Mobile Manipulation and Task Execution of a Tiago Robot

Kate Pendavinji

University of Colorado Boulder

CSCA 5342: Robotic Path Planning and Task Execution

Instructor *Nikolaus Correll*

April *22, 2025*

Introduction: Mobile Manipulation and Task Execution of a Tiago Robot

Requirements: Defines project requirement specifications

System Architecture: Navigation, planning, manipulation, and controller behaviors and implementation

Results: Robot's behavior outcomes

Discussion: Improvements and recap of met requirements

Abstract

This report presents an implementation that develops mobile manipulation for a Tiago robot whose goal is to pick and place a jar in the Webots' default kitchen environment. The system integrates previous robotic concepts such as planning, navigation, mapping, and manipulation in a behavior tree to achieve the desired basic implementation behavior of picking and placing a stationary jar onto a table. The robot runs the A* algorithm to prioritize the shortest path for the robot and controls the robot with a proportional controller. The jars are perceived by color from the robot's camera. Experimental trials demonstrate a reliable jar pick and place that satisfy the base-tier requirements.

Keywords: Webots, manipulation, arm control, pick up, navigation, planning, A*, shortest path, task execution

Mobile Manipulation and Task Execution of a Tiago Robot

This project develops a comprehensive mobile manipulation pipeline for a Tiago robot in the Webots default kitchen environment. The robot must autonomously navigate from its start pose to a single (for the basic implementation) pre-placed jar on the countertop without altering its initial position. The jar must then be transferred onto the kitchen table.

Requirements

To satisfy base tier requirements the solution reuses existing navigation and mapping implementations used to navigate the robot around the kitchen counter. It implements A* path planning over a configuration space and drives the robot with a proportional controller. The behavior tree orchestrates these modules and ensures a repeatable operation of picking and placing a jar for viewers. To grasp the jar the robot uses a joint controller to manipulate its body to move the arm using predefined poses. The robot also has an implementation of object identification which will be further explored in the discussion.

This document outlines the system architecture by exploring the foundations of planning and control strategies. It will summarize the experimental results for the jar pick and place and discuss the trade offs of different implementations and shortcomings of the code.

System Architecture

The system follows a similar architecture to previous projects. It begins with a controller which utilizes a *behavior tree* for task execution. The behavior tree uses a single sequence to phase through mapping, planning, navigation and manipulation. The phases are memory enabled in order to ensure task execution. A highlight of the behavior tree is the usage of the blackboard pattern. The controller uses the blackboard to write waypoints, joint configurations, and perception.

Below is a brief listed overview of the functionality of these phases:

1. *Navigation*:
    a. Reads a waypoint list generated from testing marker location.

   b. As time step moves forward, robot uses GPS and compass to calculate crucial values such as rho, alpha, and more to ensure angular and linear stability when driving.
   c. Proportional control applied to wheel velocities to maintain smooth movement.
   d. Applies helpers such as stopping the robot and backing up for manipulation control.
2. *Planning*:
   a. Reads start pose and attempts to achieve goal via configuration space.
   b. Runs A* algorithm to produce a list to convert to the robot's world waypoints and writes it to blackboard.
3. *Mapping*:
   a. Translates lidar and GPS outcomes.
   b. Builds occupancy grid with inflation.
   c. This implementation already has cspace generated.
4. *Manipulation*:
   a. Arm control implemented to move joints toward pre-defined target poses.
   b. Get object uses force feedback to retrieve the object by claming grippers.
   c. Release object drops off object by manipulation of gripper fingers.
   d. Prepare gripper recognizes the object but doesn't contribute to the overall implementation at this time.

Navigation and Planning

The navigation class in this implementation uses real world GPS frames intialized as

$$x_w = GPS_x,\ y_w = GPS_y$$

and angle heading as

$$\theta = arctan2(compass_x, compass_y).$$

It then calculates the distance and angle to waypoints using rho and alpha:

$$\rho = \sqrt{(x_w - x_{wp})^2 + (y_w - y_{wp})^2}$$

$\alpha = arctan2(y_{wp} - y_w, x_{wp} - x_w) - \theta.$

These compute the errors to the desired waypoints initialized in the controller method. Alpha is then normalized from -π to π. The markers are then set for *visualization purposes only*.

Once the locations are initialized the navigator applies proportional control using these values. Gains apply heading and distance as

$p_1 = 4, p_2 = 2$

which is applied to calculate the velocities of each wheel:

$v_{left} = -p_1\alpha + p_2\rho$

$v_{right} = +p_1\alpha + p_2\rho.$

These values are clamped to the maximum velocity, 6.28. The velocities are then actuated to the motors which is thoroughly documented in the code. Rho is then checked for thresholding to indicate waypoint arrival success.

In addition, the navigator has helper functions allowing it to complete to a complete stop upon termination of the task. It also has a function outlining a back up method for the robot to be able to back away from the counter upon picking up the object. It enables a counter that maintains timestep ticks and controls the wheels with a negative velocity to produce a backing up motion.

This implementation is then translated to the planning implementation where the robot is given a defined goal (waypoints) to achieve. Planning uses the gets shortest path function which uses the A* algorithm for computing ideal paths. The A* algorithm maintains the following flow:

1. Initialization of a priority queue keyed by individual nodes' current cost and heuristic of euclidean distance.
2. Traverse through nodes until goal is reached
    - 2.1. Unvisited node u with smallest priority gets popped form the queue
    - 2.2. Selected node gets marked as visited
    - 2.3. Traverse each neighbor node that is not yet visited
        - 2.3.1. Calculate costs

$u = (v[0] + dx, v[1] + dy)$

2.3.2. If cost is less, append the current node:

$u, np.sqrt(dx ** 2 + dy ** 2)$

2.4. Cost is recorded and shortest path is yielded. [1]

In this implementation the planning is committed on a mapping configuration space translated to a map of the robot's world coordinate system.

Controllers and Behavior Trees

This implementation features an arm control class in the ***manipulation*** interface. This is where Tiago's arms and fingers are initialized. Arm control maintains a boolean operator to track if the joints being set in the pose dictionary are finished moving. If the current pose matches the desired pose configurations it returns success state so the robot can smoothly transition between pose states.

The manipulator also features two classes, GetObject and ReleaseObject, which are being used to retrieve that first jar in the controller. GetObject uses force feedback from the robot's grippers to check if the robot has successfully gripped an object. When the robot is ready to drop the object it would use ReleaseObject to open its grippers and drop the object.

The manipulator has one more class which is not being used in this implementation called PrepareGripper. The intended functionality is to use the camera to get the object's position. It uses recognition colors and appends them to a list to match the colors the robot is seeing to the color of the object it wishes to acquire. It takes in distance and angle variables:

$$dist = normalize(objPos_{1,2}) - 0.75$$

$$angle = arctan2(objPos_{2}, objPos_{1}) - 0.05$$

with 0.75 and 0.05 being offset values to determine distance and angle from jar.

If the absolute value of the distance and angle is less than the defined threshold, then the robot would adjust its arm and body to the object. This functionality is unfortunately not operational in my implementation which will be further explored in the discussion.

The ***controller*** defines a dictionary of poses for the robot to use. This includes a

default position the robot should be during the driving navigation process. Additionally it includes retract, grabbing, releasing, gripOpen, and gripClose for the robot to transition to in between states. The controller uses a blackboard to write these poses onto as well as key aspects such as the camera, timestep, robot, and waypoints. It then uses a behavior tree to log all necessary steps to pick up a jar as follows:

1. Start with two simple moves to plan to navigate to first two waypoints. Set the robot in its default pose position.
2. Plan navigation to approach counter to transition into the jar pick up stage.
3. Jar pick up stage:
    a. opens grippers
    b. changes pose to grabbing
    c. retrieves the jar with GetObject
    d. steps back to default position
    e. continues navigation by planning to fourth waypoint
4. Robot arrives at table, changes pose to releasing, and drop the jar on the table using ReleaseObject.

This is incremented using timesteps where at each step the tree ticks once. Once the tree receives that all moves have been accomplished through success dialoguing, the controller terminates and the robot stops.

Results

The results are displayed in the Webots default kitchen environment using a Tiago robot. The Tiago robot has a camera in the camera slot in addition to a GPS, compass, and lidar (Hokuyo URG-04LX-UG01).

The robot drives to the counter, picks up a jar, backs up, and places it on the kitchen table. Each step is documented with

output statements as follows:

```
Reached 0 35
Reached 1 35
Reached 2 35
Reached 3 35
Reached 4 35
Reached 5 35
Reached 6 35
Reached 7 35
Reached 8 35
Reached 9 35
Reached 10 35
Reached 11 35
```

for reaching waypoints,

```
Robo is attempting an object pick up.
```

for grabbing an object,

```
Robo is attempting an object pick up.
Robo successfully gripped object.
Reached 0 2
Reached 1 2
Robo released the object.
Jar dropped, controller finished.
INFO: 'BT_mapping_navigation' controller exited successfully.
```

and for successfully gripping and releasing the object. Termination is logged when controller is finished. This can also be visualized with the robot scenes below:

robot in initial stage of viewing jar,

robot grabs the jar,

robot moves to table with jar,

jar is placed on table.

Discussion

Although the robot clearly and completely executes the desired

basic implementation functionality it is important to note some special considerations. First, it is important to note that this implementation does not achieve any bonus requirements. There is no reactive layer, IK solution, or nonprehensile manipulation featured in this project.

Additionally, the robot is not using perception or the PrepareGripper functionality. This was the limiting factor in developing it only to the basic implementation. If the thresholding was better adjusted to the project, then the robot would be better equipped to grab all three jars instead of just one.

Overall, there are several achievements this implementation reached such as reliable path planning and jar picking with a clean behavior tree. Future considerations will address the limitations as aforementioned by integrating a live PrepareGripper, reactive obstacle avoidance, and inverse kinematics.

References

[1] Chapter 13, Correll, Nikolaus, Bradley Hayes, Christoffer Heckman, and Alessandro Roncone. *Introduction to autonomous robots: mechanisms, sensors, actuators, and algorithms*. MIT Press, 2022