Mapping, Navigation and Reconnaissance using TurtleBot

German Gonzalez[1], Saicharan Thirandas[1], Seung Hun Lee[1], Puja Chaudhary[1], Yiqi Dou[1]
Northeastern University, Boston, MA 02115, USA

***Abstract*—This paper presents a comprehensive system for autonomous navigation and exploration using a Turtlebot robot. The system integrates April Tag detection and tracking, SLAM using GTSAM factor graphs, frontier-based goal generation, and Tube MPPI control. April Tags enable reliable landmark localization for GTSAM that optimizes the robot's trajectory and pose. Frontier-based goal generation utilizes LIDAR and Camera fusion for exploration, and Tube MPPI generates the controls for smooth, collision-free trajectories. Simulations demonstrate the system's effectiveness in detecting and tracking April Tags, mapping, and exploration. Experimental results validate the system's performance, while limitations and future directions are discussed. This research contributes to the advancement of autonomous mobile robotics by providing a robust solution for navigation and exploration in unknown environments.**

You can find the code at: https://github.com/gmangonz/Tenacious-Turtles.

I. Introduction and Motivation

In recent years, the field of robotics has seen significant advancements in autonomous navigation and exploration [1], [2]. One of the key challenges in this domain is enabling robots to effectively navigate and map unknown environments while detecting and tracking specific landmarks [3]. In this project we develop a comprehensive system for a Turtlebot robot that integrates April Tag detection and tracking, Simultaneous Localization and Mapping (SLAM) using GTSAM factor graphs [4], frontier-based goal generation [5], and an advanced navigation algorithm called Tube Model Predictive Path Integral (MPPI) control [6].

April Tags, which are fiducial markers commonly used in robotics, play a crucial role in this project. The April Tag detection node utilizes the pupil-apriltags package to detect and classify these tags from the robot's camera feed. Given the camera's intrinsic parameters, the detector can estimate the tag's pose in 3D space. The detected tags are then tracked by the April Tag tracking node, which maintains accurate real-time positions of the tags and updates them as landmarks in the SLAM system.

The SLAM node is responsible for both localization and mapping. It uses GTSAM (Georgia Tech Smoothing and Mapping) [7], a library for factor graph optimization, to estimate the robot's pose and construct a consistent map of the environment.

To autonomously explore the environment, the frontier node generates goals based on the concept of frontiers, which are the boundaries between explored and unexplored regions [8]. Unlike the EXPLORE-LITE package [9], which relies solely on LIDAR data, the proposed approach considers the presence of April Tags to prioritize goal selection.

For efficient navigation, the Tube MPPI algorithm is employed. This extension of the Model Predictive Path Integral (MPPI) control scheme [10] uses a nominal system to create a path for the robot to follow and an ancillary system to minimize the deviation between the nominal and actual paths. The algorithm takes into account the presence of noise in the system and generates smooth, collision-fraee trajectories.

The project combines these modules to create a robust and efficient system for autonomous navigation and exploration. The system was tested in simulation using Gazebo, where it demonstrated its effectiveness in detecting and tracking April Tags, generating maps, and exploration. Although some limitations have been identified, such as computational complexity and hardware issues, the project shows promise for real-world applications.

In the following sections, we will delve into the details of each module, present the results obtained, discuss the limitations, and outline future directions for improving the system.

II. Proposed Solution

The proposed solution for autonomous navigation and exploration using a Turtlebot robot consists of several interconnected modules, as shown in Figure 1.

The primary components include April Tag detection and tracking, SLAM using GTSAM factor graphs, frontier-based goal generation, and Tube MPPI control for navigation.

The April Tag detection node utilizes the pupil-apriltags package to detect and classify April Tags from the robot's camera feed. The node converts the video stream to grayscale, detects the tags, estimates their poses using the camera's intrinsic parameters, and publishes the detected tags along with their respective poses. The April Tag tracking node then tracks the detected tags and maintains their accurate real-time positions. It adjusts the tag positions based on updated detections from the SLAM node and new transformations between the camera and the tags. The tracked tags are used as landmarks in the SLAM system.

The SLAM node is responsible for simultaneous localization and mapping. It employs the GTSAM library, which uses factor graphs to optimize the robot's trajectory and landmark positions. The node takes input from odometry and sensor data, including the tracked April Tags, and estimates the turtle's pose

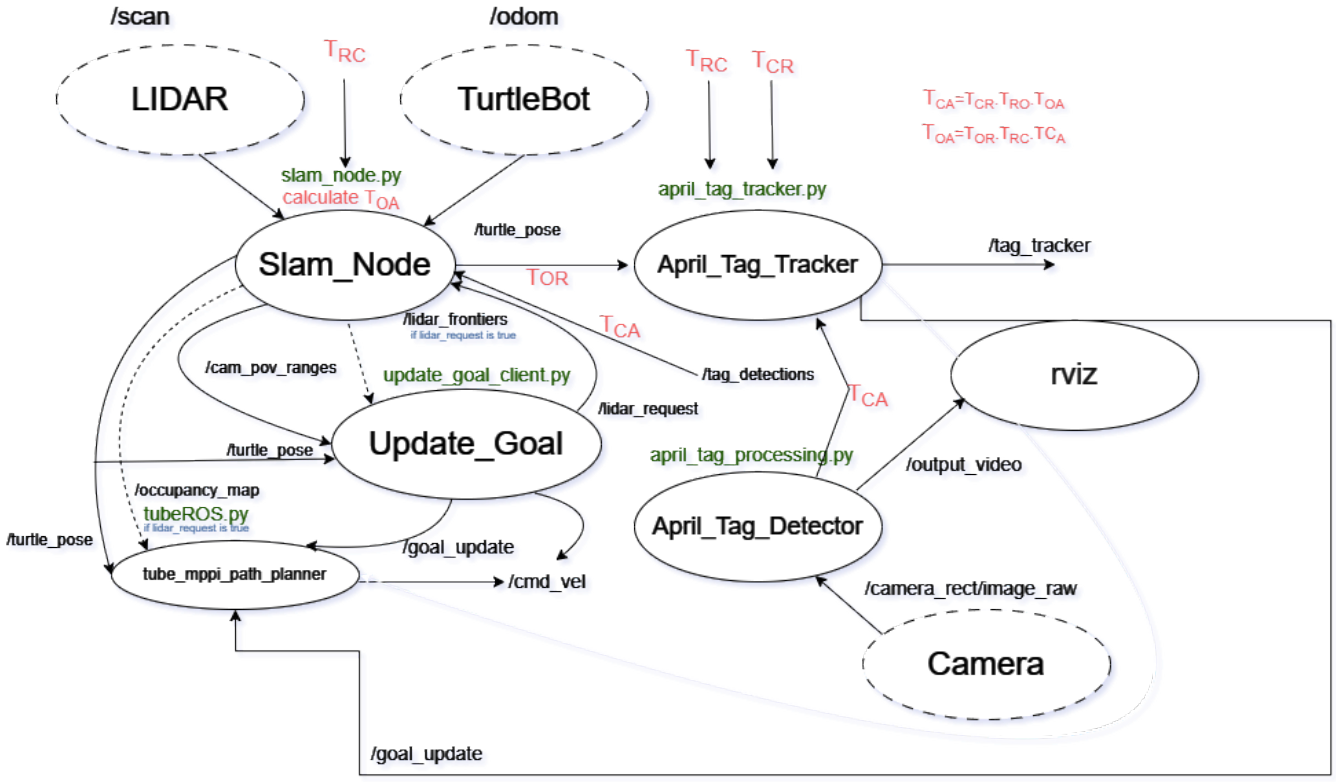

Fig. 1. Overview of the proposed autonomous navigation and exploration system.

and trajectory within the environment. Within the SLAM node, LIDAR data is processed for obstacle avoidance and frontiers generation. We take into consideration the camera's field of view in our frontier generation for better AprilTag detection.

Taking input from the SLAM and LIDAR node, the frontier node prioritizes goal selection based on the presence of April Tags, assigning higher priorities to goals associated with detected tags. This approach ensures that the robot explores the environment efficiently while considering the locations of the tags.

For navigation, the Tube MPPI algorithm is employed. This extension of the MPPI control scheme uses a nominal system to create a reference path for the robot to follow and an ancillary system to minimize the deviation between the nominal and actual paths. The algorithm takes into account the presence of noise in the system and generates smooth, collision-free trajectories. The navigation node receives goal points from the frontier node and uses the Tube MPPI algorithm to plan and execute the robot's motion.

Our overall system leverages the strengths of each component, such as the reliability of April Tags for localization, GTSAM factor graphs for pose estimation, the effectiveness of frontier-based goal generation for exploration, and the robustness of Tube MPPI control for navigation in the presence of noise.

In the following sections, we will present the implementation details of each module, discuss the experimental results obtained through simulations, analyze the limitations of the current system, and propose future directions for improvement.

III. System Design

A. Turtlebot 3 - Transformations and Camera Setup

The Turtlebot3 has a versatile design optimized for various environments, leveraging its two parallel wheels, each with a radius of 0.030 meters, and a track width of 0.15 meters for smooth and precise movement. Integrating a camera into the Turtlebot's URDF file, we incorporated it into the robot's kinematic chain. This integration ensures coordination between the camera's frame and the robot's frame, enhancing its performance in tasks relying on visual perception and spatial awareness. We derived the necessary transformations directly from the URDF file. This approach ensured accurate alignment and calibration of the camera, streamlining the setup process while maintaining precision in visual data interpretation.

To calibrate the camera, we followed the ROS tutorial for Monocular Camera Calibration. Using a 9×7 checkerboard, we obtained calibration data stored in an ost.yaml file. This file contains information such as the camera matrix, distortion coefficients, and rectification parameters. It plays a vital role in correcting any distortion in the camera's images, thus improving the accuracy of visual data used in tasks.

B. GTSAM

The GTSAM (Georgia Tech Smoothing and Mapping) approach is employed to enhance the localization and mapping capabilities of the TurtleBot3. GTSAM is a powerful optimization library that uses factor graphs to efficiently solve large-scale SLAM problems [11]. In this approach, the SLAM problem is represented as a factor graph, where each node in the graph corresponds to either a robot pose or a landmark position, and edges between nodes represent constraints derived from sensor observations or robot motion.

The factor graph is incrementally constructed as the robot navigates through the environment. The initialization of the graph includes a prior pose factor, which serves as an anchor to prevent drift in the absence of absolute measurements. As the robot moves, each subsequent pose is connected to the previous pose through odometry measurements, forming a sequential chain of pose nodes. These odometry factors encode the relative motion between consecutive poses based on wheel encoder readings or other proprioceptive sensors.

When AprilTags are detected in the environment, they are treated as landmarks and added to the factor graph. The detection of an AprilTag provides a relative pose measurement between the robot's current pose and the tag's position. These measurements are represented as edges in the graph, connecting the corresponding robot pose node to the landmark node. The AprilTag observations help to constrain the robot's pose and reduce uncertainty in the overall SLAM solution.

Figure 2 illustrates the structure of the factor graph used in the GTSAM approach for the TurtleBot3. The factor graph includes pose nodes (Pose 0, Pose n, Pose m) and landmark nodes (L(id n)). The edges represent the transformations between consecutive poses $(\Delta X_1, \Delta X_1 + \Delta X_2)$ and the transformations between the robot's pose and the detected AprilTags.

As the robot explores the environment and collects more measurements, the factor graph grows in size. GTSAM performs incremental optimization by minimizing the error between the predicted and observed measurements. The optimization process adjusts the robot's pose with respect to the origin and landmark positions to find the most likely configuration that satisfies all the constraints in the graph.

C. April Tag Detection and Tracking

As mentioned in the previous section, when AprilTags are detected in the environment, the detection of an AprilTag provides the relative pose measurement between the robot's camera and the tag's position. Using this information and the predicted pose from GTSAM, we can retrieve the tag's position with respect to the origin.

The transformations between the poses and tags detected can be expressed using the following equations:

$$T_{OA} = T_{OR} \cdot T_{RC} \cdot T_{CA}$$
$$T_{CA} = T_{CR} \cdot T_{RA}$$

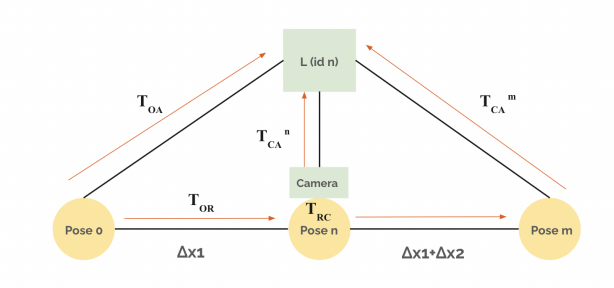

Fig. 2. Factor graph representation in the GTSAM approach for TurtleBot3 SLAM. Nodes represent robot poses and landmarks, while edges represent constraints based on odometry and AprilTag observations.

where T_{OA} represents the transformation from the origin to the AprilTag, T_{OR} is the transformation from the origin to the robot, T_{RC} is the transformation from the robot to the camera, T_{RA} is the transformation from the robot to the April tag and T_{CA} is the transformation from the camera to the AprilTag. As the robot continues to explore, knowing previously detected tag's T_{OA} helps it keep track of their position at all times.

D. Frontier Node

As outlined earlier, our exploration strategy relies on two crucial components: lidar maps and frontier detection. Firstly, we generate a lidar-based occupancy map that provides a comprehensive view of the environment, while also generating a second map that only considers the camera's field of view, ensuring real-time information on surrounding obstacles and terrain.

Using these maps, we identify frontiers, which are the boundary regions between explored and unexplored areas. These frontiers represent the areas where the robot can potentially explore further. To refine our exploration path, we employ XOR (exclusive OR) operations between the two frontier maps. This operation allows us to generate a refined map highlighting the remaining unexplored frontiers.

With the refined map in hand, we proceed to select specific points for exploration. To do this, we randomly sample a small percentage, typically 5 percent of the points from the binary map. This random sampling ensures a diverse selection of potential exploration points across unexplored regions.

Each sampled point's distance from the robot's current pose is calculated. This distance metric helps us prioritize which frontier points to explore first. We have two options for selection: either we choose the point with the minimum distance from the robot's current position or opt for the point closest to the median distance. This selection process ensures that we efficiently navigate toward unexplored areas while considering both the proximity and distribution of exploration points.

To visualize this process, refer to the accompanying image, which illustrates the steps involved in frontier detection, point selection, and exploration path refinement. This iterative approach enables our robot to systematically explore its

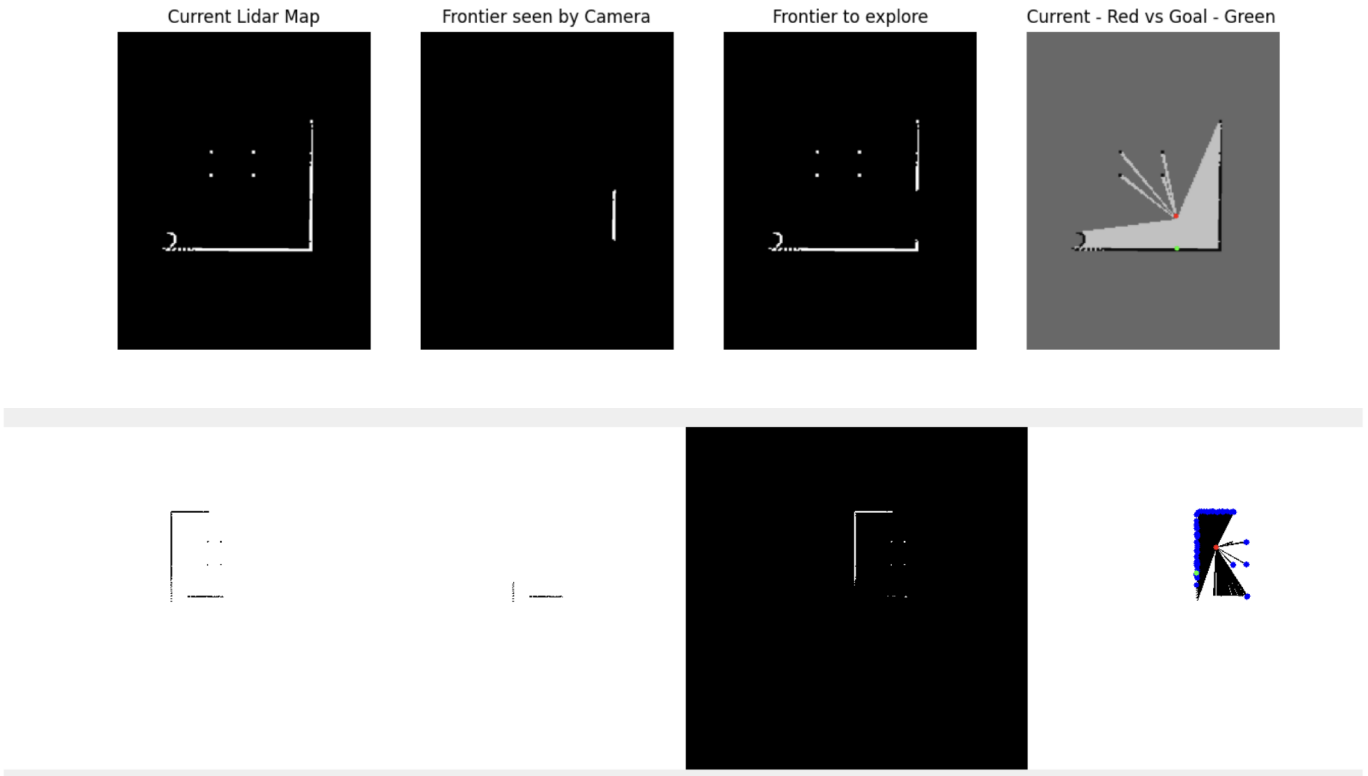

Fig. 3. Summary of the autonomous navigation and exploration system proposal: The first row presents the starting situation within the gazebo house, while the second row highlights the samples generated as potential goals in the final image. Red indicates current position and green indicates the selected target

environment, effectively mapping out uncharted territories while optimizing its exploration trajectory.

E. Goal Generation and Goal Detection

The strategy is twofold: firstly, the system incorporates a 'Goal Reset' mechanism which prompts the robot to revise its target when it finds itself close to an obstacle in the camera's point of view, enhancing its adaptability to sudden environmental changes. Secondly, the system employs 'April Tag Detection and Tracking' wherein the robot utilizes onboard sensors and cameras to detect April Tags within its vicinity. Upon recognition, the robot immediately updates its destination, with the tags' spatial data directly influencing the robot's trajectory. This method not only refines the goal-directed movement based on immediate sensor input but also prioritizes AprilTags as goals. The navigation and frontier nodes, depicted in the following graphic, symbolize decision points and the edges of the explored territory, respectively, playing a key role in path planning and exploration tasks. This adaptive goal management system is designed to optimize the robot's path efficiency, reduce operational time, and ensure high reliability in goal-oriented tasks.

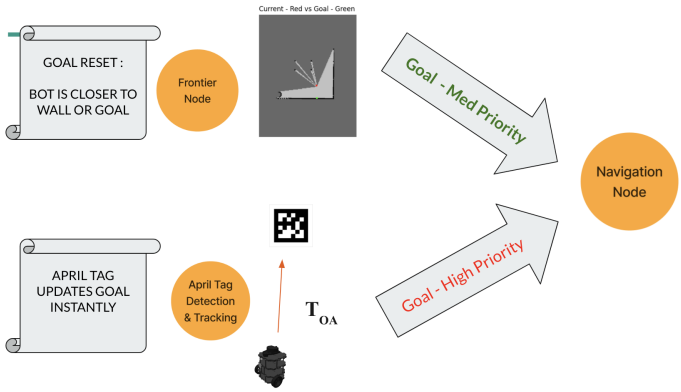

Fig. 4. Goal update and goal reset.

F. TubeMPPI

Our Tube MPPI implementation is based on the paper from Williams et al. [12] which uses the Tube MPC scheme described by Mayne and Kerrigan [13]

1) Tube MPC: We assume the system we want to control is described by some non-linear function with a bounded disturbance:

$$\dot{x} = f(x, u) + \omega$$

Where x is the current state, u is the current action and ω is the bounded disturbance. This system is also subject to the following constraints:

$$x \in \mathbb{X}, \quad u \in \mathbb{U}$$

where $\mathbb{X}$ and $\mathbb{U}$ are compact sets containing the origin of their interior.

2) Nominal Controller: The nominal system is defined by:

$$\dot{z} = f(z, v)$$

where z is the current nominal state and v is the current nominal action. This is equivalent to the uncertain system $\dot{x}$ without the added disturbance. The cost function associated with the nominal system is defined by:

$$\bar{V_T}(z, \mathbf{v}) = V_f(z(N)) + \sum_{k=0}^{T-1} \ell(z(k), v(k))$$

where z is the initial state, $\mathbf{v}$ is control sequence found by the nominal controller, Q and R are positive definite, and V_f is the cost of the final state. This is a quadratic cost with respect to the states and actions of the nominal system defined as:

$$\ell(x, u) = (1/2)x^T Q x + (1/2)u^T R u$$

The nominal system is subject to similar constraints as the system to be controlled:

$$z \in \mathbb{Z}, \quad v \in \mathbb{V}$$

Where $\mathbb{Z} \subset \mathbb{X}$ and $\mathbb{V} \subset \mathbb{U}$. Thus, the nominal problem is defined as minimizing the above cost function with respect to the nominal controls $\mathbf{v}$ for all time steps T.

3) Ancillary Controller: The ancillary controller is used in order to keep the states of the uncertain system close to those of the nominal controller. To do this, our ancillary controller is based on the following two systems:

$$\dot{x} = f(x, u)$$
$$\dot{z} = f(z, \mathbf{v})$$

Where $\dot{x}$ is the system we want to control with no added noise and $\dot{z}$ is the solution of the nominal system. Since the goal of the ancillary controller is to keep the uncertain system close to the trajectory of the nominal system, the cost function is defined as the difference between the two trajectories:

$$V_T(x, z, \mathbf{u}) = \sum_{k=0}^{T-1} \ell(x(k) - z^*(k, z), u(k) - v^*(k, z))$$

where $x(k)$ and $u(k)$ are from the solution of the ancillary system at time k and $z^*(k, z)$ and $v^*(k, z)$ are the nominal solutions at time k with initial state z. The ancillary problem is then defined as the minimization of this cost function with respect to $\mathbf{u}$ for all time steps $T - 1$ where $\mathbf{u}$ is subject to the control constraints and the terminal state of the ancillary problem must be the same as the terminal state of the given nominal problem.

4) Tube MPPI: We will use two controllers to send to the system: MPPI for the nominal controller and iLQR for the ancillary controller. For the system to be controlled, we use the discrete Unicycle model described by the difference equation:

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \\ \theta_{k+1} \end{bmatrix} = \begin{bmatrix} x_k \\ y_k \\ \theta_k \end{bmatrix} + dt \begin{bmatrix} \cos(\theta) & 0 \\ \sin(\theta) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \dot{\theta} \end{bmatrix}$$

5) MPPI: MPPI is a sampling-based method for finding the optimal controls to a system. To do this, we sample many roll-outs with random controls subject to the constraints to the system. We calculate the cost associated with each state and action given by each of these roll-outs and take its sum. Then, the controls for each roll-out are weighted inversely to its cost, so higher cost roll-outs have less impact on the final controls than the lower cost roll-outs. Then, only the first control is sent to the uncertain system. Finally, the actions found in this iteration are used in the next iteration.

Algorithm 1 Model Predictive Path Integral (MPPI)

Parameters: K number of samples, T number of timesteps

Initialize $U = (0, \ldots, 0)$
while not done **do**
  $x_0 \leftarrow$ new state from system
  for k in $1 : K$ **do**
    $x \leftarrow x_0$
    Sample $T - 1$ random actions $\mathcal{E}^k = (\varepsilon_0^k, \ldots, \varepsilon_{T-1}^k)$
    Apply random actions and calculate cost C_k of rollout
  end for
  for each rollout cost C_k **do**
    $w_k \leftarrow \exp(-\frac{1}{\lambda}(C_k - \min_k(C_k))$
    $\eta += w_k$
  end for
  for t in $1 : T - 1$ **do**
    $U \leftarrow U + \frac{1}{\eta} \sum_{k=1}^{K} w_k \mathcal{E}^k$
  end for
  $X \leftarrow$ simulate(U)
  Publish(X, U)
end while

6) iLQR: To solve the ancillary problem, iterative linear quadratic regulator (iLQR) as described by Tassa, Mansard and Todorov is used [14]. iLQR is a lightweight controller which linearizes the dynamics of the system and iteratively makes forwards and backwards passes in order to find a local minimum for the given cost function. As described above, the cost function for iLQR has been modified to be the difference between the states and actions of MPPI and the states and actions found by iLQR.

7) Combining the Controllers: With MPPI as our nominal controller and iLQR as our ancillary controller the Tube MPPI algorithm is as follows:

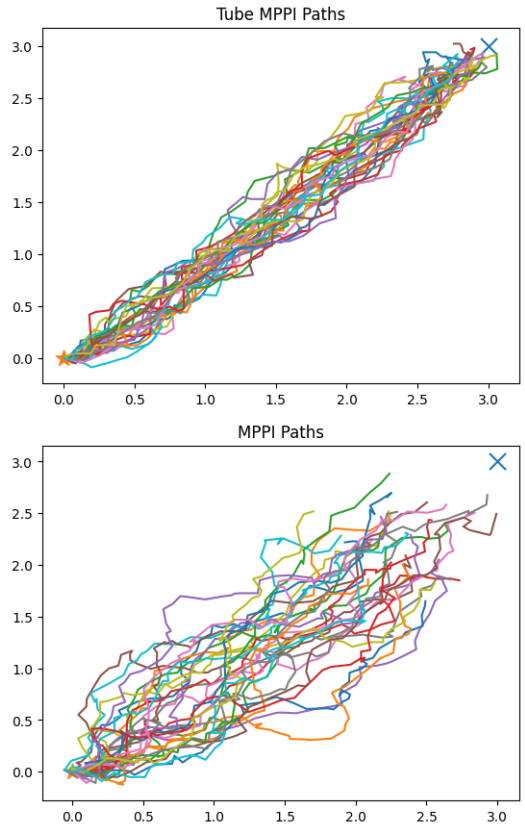

Fig. 5. Plotted are 30 runs of Tube MPPI and MPPI on the Noisy Unicycle for 50 timesteps. The starting position is represented by a orange star, and the goal position is represented by a blue x.

Algorithm 2 Tube Model Predictive Path Integral (Tube MPPI)

Data: Initial state x_0

Set $(x, z) \leftarrow (x_0, x_0)$
while not done **do**
  Determine $\mathbf{z}$ and $\mathbf{v}$ from MPPI with initial state z
  Solve the ancillary problem with iLQR and $\mathbf{z}, \mathbf{v}$
  Apply control u_0 found by iLQR to the system being controller and measure new state x^+
  Set new z to be the first state after the first control found by MPPI: $z^+ \leftarrow z(1, z)$
  Set $(x, z) = (x^+, z^+)$
end while

8) Moving to Waypoint With Noisy System: In this example, we move a noisy unicycle model from $(0, 0)$ to $(3, 3)$. The noise added to the velocity control is $\mathcal{N}(0, 0.5)$ and for the angular control $\mathcal{N}(0, 4)$. The noise added to the x and y coordinates comes from a uniform distribution between $[-0.06, 0.06]$ and for the direction of the unicycle model $[-0.3, 0.3]$. The paths traced by Tube MPPI and MPPI are found in Figure 5 Tube MPPI handles the noise much better than MPPI and reaches the goal within the allotted amount of time steps while MPPI fails to reach the goal. The trajectories of MPPI are also much more varied than the Tube MPPI paths, leading to drastic movements from the unicycle model.

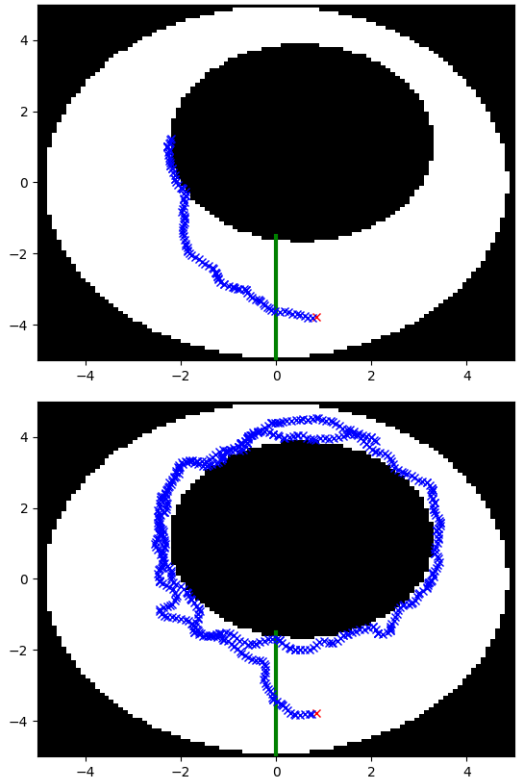

Fig. 6. The top plot is a run around the racetrack using MPPI and the noisy unicycle. The bottom plot is a run around the racetrack using Tube MPPI.

9) Driving Around a RaceTrack: In this example, we move a noisy unicycle around in a circular racetrack. The noise added to the controls are $\mathcal{N}(0, 0.5)$ for the velocity control, $\mathcal{N}(0, 2)$ for the angular control, $[-0.04, 0.04]$ for the x and y positions and $[-0.2, 0.2]$ for the direction the unicycle model is facing.

Figure 6 shows the noisy unicycle model being driven around a racetrack by Tube MPPI and MPPI. Tube MPPI still struggles to stay completely unphased by the noise, but does a much better job than MPPI, which crashes into an occupied space very early.

IV. Results

A. Environment Setup

The Turtlebot3 was subjected to testing in both physical and simulated environments, namely the Gazebo house. This section delves into the outcomes of these tests.

In both environments, the detection of April tags was notably robust. The incorporation of a second lidar map alongside the camera viewpoint effectively mitigated blind spots. Enclosed below are select results showcasing the successful recognition of April tags in both settings, depicted in the accompanying Figure 7.

Regarding mapping, a custom algorithm was deployed. In the Gazebo house, characterized by its intricate layout, the robots managed to map four out of five rooms. However, an oversight occurred on the opposite end of one room, revealing a potential

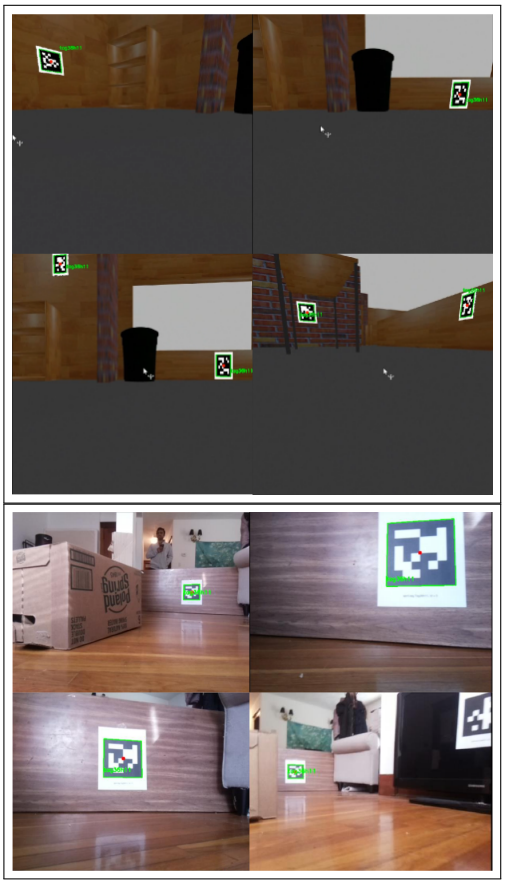

Fig. 7. The top plot showing the April tag detection in gazebo house and physical environment for the turtlebot testing.

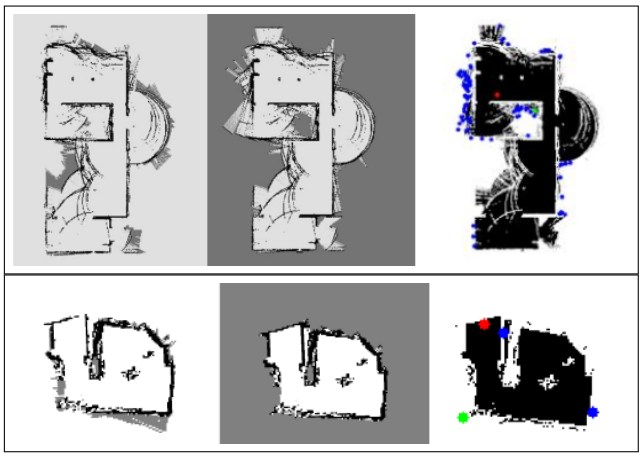

Fig. 8. The top plot shows the Mapping in the gazebo house and the bottom is in the physical environment. From the left - the first map is a camera pov lidar map followed by the complete lidar map and the final map indicates the sampling for the goal point in the frontier node.

area for enhancement in the frontier exploration algorithm. Optimizing this algorithm to better handle unoccupied frontiers could lead to improved exploration outcomes.

In the physical environment, which is comparatively smaller, complete mapping of the area was achieved successfully. Nonetheless, irregularities in the mapping were observed due to gaps in the cardboard boxes, highlighting a challenge in mapping complex environments with physical obstacles.

V. Limitations and Future Scope

A. LIDAR

As building the occupancy map from LIDAR and publishing 2 maps at 400x400 resolution is a relatively slow process in comparison to the other components, the resulting occupancy maps were quite noisy and often showed drift. Therefore, to combat this, we decided not to compute and publish the occupancy maps at every timestep. Using the update goal node, we recompute the occupancy maps only when there is a request sent for additional LIDAR information. However, we still perform obstacle detection using only a small subset of the LIDAR's angle ranges. This improves the LIDAR data processing but does, at times, limit the robot's navigation as it doesn't have a full picture of the environment at all times.

B. GTSAM

As GTSAM relied on the odometry of the robot, it had issues with drifting the predicted position from the true position. When the robot got stuck on walls, as the wheels continued to rotate, the odometry measurements would reflect these rotations and not the actual robot's movement. One possible solution we would like to implement to increase GTSAM's robustness would be to incorporate GTSAM with IMU data through its PreintegratedImuMeasurements class. Taking into account the robot's acceleration, we can improve GTSAM's predicted poses.

C. Odometry and the Motion Model

Although all tags were successfully detected, the use of TubeMPPI controls resulted in slower operations. The time taken to detect all tags was notably higher compared to traditional TubeMPPI methods. Moreover, the process involved multiple interruptions, necessitating pauses after each control iteration to await TubeMPPI responses, leading to occasional jerky movements.

Additionally, in simulation, the robot's motion was unpredictable and unreliable, making it arduous for TubeMPPI to correctly navigate the robot to the goals. Sending a simple move forward command resulted in notable rotation at times. When moving to the physical Turtlebot, the movement was more reliable. However, our motion model needed to be limited to the turtlebot's maximum speed of 0.22 m/s. This resulted in most of the linear velocity commands sent to the robot being either -0.22, 0.11, or 0.22, which meant the navigation was not very smooth.

VI. Conclusion

In conclusion, the results showcased reliable working of the Turtlebot3 in both physical and simulated environments, particularly in the detection of April tags. By integrating a second lidar map alongside the camera viewpoint, blind spots were effectively mitigated, enhancing the overall performance

of tag recognition. Mapping in the Gazebo house showcased successful mapping of four out of five rooms, albeit with a minor oversight that revealed a potential area for enhancement in the frontier exploration algorithm. In the physical environment, complete mapping was achieved successfully, though irregularities were observed due to obstacles, highlighting challenges in mapping complex physical environments. These findings underscore the importance of further refining algorithms to improve navigation and mapping accuracy in diverse environments.

However, several limitations were identified, suggesting avenues for future research. Processing LIDAR data was found to be relatively slow, resulting in noisy occupancy maps with drift. Incorporating IMU data into the GTSAM algorithm could enhance its robustness by providing more accurate predicted poses, especially in scenarios where odometry measurements are prone to drift. Additionally, while TubeMPPI controls were effective in tag detection, they resulted in slower operations and occasionally jerky movements, particularly in simulation. Improving the motion model to better reflect the Turtlebot's capabilities could lead to smoother navigation and better performance, both in simulation and physical environments. Addressing these limitations will be crucial for advancing the Turtlebot's capabilities in real-world applications.

VII. References

References

[1] S. Thrun, *Robotic mapping: A survey*. Morgan Kaufmann, 2002, pp. 1–35.

[2] C. Cadena *et al.*, "Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age," *IEEE Transactions on Robotics*, vol. 32, no. 6, pp. 1309–1332, 2016.

[3] J. Fuentes-Pacheco, J. Ruiz-Ascencio, and J. M. Rendón-Mancha, "Visual simultaneous localization and mapping: a survey," *Artificial Intelligence Review*, vol. 43, no. 1, pp. 55–81, 2015.

[4] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part i," *IEEE Robotics & Automation Magazine*, vol. 13, no. 2, pp. 99–110, 2006.

[5] B. Yamauchi, "A frontier-based approach for autonomous exploration," in *1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA'97. 'Towards New Computational Principles for Robotics and Automation'*, 1997, pp. 146–151.

[6] G. Williams *et al.*, "Information theoretic mpc for model-based reinforcement learning," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 1714–1721.

[7] S. Thrun, W. Burgard, and D. Fox, *Probabilistic robotics*. MIT Press, 2005.

[8] B. Yamauchi, "Frontier-based exploration using multiple robots," in *Proceedings of the second international conference on Autonomous agents*, 1998, pp. 47–53.

[9] "Explore lite," http://wiki.ros.org/explore_lite, accessed: 14-Apr-2023.

[10] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou, "Aggressive driving with model predictive path integral control," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 1433–1440.

[11] F. Dellaert and M. Kaess, "Factor graphs for robot perception," *Foundations and Trends in Robotics*, vol. 6, no. 1-2, pp. 1–139, 2017.

[12] G. Williams, B. Goldfain, P. Drews, K. Saigol, J. Rehg, and E. Theodorou, "Robust sampling based model predictive control with sparse objective information," in *Robotics*, ser. Robotics: Science and Systems, H. Kress-Gazit, S. Srinivasa, T. Howard, and N. Atanasov, Eds. United States: MIT Press Journals, 2018, funding Information: This work was made possible by the ARO award W911NF-12-1-0377, the Georgia Tech Vertical Lift Center of Excellence (VLCROE), and the Qualcomm Innovation Fellowship. Publisher Copyright: © 2018, MIT Press Journals. All rights reserved.; 14th Robotics: Science and Systems, RSS 2018 ; Conference date: 26-06-2018 Through 30-06-2018.

[13] D. Mayne, "Tube-based robust nonlinear model predictive control," vol. 40, 08 2007, pp. 36–41.

[14] Y. Tassa, N. Mansard, and E. Todorov, "Control-limited differential dynamic programming," *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1168–1175, 2014. [Online]. Available: https://api.semanticscholar.org/CorpusID:6269854

VIII. Contribution of Team Members

Each member of the research team played a crucial role in the development and implementation of the proposed methodologies. The contributions of each team member are outlined below:

- German Gonzalez: Worked on developing a node for April Tag detection transformations with respect to the origin and a custom node with GTSAM, with mapping to two custom maps. He also contributed to overall integration and debugging tasks and assisted in integrating TubeMPPI as a ROS Node.

- Saicharan Thirandas: Worked on a custom ROS node for frontier detection and goal update strategy. He resolved issues and set up TurtleBot for physical simulation, in addition to participating in overall integration and debugging tasks.

- Seung Hun Lee: Worked on TubeMPPI and its implementation.

- Puja Chaudhary: Worked on a collision avoidance algorithm in the navigation node and integrated the TubeMPPI algorithm with the ROS Node.

- Yiqi Dou: Performed camera calibration and transformations setup and tested algorithms to select the best-performing one.

Each team member's work was crucial in achieving the objectives of the project, and collaborative efforts resulted in the successful completion of the project.