2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 14-18, 2024. Abu Dhabi, UAE

Streamlining Object Pushing: Behavior Tree-Based Coordination of Control and Planning

Filippo Bertoncelli[1] and Lorenzo Sabattini[2]

Abstract— **Efficiently navigating and manipulating objects in complex environments is a fundamental challenge in robotics. This paper presents a novel approach to streamline object-pushing tasks by integrating Behavior Trees (BT) to coordinate a control and planning framework. The proposed system optimizes the execution of tasks involving the pushing of objects while ensuring adaptability to varying scenarios.**

Our approach employs BTs to encapsulate high-level task specifications and decision-making processes, facilitating a flexible and intuitive representation of robot behavior. By seamlessly integrating BT technology with a coordinated control and planning system, we enable the robot to make real-time decisions and adapt to dynamic environments.

We present experimental results demonstrating the effectiveness of our approach, highlighting its ability to improve task execution efficiency and adaptability.

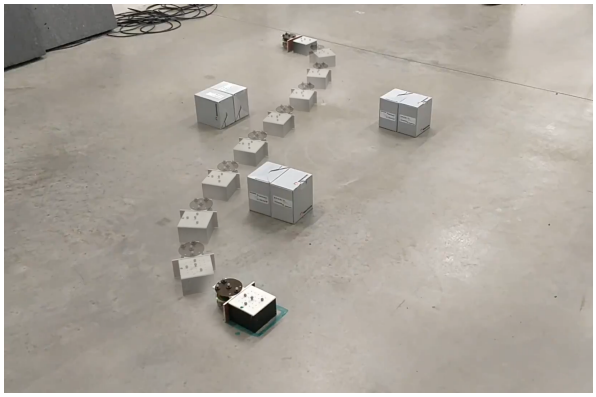

Fig. 1: Mobile robot pushing a polygonal object in an environment with obstacles.

I. INTRODUCTION

Robots have become indispensable in modern life, particularly in industrial settings where they play a significant role in performing tasks characterized by repetitive precision or involving heavy payloads. An exemplary case of this is observed in industrial robots within factories, meticulously designed to execute specific tasks with efficiency, ultimately making human labor more manageable. A common thread across various applications is the necessity for robots to engage with their surrounding environment. This interaction can assume diverse forms, ranging from the sophisticated sensing of environmental attributes to the application of manipulative actions.

In this context, we explore a specific category of interactions that robots can perform in their surroundings, referred to as *pushing*. Visualized in Figure 1, pushing is a manipulation technique that falls under the broader category of non-prehensile manipulation [1]. Conventionally, robots have often modified their environments using prehensile manipulation techniques, where the object under manipulation is tightly bound to the robot's motion [2]. This connection is established through various means, such as mechanical grippers, robotic hands, or suction end effectors. However, this mechanical linkage between the robot and the object can frequently be the weakest link in the kinematic chain. Robotic hands and grippers are susceptible to malfunction and costly to repair. Non-prehensile manipulation techniques present a potential solution to mitigate these issues. The defining characteristic of this class of manipulations is the absence of a fixed element that constrains the object to move in sync with the robot's motion. Notably, friction plays a pivotal role in executing this type of manipulation. Challenges may arise due to the inherent uncertainty of frictional parameters, making it difficult to predict the object's motion. Therefore, pushing manipulation is particularly susceptible to external disturbances and errors in the motion model.

Pushing has been deeply explored in the literature, starting from the pioneering work in [3], where it was introduced as a precursor to grasping. Most of the literature has then been devoted to identifying the assumptions and constraints that guarantee the pushing feasibility, such as upper-bounds on the rotation rate [4], [5], or assumptions on the friction characteristicsn [6]–[8]. Such concepts have then been specialized for pushing applied to mobile robots, in [9], [10] and references therein.

Several research studies have focused on planning approaches to achieve pushing, exploiting accurate modeling of the pushing dynamics. In [11], a planner using linear pushes for repositioning objects in obstacle-free environments was introduced, providing complete solutions. A motion-constrained model was developed in [12] to identify *stable pushes*, and used with the A* algorithm for planning in obstacle-free scenarios, ensuring completeness. A two-layered planning approach was proposed in [13] for object-pushing through environments with obstacles. The first layer involved mapping the environment with a cost function, and the second layer selected suitable pushes towards the nearest goal. In [14], a two-layered method for 3D manipulation planning was presented, utilizing a global RRT planner for

[1]F. Bertoncelli is with the Technology Innovation Institute (TII), Abu Dhabi, UAE. filippo.bertoncelli@tii.ae

[2]L. Sabattini is with the Department of Sciences and Methods of Engineering, University of Modena and Reggio Emilia, Italy. lorenzo.sabattini@unimore.it.

Authors would like to thank Federico Rivi for his support in the realization of the experiments.

object configuration exploration and a local planner for push sequence determination. A data-driven approach was adopted by [15], where RRT-based planning was used with learned motion modes to guide tree expansion for finding feasible paths to the goal. Moreover, [16] identified the flatness associated with pushing and proposed a planning strategy for obstacle-free environments, along with an RRT-based strategy for environments with obstacles.

The pushing problem requires peculiar handling of the behavior of the system to fulfill a manipulation task, thus requiring the introduction of an ad-hoc coordination algorithm. Many technologies can be employed to achieve this objective. Among the most commonly used strategies such as Finite-State-Machines and Decision Trees, the Behavior Tree(BT) [17], [18] architecture offers some significant advantages such as flexibility, modularity, and the ability to react in real-time to changes in the environment. The utility of Behaviour Trees has been proven many times in different scenarios. In [19] BTs are used to model the behavior of objects in Virtual Reality scenes used for fire drill training. The authors of [20] used BTs to model conditional switching behavior for 6-DOF robots. The instructed behavior reacted to the movement of obstacles in the environment and adapted the behavior of the robot to guarantee safety in collaborative environments. [21] extends BTs introducing several priority scheduling strategies to handle the concurrent execution of tasks with different priorities. [22] used BTs to determine the behavior of UAVs in a swarm tasked to collectively perform an autonomous Reconnaissance task. In this paper, we introduce a top-level decision algorithm designed to modify the behavior of the system in real-time, allowing it to react effectively to unexpected events during the manipulation process. This capability enhances the robustness and adaptability of the pushing manipulation, making it more reliable and versatile in dynamic environments.

A. Contribution

The contribution of this paper is in a novel decision algorithm that, building upon the concept of Behavior Trees, allows to effectively perform pushing maneuvers with a mobile robot in an environment with uncertainties. In particular:

1) We propose conditions to:
   a) evaluate the proper contact between the robot and object.
   b) evaluate if a planned trajectory can be tracked by the controlled system in its current configuration.
2) We design a top-level decision algorithm that:
   a) reacts to the loss of proper contact and restores it
   b) reacts to unforeseen events that invalidate the current manipulation plan and trigger online replanning.
3) We tested the developed architecture in a real environment, with a mobile robot performing pushing also in the presence of external malicious actors.

II. Problem Description

Consider a flat, two-dimensional environment containing obstacles. In this environment, the objective is to manipulate a polygonal object by applying a pushing action on one of its sides using a robotic manipulator. We establish two coordinate frames for reference: Σ_w represents the global reference frame, and Σ_o is a local frame affixed to the manipulated object's center of mass. At any given time t, let $\chi_o(t) = \begin{bmatrix} x_o(t) & y_o(t) & \theta_o(t) \end{bmatrix}^T \in \mathbb{R}^3$ denote the object's pose expressed with respect to Σ_w, where $x_o(t)$ and $y_o(t)$ represent the object's position coordinates, and $\theta_o(t)$ denotes the planar orientation of Σ_o with respect to Σ_w. Similarly, the robot's pose is described by $\chi_r(t) = \begin{bmatrix} x_r(t) & y_r(t) & \theta_r(t) \end{bmatrix}^T \in \mathbb{R}^3$. The obstacles in the environment are collected into the set $\mathcal{O}$. Assume the robot applies pushing actions on the sides of the object using a flat surface that acts as an end effector for the robot. A possible robotic pushing manipulation system is shown in Figure 2 where a mobile robot with a front bumper is used to push a square box. Combining these pushing actions it is generally possible to transport the object from one configuration to another reachable configuration in the same environment. Both the problems of pushing with a robot along a trajectory, and planning said trajectory, have been addressed by several works in literature, such as [16], [23]. These works however assume the environment has been modeled correctly and that there are no external actions that might interfere with the success or failure of the manipulation. An erroneous estimation of the friction coefficients or the presence of a malignous actor in the environment are examples of unforeseeable events that can disrupt the execution of a manipulation.

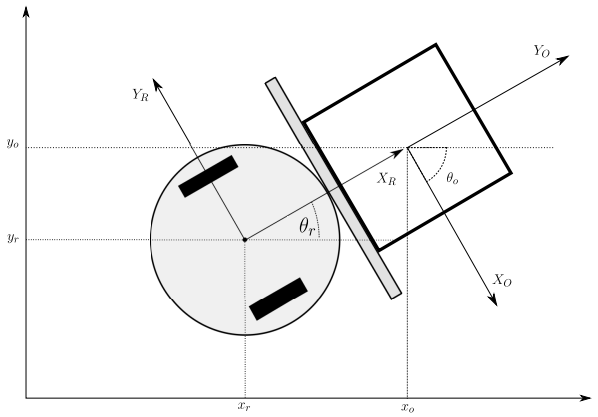

Fig. 2: Schematic representation of the differential drive mobile robot while pushing an object. The black rectangles are the wheels. The frontal bumper is represented by the white rectangle in front of the robot

In summary, in this work, we address the following problem.

Problem 1: Define the behavior of a robotic system for pushing an object towards a target, addressing potential parameter error and malicious external actions. ◇

To address this problem we propose a decision algorithm based on the Behavior Tree architecture. This architecture allows the definition of conditional execution of different tasks in a reactive and modular way. The proposed framework is

capable of fulfilling the required manipulation task under the following assumptions.

- A1 The friction conforms to Coulomb's law. The friction reaction force of a sticking contact can act in any direction tangential to the contact normal, with maximum magnitude μf_n, where $f_n \in \mathbb{R}^+$ is the magnitude of the normal contact force and $\mu \in \mathbb{R}^+$ is the friction coefficient associated with the contact. For a sliding contact, the frictional force resists the motion with a force of magnitude μf_n.
- A2 The environment is planar. All the pushing forces lie on the plane while gravity acts downward along the vertical axis.
- A3 The contact friction coefficient is uniform across the plane.
- A4 The motion is slow enough that all inertial forces are negligible or instantaneously absorbed by the frictional forces. The frictional support forces always balance the pushing forces. This is the quasi-static assumption.
- A5 The position of the obstacle in the environment is known and does not change over time.
- A6 The pushing action is performed by the robot on the object using a flat surface that acts as the end effector.
- A7 The object is not deformed nor damageable by the robot during the pushing action.
- A8 The current position of the robot and object in the environment is known with sufficient precision from an external source.

III. Preliminaries

A. Behavior Trees

Behavior Trees are a popular method of structuring the switching between different tasks in autonomous agents, such as robots or virtual entities in computer games. BTs are highly effective in creating complex systems that are both modular and reactive, which is essential for many applications. As a result, BTs have been adopted from computer game programming to many areas of artificial intelligence and robotics. At the core, BTs are directed rooted trees where the internal nodes are called control flow nodes and the leaves are called execution nodes. A BT starts its execution from the root node that generates signals that allow the execution of a node called ticks with a given frequency, which is sent to its children. A node is executed if and only if it receives ticks. The child immediately returns *Running* to the parent, if its execution is underway, *Success* if it has achieved its goal, or *Failure* otherwise. In the classical formulation, there are 4 types of control flow nodes (Sequence, Fallback, Parallel, and Decorator) and 2 types of execution nodes (Action and Condition). The Sequence node ticks its children in order. The node reports success only when its last child succeeds, while, if any of them fails, it reports failure. If any of its children return the *Running* status, the node reports running as well. The Fallback node is used to select between hierarchically ordered behaviors. The node reports *Running* if any of its children report running, while if one of them succeeds it reports success, and failure if all of its children fail. The Parallel node executes all of its children concurrently and reports success if either any or all of its children report success. The Decorator node is used to modify the return status of its child node (*Success* becomes *Running*, *Failure* becomes *Running*, *Success* becomes *Failure*, etc.). The execution nodes are the nodes that carry the actual work in the tree. The Condition nodes, through their reported status, determine if a condition is verified or not, therefore they never report *Running*. The Action nodes are used to define temporary goals for the system, achievable through specific actions. In the proposed solution, the behavior tree interfaces with the other components in the framework through 3 different actions: 1) Compute a plan, 2) Move the robot to a target pose, and 3) Push the object along a trajectory. For each of these actions, the corresponding node will report *Running* until the action has been completed.

B. Pushing

The principal connotation of pushing manipulation is the limited set of generated motions due to the reduced contact interaction and lack of force closure. For example, an object that can be pushed in one direction cannot be moved in the opposite one by simply reversing the motion of the pusher.

Analyzing the contact with a polygonal object it is possible to determine motion directions where the contact is maintained using friction alone [6], [12], [24] to the point that the motions reduce into the class of limited curvature trajectories, corresponding to Dubins curves [16], [25], [26]. To bring an object towards a target only by pushing, it is then necessary to have a manipulation plan that is feasible with respect to the constraints. The plan is generated starting from a set of pushing actions that the robotic agent can apply to the object. A planning algorithm then constructs a series of pushing actions to apply to the object sequentially. Semantically, each action is composed of a contact configuration between the robot and the object and a trajectory to follow that satisfies the pushing motion constraints deriving from the contact configuration. The second necessary component to complete the manipulation is a controller that instructs the motion of the robot so that the object-robot system executes the pushing actions in the manipulation plan.

The proposed solution will orchestrate these two essential components as well as decide the behavior of the robot when not in contact with the object. The proposed algorithm is designed to be independent of the object and robot performing the manipulation: therefore, at a high level, the only information it needs to know is the possible pushing actions to be applied, the pose of the robot, object, and target. More specifically, the proposed solution becomes part of a larger framework for planar pushing manipulation with a mobile robot [23]. Briefly, the employed planning algorithm makes use of a two-layer planning strategy designed to compute collision-free and curvature-constrained trajectories to guide a mobile robot in pushing an object to a desired pose. The algorithm outputs a list of pushing actions (trajectories with a specific associated side of the object to push). The first layer

utilizes the Rapidly-exploring Random Tree Star (RRT*) algorithm to find the shortest path around obstacles to the goal, avoiding collisions by considering both the path length and the clearance from obstacles. This path serves as a guide for the second layer, which employs an A* search algorithm to determine a sequence of manipulation trajectories. The A* algorithm uses the waypoints from the RRT* path to create a graph of potential trajectories, selecting those that meet the curvature constraints and avoid collisions. This search process results in a manipulation plan comprising a sequence of pushing actions, each associated with a specific side of the object, guiding the robot to move the object into the desired pose efficiently.

Initial experiments with an open-loop control strategy revealed that small imperfections, such as uneven floor surfaces or varying friction, could lead to significant errors over time. To address these challenges, a feedback loop was introduced through a Linear Time-Varying Model Predictive Control (LTVMPC) algorithm, focusing on precisely regulating the object's movement based on the planned trajectory.

The LTVMPC algorithm works by predicting and adjusting the robot's movements to ensure the object follows a desired path with high accuracy. This control strategy adapts to time-varying dynamics and constraints of the motion model, accounting for both linear and angular velocities of the robot. The algorithm's predictive nature allows for anticipating future states and making adjustments to minimize errors and ensure compliance with physical constraints, like avoiding collisions and managing the robot's interaction with the object to prevent slippage.

To achieve this, the control architecture employs a model that captures the robot's interaction with the object, considering the robot's linear and angular velocities and their impact on the object's trajectory. The model is then linearized and discretized to form the basis of the LTVMPC optimization problem, which aims to minimize a cost function over a finite prediction horizon while adhering to motion constraints and avoiding collisions.

The MPC formulation includes constraints to ensure that the robot's movements do not cause the object to slip and to maintain a safe distance from obstacles. This is achieved by defining convex constraints for both the robot's motion relative to the object and the avoidance of fixed obstacles in the environment. The optimization problem is solved iteratively to find the optimal control inputs that guide the object along the desired trajectory while considering the dynamic constraints and uncertainties of the real-world environment.

More details about the planning and control framework can be found in [23].

The proposed Behavior Trees-based coordination algorithm is introduced to improve the reliability of the framework and provide the ability to react to unforeseen events.

IV. Behavior Tree Approach

To achieve a solution to the problem described in Section II it is necessary to distill some key requirements for the behavior of the system.

A. Requirement: Plan Exists and is Feasible

The first requirement we identify is the need to ensure that a manipulation plan has been computed and that the robot is capable of executing it. The existence of a plan is quite straightforward to determine, however, the ability of the system to execute the plan is non-trivial. In practical terms, if we assume proper contact between the robot and the object, the first condition translates into determining if the system can converge into the planned trajectory from the current position. The proposed solution to compute this conditional check is to search for a trajectory in the form of a Dubin's curve that starts in the current object pose and ends in one of the poses of the current reference trajectory. This search is performed with an iterative approach until a collision-free trajectory is found, and, if no such trajectory is feasible, the reference trajectory is determined to be unfeasible, and therefore a new manipulation plan has to be computed. This algorithm is detailed as pseudo-code in Alg. 1. An example of a collision-free trajectory and a colliding trajectory is presented in Fig. 3. The conditional check described in

Algorithm 1 Reference is Feasible

Require: $\mathcal{R}, s$ ▷ Reference trajectory and associated pushing side
Require: $\mathcal{O}$ ▷ Set of obstacles in the environment
Require: $\chi_o(t)$ ▷ Current object pose
function IsReferenceFeasible($\mathcal{R}, s, \chi_o, \mathcal{O}$)
  for all $r \in \mathcal{R}$ **do**
    $T \leftarrow$ DubinsTrajectory(χ_o, s, r)
    if IsTrajectoryCollisionFree($T, \mathcal{O}$) **then**
      return True ▷ Collision Free Trajectory Found
    end if
  end for
  return False
end function
function CollisionCheck($\tau, \mathcal{O}$)
  for all $o \in \mathcal{O}$ **do**
    if Colliding(τ, o) **then return** True
    end if
  end for
end function
function IsTrajectoryCollisionFree($T, \mathcal{O}$)
  for all $\tau \in T$ **do**
    if CollisionCheck($\tau, \mathcal{O}$) **then**
      return False ▷ T is unfeasible
    end if
  end for
  return True
end function

Alg. 1 is potentially computationally expensive since every configuration in every generated Dubins trajectory is checked for collision against every obstacle in the environment. This can interfere with the ticking rate of the Behaviour Tree, blocking its execution. During preliminary tests, the check was performed in $0.09s \pm 0.05s$ on a Laptop equipped with

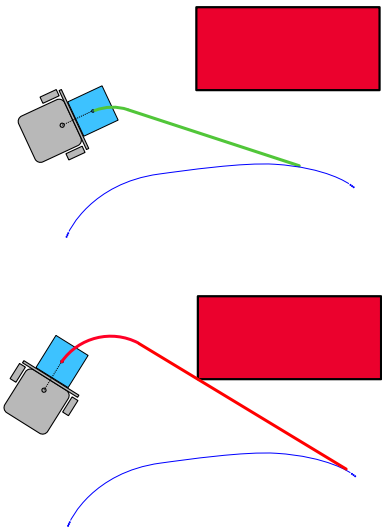

Fig. 3: Visualization of suitable (green) and unsuitable (red) trajectories for the system converging into the reference.

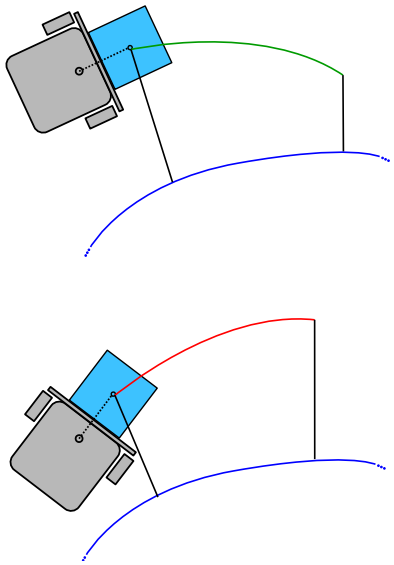

Fig. 4: Visualization of suitable (green) and unsuitable (red) pair of MPC prediction and reference.

a 9th generation Intel® Core™ i7, which is fast enough for the BT execution. However, as we will detail in Section V, this strategy is sensitive to noise in the object's position measurement, especially to errors in orientation. This results in potential false negatives that trigger unnecessary replanning.

To address this, we propose a second procedure to check the feasibility of a reference trajectory that leverages the underlying framework. During pushing operations, the MPC controller used to compute the control inputs produces a prediction of the expected future behavior of the system. This predicted trajectory is collision-free since obstacle avoidance is integrated in the controller, and updated at $10Hz$ independently of the BT's execution. We propose to use the MPC's predicted motion to identify if the controlled system can converge to the reference trajectory. Suppose the prediction's final position is not closer to the reference than its initial position, then it is safe to assume that the controller is not capable of following the desired trajectory. An example of suitable (green) and unsuitable (red) configurations is visualized in Fig. 4.

B. Requirement: Robot is in contact

The second requirement is that the robot needs to be in proper contact with the assigned object side to apply the expected range of pushing actions. The motion constraints are computed considering contact on the full side of the polygonal object. Having partial contact could lead to possibly unexpected behaviors during the pushing manipulation and lead to manipulation failure. Therefore, it is of utmost importance that the pushing contact between the robot and the object happens on the full length of the object's side. To identify this condition, we compute the relative position of the object to the robot and check if it belongs to a rectangular area. This condition is displayed in Fig. 5 for the case study of a mobile robot pushing a cube. The area is designed

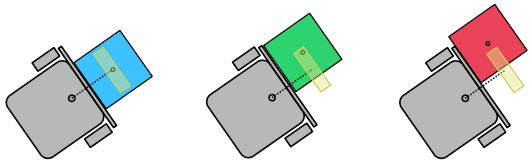

Fig. 5: Visualization of the proper contact between the robot and the object. The ideal contact is shown in blue. Borderline, but still suitable contact is shown in green while improper contact is shown in red.

to account for noise in the position of the object in the robot's X direction while sliding along the front bumper is allowed as long as the full side contact is maintained. If this requirement is not broken, the robot should interrupt the pushing operation and reposition itself in proper contact.

C. Tree Structure

The proposed structure for the Behavior Tree is visualized in Fig. 6. The Manipulation Task tree is defined using a *Fallback* node as its root with 3 children, the first of which is a condition to execute the task. This pattern composed by *Fallback, Condition, and Action* allows for the conditional execution of a specified action and is commonly used in building behavior trees. In this specific example, the pattern is used to specify the need to execute a manipulation task, since if the condition specified by the question *"Is the object at target?"* reports success, the whole root reports success and there is no need to execute a manipulation because the object is already at target. If the object is not at target, the *"Manipulate"* sub-tree is invoked as the part of the tree that is responsible for handling the manipulation in all its facets. The sub-tree starts with a *Sequence* node that ticks 3 children nodes. The first child is responsible for the first requirement: ensuring that a manipulation plan exists and that the robot

can perform it. The node is again created using the *Fallback, Condition, and Action* pattern. The *"Check Trajectory"* Node is responsible for determining if a trajectory is suitable for manipulation while the *"Compute Trajectory"* node interacts with the planner when necessary. Similarly, the second child of the *"Manipulate"* sub-tree takes care of the second design requirement: ensuring that the robot is positioned properly. If this requirement is not met, the robot should proceed to reposition itself, possibly interrupting the pushing sequence. The repositioning sequence, composed of 1) distancing from the object, 2) aligning with the correct side, and 3) moving in contact with the object is implemented using a *Sequence* node with memory. This node is a variation of the classical Sequence that remembers which children have returned *Success* or *Failure* and avoids re-executing the child until the whole Sequence reports *Success* or *Failure* or is terminated by a node higher in the tree. The sequence starts by checking if the robot is free to move by controlling the distance from the object. If the robot and the object are too close, the robot is instructed to move away from the object. This condition normally would collide with the proper contact condition since they can not be both verified at the same time. However, since the proper contact is higher in the hierarchy of the tree, it takes priority in defining which behavior the robot should follow. Once the robot is far enough from the object, the robot is then instructed to move to align itself with the side it needs to push onto. Once the robot is aligned, it can now move into contact. All these target positions for the robot are computed considering the object's position and dimensions other than the robot's position.

The third child of the "Manipulate" node is the "Push along trajectory" action. In this action, the robot is instructed to move pushing the object on its side, along the computed trajectory. This node is the lowest hierarchy node in the tree. Therefore, for it to be triggered, all the necessary conditions have to be present: a suitable plan that leads to the target is available and the robot is in contact with the correct side and ready to push. If any of these conditions become absent, the behavior tree reacts and attempts to correct it. The presented Behavior Tree is designed to be used by any robot as long as the assumptions are maintained, however, we acknowledge that the alternative condition for plan feasibility presented in Section IV-A is significantly tied to the underlying robotic framework and therefore not readily applicable to all scenarios. Nevertheless, the existence of other works in literature such as [27] that propose a predictive controller for pushing with different kinds of robots, prove that the presented BT can be adapted to other configurations.

V. Experimental Validation

The experimental evaluation of the proposed algorithm will be detailed in this section. First, the experimental setup will be described, followed by the implementation details of the proposed algorithm. While the proposed Behavior Tree is designed to be generally applicable to any robotic manipulator as long as it is equipped with a front bumper,

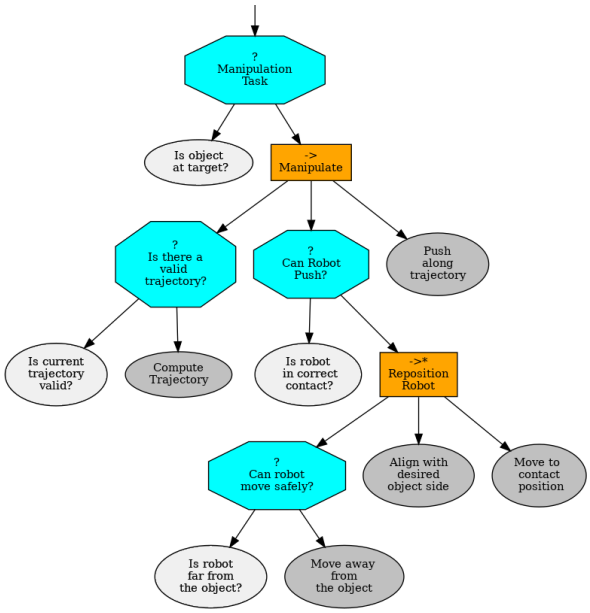

Fig. 6: Complete visualization of the BT structure used in the proposed solution. *Fallback* nodes are colored with cyan, *Sequence* nodes are colored orange, *Condition* nodes are colored light-gray, and *Action* nodes are colored dark-gray.

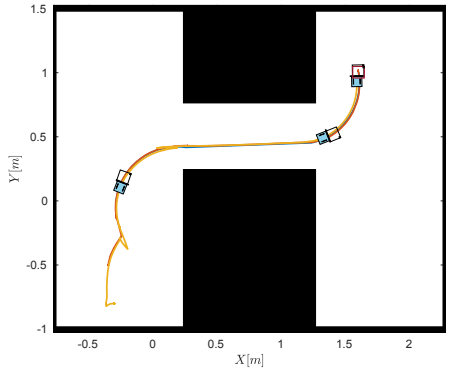

Fig. 7: Final result of the test.

in this manuscript we use a differential-drive mobile robot for its simplicity and ease of use in wide environments.

A. Setup

The testing environment comprises a planar arena equipped with the Optitrack [28] motion capture system. The pushing operations are performed using a Turtlebot3 equipped with a front bumper. The manipulation tests are performed on a square box with a side of 9 cm. The proposed behavior tree algorithm is implemented using the open-source library *py_trees* [29]. The algorithm interfaces with the robot controller and planning algorithm using Robot Operating System (ROS) [30]. The ticking frequency of the Behavior Tree is set to $200ms$. This choice allows the algorithm to quickly react to unexpected events without overloading the system. More details on the implementation of the planner and control algorithm are provided in [23].

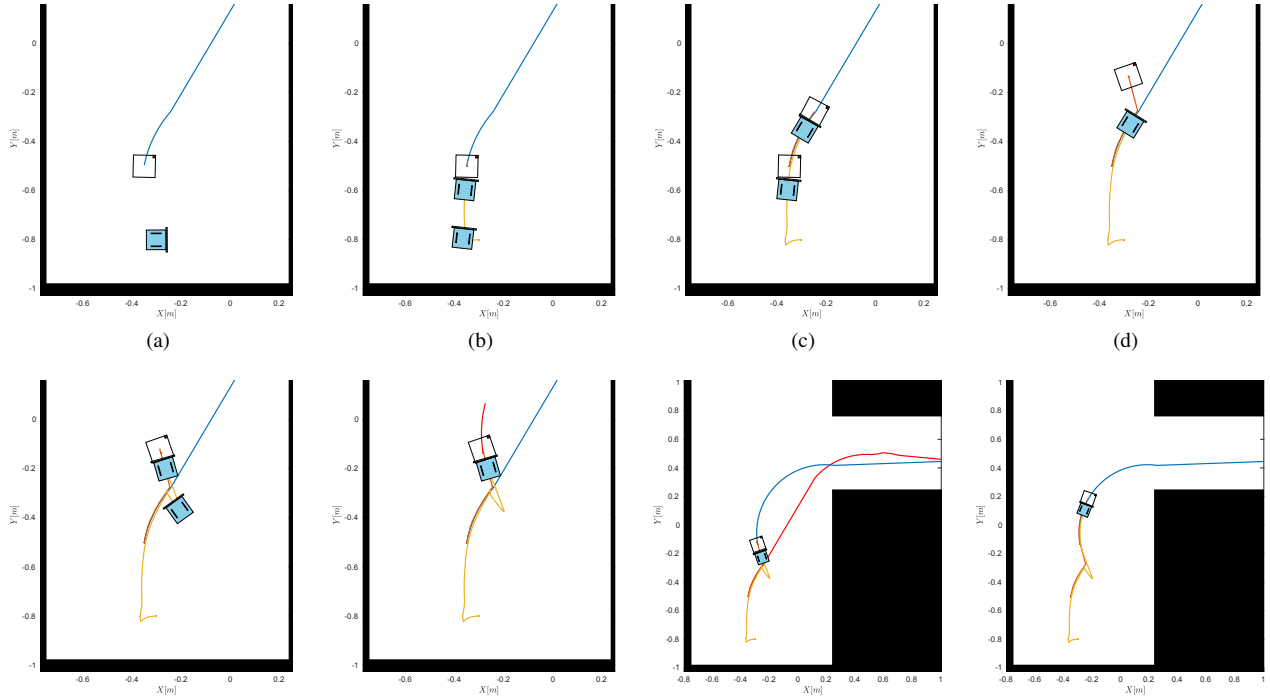

Fig. 8: Snapshots of test execution where the system is forced to react to external actions.

B. Validation

Before evaluating the coordination algorithm's efficacy, we initiated an initial test focused on examining the success rates of two specific plan feasibility conditions. This foundational test involved a manipulation task where the objective was to move an object from the bottom left to the top right corner of the environment, navigating through a tight passage. The findings from this initial examination are documented in Table I, which elucidates the false negative rate associated with each plan feasibility condition.

During this first test, the condition based on Dubin's curve reported a 40% false negative rate, indicating that unnecessary replanning was triggered in nearly half of the trials. This highlights the condition's conservative nature, often misjudging feasible plans as unfeasible.

Furthermore, the second condition, relying on MPC predictions, was scrutinized across various horizon lengths to evaluate its impact on performance. The test results revealed a superior performance is obtained with shorter horizons, showcasing a reduction in the incidence of false negatives. This improvement is attributed to the control horizon's influence, which only allows for control input adjustments during the initial stages of the prediction.

Following this preliminary assessment, a more detailed evaluation of the coordination algorithm was conducted through a specific manipulation task. The task demanded the transportation of an object across the environment, a challenge compounded by the interference of an external actor aiming to displace the object. The execution of this task is depicted through various stages in Figures 7 and 8.

TABLE I: False negative rate of plan feasibility condition.

Condition	FN Rate
Dubin's based	40%
MPC based T=2.5s	5%
MPC based T=5.0s	20%
MPC based T=7.5s	35%

Initially, a plan is generated (blue in Fig. 8a), followed by the robot aligning itself with the object (Fig. 8b) and initiating the push (Fig. 8c). An unexpected event, shown in Fig. 8d, interrupts the process as the object is forcibly moved, resulting in a loss of contact. The robot's strategic response involves realignment and repositioning (Fig. 8e), preceding another attempt at the task. However, the algorithm detects a deviation from the intended trajectory based on the MPC's prediction (Fig. 8f), necessitating a phase of replanning, the outcome of which is depicted in Fig. 8g. This allows the robot to adjust its strategy and continue along the corrected path. The successful completion of this manipulation task is illustrated in Fig. 7, showcasing the algorithm's capability to adapt and respond to dynamic challenges effectively.

VI. Conclusions

In this work, we developed a Behavior Tree-based coordination algorithm aimed at streamlining the process of robotic object pushing in complex and dynamic environments. In particular, we presented the conditions for the success of a pushing manipulation of having a feasible plan and enforcing the proper contact between the robot and the object, introducing two procedures to compute the former. Our experimental investigation began with an evaluation

aimed at discerning the reliability of two plan feasibility conditions. The first condition, based on Dubin's curve, revealed a significant propensity towards conservative assessments, with a 40% false negative rate indicating unnecessary replanning in numerous instances. This underscored the need for refinement in our feasibility evaluation to mitigate overcaution and enhance operational efficiency. In contrast, the second condition, leveraging MPC predictions, demonstrated a marked improvement in accuracy. Counterintuitively, our tests on the influence of horizon length on accuracy resulted in a degradation of performance with longer horizons. This was attributed to the control horizon's influence on the prediction. Our comprehensive testing further explored the algorithm's proficiency in navigating and manipulating objects through intricate environments, even amid external disruptions. This was epitomized in a scenario where an object was to be maneuvered across the environment, challenged by intentional interference. The algorithm's capability to dynamically realign and replan in response to unexpected deviations showcased its robust adaptability and the effectiveness of the underlying BT framework in managing complex manipulation tasks. Future endeavors will focus on generalizing the presented BT-based algorithm to a wider array of scenarios, increasing its versatility and robustness, particularly in handling edge cases and failure modes not covered in the current framework. Enhancing the algorithm's predictive capabilities and integrating advanced machine learning techniques could provide more nuanced control and planning strategies, allowing robots to anticipate and mitigate potential disruptions more effectively. Moreover, exploring the integration of sensory feedback mechanisms could further refine the robot's adaptability and precision in complex environments, proving the algorithm can work without external localization sources.

REFERENCES

[1] F. Ruggiero, V. Lippiello, and B. Siciliano, "Nonprehensile dynamic manipulation: A survey," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1711–1718, 2018.

[2] D. Prattichizzo and J. C. Trinkle, *Grasping*. Cham: Springer International Publishing, 2016, pp. 955–988. [Online]. Available: https://doi.org/10.1007/978-3-319-32552-1_38

[3] M. T. Mason, "Mechanics and planning of manipulator pushing operations," *The International Journal of Robotics Research*, vol. 5, no. 3, pp. 53–71, 1986. [Online]. Available: https://doi.org/10.1177/027836498600500303

[4] M. T. Mason and R. C. Brost, *Automatic Grasp Planning. An Operation Space Approach*. Boston, MA: Springer US, 1987, pp. 380–387. [Online]. Available: https://doi.org/10.1007/978-1-4684-6915-8_37

[5] M. Peshkin and A. Sanderson, "Planning robotic manipulation strategies for workpieces that slide," *IEEE Journal on Robotics and Automation*, vol. 4, no. 5, pp. 524–531, 1988.

[6] K. M. Lynch, "Locally controllable manipulation by stable pushing," *IEEE Transactions on Robotics and Automation*, vol. 15, no. 2, pp. 318–327, 1999.

[7] K. Harada, J. Nishiyama, Y. Murakami, and M. Kaneko, "Pushing Manipulation for Multiple Objects," *Journal of Dynamic Systems, Measurement, and Control*, vol. 128, no. 2, pp. 422–427, 04 2005. [Online]. Available: https://doi.org/10.1115/1.2199857

[8] J. Bernheisel and K. Lynch, "Stable transport of assemblies by pushing," *IEEE Transactions on Robotics*, vol. 22, no. 4, pp. 740–750, 2006.

[9] F. Bertoncelli, F. Ruggiero, and L. Sabattini, "Linear time-varying mpc for nonprehensile object manipulation with a nonholonomic mobile robot," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 11 032–11 038.

[10] S. Krivic and J. Piater, "Pushing corridors for delivering unknown objects with a mobile robot," *Autonomous Robots*, vol. 43, no. 6, pp. 1435–1452, 2019.

[11] S. Akella and M. T. Mason, "Posing polygonal objects in the plane by pushing," *The International Journal of Robotics Research*, vol. 17, no. 1, pp. 70–88, 1998. [Online]. Available: https://doi.org/10.1177/027836499801700107

[12] K. M. Lynch and M. T. Mason, "Stable pushing: Mechanics, controllability, and planning," *The International Journal of Robotics Research*, vol. 15, no. 6, pp. 533–556, 1996.

[13] O. Shahar and E. Rivlin, "To push or not to push: on the rearrangement of movable objects by a mobile robot," in *Proceedings of IEEE International Conference on Robotics and Automation*, vol. 1, 1996, pp. 159–164 vol.1.

[14] C. Zito, R. Stolkin, M. Kopicki, and J. L. Wyatt, "Two-level rrt planning for robotic push manipulation," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012, pp. 678–685.

[15] T. Meriçli, M. Veloso, and H. L. Akın, "Push-manipulation of complex passive mobile objects using experimentally acquired motion models," *Autonomous Robots*, vol. 38, no. 3, pp. 317–329, 2015.

[16] J. Zhou, Y. Hou, and M. T. Mason, "Pushing revisited: Differential flatness, trajectory planning, and stabilization," *The International Journal of Robotics Research*, vol. 38, no. 12-13, pp. 1477–1489, 2019. [Online]. Available: https://doi.org/10.1177/0278364919872532

[17] M. Colledanchise and P. Ögren, "How behavior trees modularize hybrid control systems and generalize sequential behavior compositions, the subsumption architecture, and decision trees," *IEEE Transactions on Robotics*, vol. 33, no. 2, pp. 372–389, 2017.

[18] ——, *Behavior Trees in Robotics and AI*. CRC Press, jul 2018. [Online]. Available: https://doi.org/10.1201%2F9780429489105

[19] U. Yang and J. Seo, "Behavior tree-based object event modeling with dynamic change control for virtual reality fire fighting training contents," in *2020 International Conference on Information and Communication Technology Convergence (ICTC)*, 2020, pp. 1647–1649.

[20] Y. Tanaka and S. Katsura, "Task switching model for acceleration control of multi-dof manipulator using behavior trees," in *IECON 2023- 49th Annual Conference of the IEEE Industrial Electronics Society*, 2023, pp. 1–6.

[21] Y. Ma, C. Deng, J. Zhang, Y. Wu, H. Jin, and Y. Wang, "Resource scheduling in behavior trees," in *2022 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, 2022, pp. 2287–2292.

[22] Q. Huang, X. Ma, K. Liu, X. Ma, and W. Pang, "Autonomous reconnaissance action of swarm unmanned system driven by behavior tree," in *2022 IEEE International Conference on Unmanned Systems (ICUS)*, 2022, pp. 1540–1544.

[23] F. Bertoncelli, F. Ruggiero, and L. Sabattini, "Non-prehensile planar manipulation: A framework for pushing with a single mobile robot," *submitted to International Journal of Robotics Research (IJRR)*, 2023. [Online]. Available: https://shorturl.at/drsQS

[24] K. Lynch, "The mechanics of fine manipulation by pushing," in *Proceedings 1992 IEEE International Conference on Robotics and Automation*, 1992, pp. 2269–2276 vol.3.

[25] L. E. Dubins, "On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents," *American Journal of Mathematics*, vol. 79, no. 3, pp. 497–516, 1957. [Online]. Available: http://www.jstor.org/stable/2372560

[26] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006.

[27] F. R. Hogan, E. R. Grau, and A. Rodriguez, "Reactive planar manipulation with convex hybrid mpc," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 247–253.

[28] NaturalPoint, "Motion capture systems." [Online]. Available: https://optitrack.com/

[29] "Py Trees," https://github.com/splintered-reality/py_trees, 2023, online.

[30] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, 2009.