. Diaz-Agudo, and P. A. Gonzalez-Calero, "Query-enabled behavior trees," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 1, no. 4, pp. 298–308, 2009.