BOSTON UNIVERSITY

COLLEGE OF ENGINEERING

Thesis

MULTI-AGENT MOBILE MANIPULATION METHODS

by

ROY XING

Submitted in partial fulfillment of the

requirements for the degree of

Bachelor of Science in Electrical Engineering

2023

Approved by

First Reader

John Baillieul, PhD
Distinguished Professor of Mechanical Engineering
Distinguished Professor of Systems Engineering
Distinguished Professor of Electrical & Computer Engineering

Second Reader

Roberto Tron, PhD
Assistant Professor of Mechanical Engineering
Assistant Professor of Systems Engineering

Third Reader

Tali Moreshet, PhD
ECE Associate Chair for Undergraduate Programs
Senior Lecturer
Research Assistant Professor

Fourth Reader

Alan Pisano, PhD
Associate Professor of the Practice of Electrical & Computer Engineering

ACKNOWLEDGMENTS

This dissertation is dedicated to my mentors from Boston University, MassRobotics, and Agility. Thank you for the guidance and help that lead me to where I am today.

Roy Xing

Spring 2023

Boston, MA

MULTI-AGENT MOBILE MANIPULATION METHODS

ROY XING

Boston University, College of Engineering, 2023

Major Professor: John Bailleul, Distinguished Professor ME, SE, ECE

ABSTRACT

Mobile manipulation is a relatively new term in robotics, being first coined in the 1990's—as compared to the fields that comprise it: *robot manipulation* and *autonomous mobile robots* that emerged in the 1950's. As such, there are still many avenues to explore about the full functionality of robots with mobile manipulation capabilities. Aspects of promising research paths include integrating visual sensing in the motion planning and execution of mobile manipulation as well as how to fully exploit the additional degrees of freedom (DOF) that a mobile base brings to a robot arm. Specifically, a largely unexplored research direction is that of multi-robot mobile manipulation in which different mobile agents will plan together to complete a manipulation task that would be challenging or impossible for a single agent. Formulation of principles of decentralized multi-robot cooperative motion planning and control is needed. As research in robotics continues to advance enabling robots to function in the real world, it is important that robots be given the ability to work with other robots or humans to accomplish a task. This thesis explores foundational principles of multi-agent mobile manipulation. The mobile manipulators under consideration exhibit kinematic redundancy, meaning they are capable of more degrees-of-freedom of movement than the minimum required to operate in the task space. In what follows, we explore approaches to resolving

kinematic redundancy so as to create consistently repeatable behaviors and predictable motions. For tasks in which multiple agents cooperate, principles of dynamic task partitioning are proposed. Empirical validation of the work is carried out using two sophisticated platforms consisting of Kinova Gen 3 Lite manipulator arms mounted of Clearpath Dingo-O omni-directional mobile bases. Software and engineering challenges are described.

CONTENTS

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

DOF	Degree Of Freedom
MPC	Model Predictive Control
SLQ .	Sequential Linear Quadratic
SLAM	Simultaneous Localization And Mapping
HOT	Higher Order Terms
FK .	Forward Kinematics
IK . .	Inverse Kinematics

CHAPTER 1

Introduction

Mobile manipulation is an ever-growing and important field as robots move out of the research lab and into real world applications. The goal of this thesis is to contribute to the current methods and implementations of mobile manipulator control. For some background, mobile manipulation refers to a type of robot in which a mobile platform is combined with a robot arm. The most common control approach to mobile manipulators is that of two separate control loops. One control loop is for the mobile base whilst the other is for the robot arm. Typically, the base will perform and execute path planning to a goal point. The goal point will be the basis for a new workspace for the robot arm. Once the arm is moved to the appropriate space, trajectory (motion) planning is done for the manipulation task. This approach to mobile manipulation is straightforward in implementation and theory, but does not take advantage of the system's total degrees of freedom.

This thesis details the theory, methodology, and implementation of a whole body mobile manipulation control with potential for multi-agent execution of tasks. Multi-agent refers to more than one robot collaborating to complete a task. Most research in multi-agent robots typically either deal with low numbers of degrees-of-freedom (DOF) mobile robots or stationary arms. However, here we explore configurations with high DOF multi-agent mobile manipulation capabilities and explore possible future approaches.

1.1 RESEARCH CONCEPTS AND DEVELOPMENT

1.1.1 Research Statement

The proposed research direction will be towards that of uniting the separated worlds of control of the mobile platform and robotic manipulator across multiple robotic agents. By combining the isolated control loops of the arm and base, the resulting coordination between the arm and base will lead to the potential of behaviors that are able to react and adapt to a dynamic environment. This added control authority and behavior of a single robot unit will be vital in coordinating multiple mobile manipulators to accomplish a task. Numerous aspects of the problem formulation and analysis are ripe for exploring novel approaches. Starting from the ground up, it is important to address how the mobile manipulator is formulated. Current approaches will typically make the distinction between the arm and base with their own separate dynamics and kinematics, which leads to the segmented end behaviors. Thus, this research direction will tackle the formulation of a mobile manipulator as a high DOF system, specifically that of an kinematically redundant arm. The mobile platform can be considered as a holonomic base joint to the robotic arm, capturing the wide workspace range that mobile platforms boast. Finally, the question of how multiple agents would infer and work together to complete a task is an open one.

Continuing from the different formulation of a mobile manipulator, it becomes important to discuss the planning approach for the system. Here it is appropriate to define the differences between the types of planning typically used in this space. Path planning is an approach in which the robot's motion is described geometrically[4]. Motion planning is defined by describing the robot's motion

as a function of time t whilst considering the system's velocity, acceleration, and kinematics. Sometimes trajectory planning is used interchangeably with motion planning, however trajectory planning typically carries the notion that the time evolution of the system's velocity and acceleration are considered [4].

Most mobile manipulator approaches would involve two isolated parts. The first part involves path planning for the mobile base to get the manipulator as close as possible to the target object. The second part comprises of trajectory planning for the robotic arm to finish the manipulation task with little to no consideration of the mobile base. Due to past computational power costs, methods in which the whole arm and base system are considered in the same time loop were not actively explored. However, given recent advancements in computing power it is now an appropriate time to consider these unexplored approaches to mobile manipulation. Specifically, a promising route is to consider the mobile manipulator as a single high DOF device so that the system's kinematic redundancy can be fully utilized to plan complex motions within a sensed dynamic environment.

1.1.2 Research Objectives, Development, and Deliverables

- A mathematical formulation of the mobile manipulation as a singular high DOF system.
- Algorithms for trajectory planning to achieve complex motions by exploiting the redundant kinematics.
- Motion concepts based on reduced order idealized models.
- Implement the previously completed algorithms into a physical simulation of a real robot system. Specifically, a Kinova Gen3 arm mounted on a Clearpath

Robotics™ Dingo-O mobile platform.

- Perform sim-to-real transfer of the controls algorithm from the physics simulation to a real mobile manipulator robot to test and verify the performance of the algorithms.

- A beginner-friendly git repository in python3 that is well documented and ready for sharing to enable other researchers to quickly extend this research.

1.2 PAST RESEARCH

Cefalo, Orilo, and Vendittelli created a control-based motion planner that operates in the constrained task-space of the robot with an extended time dimension. Their method generates samples by first selecting a task space point from the desired path of the task and computing the configuration using inverse kinematics (IK). This was improved by utilizing a Jacobian-based equation of motion with motion primitives for trajectory generation that still maintained continuous and constrained task execution. This approach can operate within a dynamic environment given that the moving object(s)'s trajectories are known and or measured precisely [5].

Brock and Kavraki calculated the free space representation of the robot as a swept volume with a wavefront expansion algorithm. Then a local minima free potential field function was implemented to find an optimal path through the tunnel. This method boasted real-time performance and minimum safety distances from obstacles in the environment at the cost of mathematical completeness. They tested their algorithm and validated it on a free-floating mobile manipulator representation. Due to their formulation of the robot, this approach is applicable for

holonomic wheeled mobile manipulators [6], such as the Dingo-O.

A particular approach named Sequential Expanded Lagrangian Homotopy (SELH) is proposed in [7]. It functions by sequentially calculating the configurations for a given path. The algorithm calculated the configuration for each point in the discretized trajectory. This calculation was achieved by optimizing the manipulability index using SELH in which the algorithm utilized the information from the previous calculated configuration in the next configuration calculation. This method resulted in collision avoidance, soft constraints on the task, and dynamic obstacle avoidance. Unfortunately, the quality of the planned trajectory depends on the starting configuration which must be done offline for a quality estimation.

Su and Xie proposed a concept, coined the *Representation Space (RS)*, in [8] which is defined as a space spanned by the attributes of the robot system and the given task. Configuration space and task space can be considered as subsets of RS where a link of the manipulator can be one of the variables considered in RS. The authors explicitly calculated RS through discretizing each dimension to check for feasibility. The resulting motion plan is then constructed by carrying out the A* algorithm through all the valid points in RS [8].

As for the topic of multi-robot manipulation, most past research involves groups of three or more robots that do not utilize any explicit end-effectors, as seen in [10] and [11]. In past system formulations, the robots are treated as agents that are only able to move in their environment and not be able to manipulate any objects as seen in robot arm applications. Hence, there are open opportunities to expand upon robots' abilities for multi-agent manipulation by fully utilizing each mobile manipulator's functionality.

1.3 ENGINEERING REQUIREMENTS

The following subsections detail and categorize the specific requirements and constraints that must be met for the research project to be a success. We created the specifications for the robots' designs as listed below, separated into mechanical and electrical bullet points. Modifications to the original design concept as seen in the finalized real robots are also discussed. The finalized versions of the four total robots are rendered in RViz as seen in figures 1.1 and 1.2.

1.3.1 Mechanical

- Workspace of the robot arm must not be severely constrained by the configuration of the robot base and sensors. Specifically, the arm must have a free-space of operation of at least 180 degrees horizontally parallel to the mobile platform without any vertical space obstructions.

- The sensors of the robot, depth cameras and LiDAR, must have an unobstructed field of vision from their respective mounting points. This constraint led to the loss of a tower structure for the arm, which means that the arm no longer can operate in the same workspace as standing humans. Instead of a tower, the robots now have stand-off platforms for the arm and sensors which can be easily reconfigured for specific experiments and applications.

- At least one of the depth cameras need to be able to change its field of view. This leads to the need of an actuated gimbal system on top of which the sensor is mounted. The resulting actuated gimbal system is also mounted on the modular stand-off platforms.

1.3.2 Electrical

- The lithium ion batteries of the robot must be able to supply power to both the mobile base and the arm simultaneously. Ideally, the robot should have an operating time of at least two hours with all actuated DOFs engaged and at least five hours when idle.
- The robot must also have wireless and bluetooth modules capable of supporting up to 200 meters from the active development laptop, which the robot must communicate with for operation.

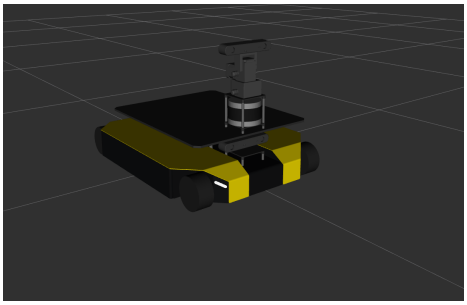
1.3.3 Whole Body Mobile Manipulation Control

1.3.3.1 ROS

The robot must be built with the framework of ROS (Robot Operating System) to ensure modularity and compatibility with numerous other robotic platforms. This enables easy adoption for other researchers in implementing and adopting the controls algorithm code.

1.3.3.2 Gazebo

A physics simulation model of the robot must first be verified within a simulator, such as Gazebo. Verification is defined as accurate detection of collision behaviors of the various parts of the robot (arm links, sensor mounts, etc.) along with accurate modeling of the motion behavior of the real mobile base in terms of velocity and acceleration with respect to the ground properties (i.e. friction).



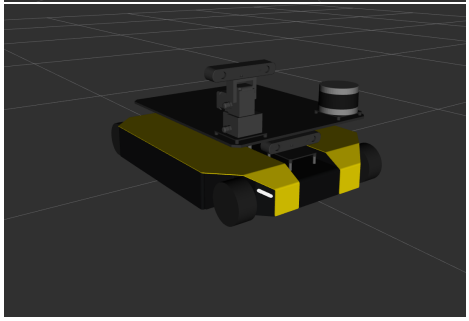


Figure 1.1: Two dingo with arm configurations and sensors (labeled d1 and d2).

Figure 1.2: Two dingo sensors only configurations (labeled d3 and d4).

1.3.3.3 Trajectory Planning

The control loop of the robot must be able to perform in real-time on the real robot with a termination state of the planned trajectory. The robot must be fully autonomous in both recognition of its target and the motion that will lead it to said target.

1.3.3.4 Coordination & Robustness

Each robot within the environment must be able to coordinate together to accomplish a manipulation task. In addition, each robot must also be able to account and adjust accordingly for changes in their own or their counterpart's planned trajectory.

CHAPTER 2
Whole Body Control & Multi-Agents

2.1 INTRODUCTION

In this chapter we study the problem of trajectory planning with the mobile manipulator modeled as a single high DOF system. The system is modeled as a large redundant kinematic chain [9]. As seen in [15], this formulation of the mobile manipulator problem has been analyzed in the past, but implementation still remains a challenge on real robots. Within this chapter we first discuss the components that make up a mobile manipulator and system modeling. Then we will cover the whole body control from forward and inverse kinematics all the way to the model predictive control framework. Finally, we will discuss the extension of controlling multiple mobile manipulator robots with the established controller and what extensions need to be made to do so.

2.2 WHOLE BODY CONTROL

We will start with the overarching concepts that lead to the MPC formulation and approaches for a multi-agent approach. The learning of motion primitives and future work will also be touched upon, but further expanded upon in the final chapter of conclusions.

2.2.1 Control of Kinematic Chains

Here we start by defining a kinematic chain, which is a sequential assembly of rigid bodies connected by joints. Kinematic chains serve as the mathematical model for mechanical systems by providing insight into the system's constrained and or

desired motion. As such, this concept will serve as the foundation on which we build on. The joints of the system are modeled by lower pairs (also referred to as kinematic pairs), which are defined as couplings of rigid bodies such that the bodies are in contact on two dimensional surfaces. Out of the eight possible lower joints, our robot will be modeled out of revolute, cylindrical, and planar joints.

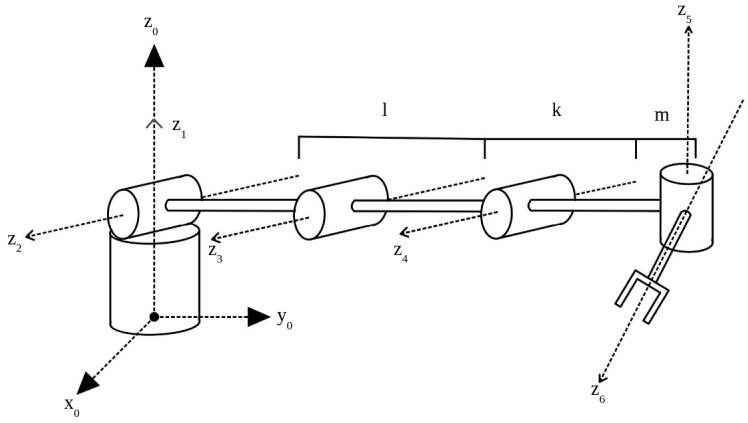


Figure 2.1: Elbow manipulator kinematic representation.

Denavit-Hartenberg (D-H) Table			
i	a_i	α_i	d_i
1	0	90	0
2	l	0	0
3	k	0	0
4	m	-90	0
5	0	90	0
6	0	0	0

Table 2.1: D-H Table for the elbow manipulator.

2.2.1.1 Forward Kinematics

Forward kinematics (FK) is the use of kinematic equations to compute the resulting position an end-effector of an arm due to specified joint angle values. The simplest

formulation of the FK problem is written as follows,

$$f(\theta) = x \tag{2.1}$$

Where θ is the joint angle vector and x is the resulting position and orientation position of the end-effector.

The convention for defining joint and link (i.e. linkages of the robot arm) matrices were created by Jacques Denavit and Richard Hartenberg in 1955 [16]. Let us denote the link matrices as $\mathbf{X}$ and joint matrices as $\mathbf{Z}$ and the end-link transformation in respect to the base-link as $\mathbf{T}$. We will also denote $\mathbf{Trans}_{q_i}$ and $\mathbf{Rot}_{q_i}$ as the translation and rotation matrix with q denoting the axis of motion and i as the i$^{\text{th}}$ joint or link and N as the total number of linkages. The end coordinate transformation of the end-effector can then be expressed as follows.

$$\mathbf{T} = \mathbf{Z}_1\mathbf{X}_1\mathbf{Z}_2\mathbf{X}_2...\mathbf{X}_{N-1}\mathbf{Z}_N \tag{2.2}$$

$$^{\text{i-1}}\mathbf{T}_i = \mathbf{Z}_i\mathbf{X}_i = \mathbf{Trans}_{z_i}(d_i)\mathbf{Rot}_{z_i}(\theta_i)\mathbf{Trans}_{x_i}(a_{i,i+1})\mathbf{Rot}_{x_i}(\alpha_{i,i+1}) \tag{2.3}$$

The variables of $a_i, \alpha_i, d_i, \theta_i$ are known as Denavit-Hartenberg variables and are referred to as follows. a_i is interpreted as the unique shortest line segment between coordinate frames. α_i is the twist angle for the following link. d_i is the link offset. θ_i is the joint angle. Explicit computation of the transformation matrices yields the

Denavit-Hartenberg matrix which takes the following form.

$$^{\text{i-1}}\mathbf{T}_i = \begin{bmatrix} cos\theta_i & -sin\theta_i cos\alpha_{i,i+1} & sin\theta_i cos\alpha_{i,i+1} & a_{i,i+1}cos\theta_i \\ sin\theta_i & cos\theta_i cos\alpha_{i,i+1} & -cos\theta_i cos\alpha_{i,i+1} & a_{i,i+1}sin\theta_i \\ 0 & sin\alpha_{i,i+1} & cos\alpha_{i,i+1} & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.4}$$

2.2.1.2 Inverse Kinematics

As suggested by the name, inverse kinematics (IK) is the opposite formulation of the forward kinematics problem. An interpretation of IK is that of: "If given an endpoint for the end-effector, what is the configuration (if such a configuration exists) for the joint angles to achieve said endpoint for the end-effector?" The IK problem in its simplest form may be derived from the FK problem equation 2.1 as such,

$$\theta = f^{-1}(x) \tag{2.5}$$

Where x is the given position and orientation vector and θ is the vector of joint angles for the kinematic chain. The original generalized IK solution by Whitney in 1969 takes a similar form but focused on the joint space's and configuration space's velocities, $\dot{\theta}$ & $\dot{x}$ [32]. There are two main approaches to solving the inverse kinematics problem, analytical and iterative. Analytical (algebraic) solutions are when one inverts the direct kinematics equations and then enumerate all the possible solution branches. However, analytical solutions are limited to systems in which the number of constraints are the same as the number of DOFs of the robot. Hence, we will focus on iterative numerical approach. We will cover an overview of the IK problem for redundant kinematic chains as seen in [9], detailing the extended

Jacobian technique created by Baillieul (1985).

Kinematically redundant systems are defined by the dimensions of the state position and orientation vector x and the the joint configurations vector θ. A kinematic chain is redundant when the following holds:

$$dim(\theta) > dim(x) \tag{2.6}$$

This can be interpreted as for any given x there are generally infinitely many solutions θ for each x. For a solution of θ it is useful to think of it as a solution path to the IK problem. It is possible that similar θ solutions may have similar entries and thus it is important to execute a single solution path. Another important note to define is that of kinematic singularities. Kinematically singular configurations are defined by points in the joint space of θ where the Jacobian, $\frac{\delta f}{\delta \theta}$, is not full rank. A physical interpretation of this phenomenon is that kinematic singularities are points in which the relationship between the end-effector velocity and joint angle velocities fall apart, such as when huge (or even infinite) velocities are commanded for the joints to maintain a finite end-effector velocity. Unfortunately, kinematic singularities are unavoidable even with different coordinate representations of a kinematic chain. The psuedo-inverse method for solving the IK problem cannot generally generate join trajectories that avoided singularities as proved in [34].

Suppose the end-effector frame is expressed by a state vector x which is governed by the forward kinematics as seen in equation 2.1, a nonlinear vector mapping from the n joint coordinates to the m end-effector coordinates, $\mathbb{R}^n \rightarrow \mathbb{R}^m$. Assume that the function f is differentiable. Next, let the desired (end) coordinates for the end-effector to be x_d. The general form of the extended Jacobian method

can be expressed as follows, with a kinematic cost function $g(\theta)$:

$$\Delta\theta = \alpha(J^{ext.}(\theta))^{-1}\Delta x^{ext.} \tag{2.7}$$

Where α is a scalar factor, $J^{ext.}(\theta)$ is the extended Jacobian, and $\Delta x^{ext.}$ is an incremental step to the desired end-effector position and orientation. An example of a kinematic cost function could be:

$$g(\theta) = x_d - f(\theta_d) = 0 \tag{2.8}$$

Other cost functions exist, such as one that optimizes for torque requirements. The main principle for the extended Jacobian method is the optimization in the null-space of the Jacobian, which can be interpreted as ensuring that velocities are applied within the operational space, using the cost function $g(\theta)$.

For more insight into this method we will derive the extended Jacobian as follows. As stated above, the forward kinematics $x = f(\theta)$ is a mapping from n-dimensional joint space to m-dimensional Cartesian space, $\mathbb{R}^n \rightarrow \mathbb{R}^m$. The singular value decomposition (SVD) of the Jacobian of this mapping is:

$$J(\theta) = U\Sigma V^T \tag{2.9}$$

where the rows of V, whose corresponding entry in the diagonal matrix Σ is zero, are the vectors that span the null space of the Jacobian. Then there must be at least $n - m$, assuming $n \geq m$, of these spanning vectors. Let us call these vectors η_i where $i \in [1, (n-m)]$. The objective of the extended Jacobian technique is augment the rank deficient Jacobian such that it satisfies the requirements to be an invertible

matrix. This is where the cost function $g(\theta)$ comes in, the cost function is used to be minimized with respect to θ within the null-space. This minimization process must yield the following expression:

$$\frac{\delta g}{\delta \theta}(\theta) = 0 \tag{2.10}$$

This gradient is then projected onto the null-space basis vectors since we are mainly focussed on zeroing the null-space gradient:

$$\Xi_i = \frac{\delta g}{\delta \theta}(\theta)\eta_i \tag{2.11}$$

If all possible Ξ_i for $i \in [1, (n-m)]$ are equal to zero, then the cost function is minimized in null-space. From here, the set of equations can be obtained, which are to be completed by the IK solution,

$$\begin{bmatrix} f(\theta) \\ \Xi_i \\ \dots \\ \Xi_{(n-m)} \end{bmatrix} = \begin{bmatrix} x \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{2.12}$$

Finally, for an incremental step Δx, the system can be linearized,

$$\begin{bmatrix} J(\theta) \\ \frac{\delta \Xi_i}{\delta \theta} \\ \dots \\ \frac{\delta \Xi_{(n-m)}}{\delta \theta} \end{bmatrix} \Delta\theta = \begin{bmatrix} \Delta x \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{2.13}$$

which can be rewritten with the scalar factor α if needed for the implementation

on the non-ideal physical system:

$$\alpha J^{ext.} \Delta\theta = \Delta x^{ext.} \tag{2.14}$$

The unique solution of these equations to solve the IK problem is thus $\Delta\theta = \alpha(J^{ext.}(\theta))^{-1}\Delta x^{ext.}$ where the inverse may be the Moore-Penrose inverse (also known as the psuedo-inverse). The extended Jacobian method for solving the IK problem has a long and rich history due to its numerous advantages. This technique is computationally fast, numerically robust, and its explicit optimization criterion aspect enables control over possible arm configurations.

2.2.2 System Description

The specific implementation of a whole body controller of a mobile manipulator robot will be described in two parts. One will be the actual hardware of the platform used to verify and test the controls approach in the real world and the other part will specify the controls approach itself.

2.2.2.1 Hardware

It should be noted that the robot base is holomonic, which means that the controllable degrees of freedom is the same as the total degrees of freedom of the robot. This property allows the robot to move freely in any direction. On the real hardware this is achieved through four omniwheels (mecanum wheel) instead of traditional tires. This attribute of the base allows for a non-constrained base capable of instantaneous motion in any direction in its planar task space. The robot arm is a fully actuated 6 DOF (including the end-effector gripper) system complete with absolute encoders for position. In addition, the robot features an actuated 2 DOF

gimbal for one of its depth cameras (the other is static) and a 360-degree lidar. The actuated gimbal does not contribute to the workspace of the arm nor the traversal of the base, but is solely for the purpose of augmenting the field of vision that one of the depth cameras may perceive. In terms of background parts, the robot also sports a total of 12 individual LiPo battery packs that allow for hot-swapping to enable continuous run-time when testing the robot indoors. The robot chassis is also completely made out of casted aluminum and milled steel to house its NVIDIA Jetson computer, Flir motor controller, power regulator, main motor driver, and MCU as seen in the following block diagram.

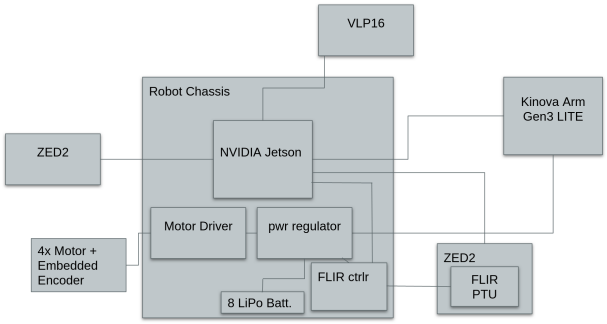


Figure 2.2: Block diagram of the physical robot hardware.

2.2.2.2 System Model

The main idea to consider when discussing the controls is the concept of interpreting the robot as a single high DOF system instead of two coupled distinct systems. A common formulation of such a system is seen in [14] and will be discussed and referenced here.

First, we want to define some of the physical aspects of the environment and robot where the robot links state is {q} and the arm joint state is $\{\theta\}$.

Let the following be true:

- {s} be fixed space frame
- {b} be the mobile base chassis frame
- {0} be the base frame of the arm
- {e} be the robot arm's end-effector frame.

Now we can formulate the configuration of the robot's end-effector in relation to the defined fixed space frame as seen in the following equation. $SE(3)$ is the special Euclidean group in terms of which we may describe the six degrees of freedom motions involving positions (3) and orientations (3) with respect to any of these frames. We may also use elements of $SE(3)$ to represent coordinate transformation between frames. Thus, in an equation like

$$X(q,\theta) = T_{se}(q,\theta) = T_{sb}(q)T_{b0}T_{0e}(0) \in SE(3) \tag{2.15}$$

the matrices can be interpreted as:

- T_{0e} is the forward kinematics of the robot arm
- T_{b0} is the fixed offset of arm base frame from base chassis
- $q = (\phi, x, y)$ the planar configuration of the base
- $$\begin{pmatrix} cos(\phi) & -sin(\phi) & 0 & x \\ sin(\phi) & cos(\phi) & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{pmatrix} = T_{sb}(q)$$
- z is the height of frame {b} off of the floor {s}

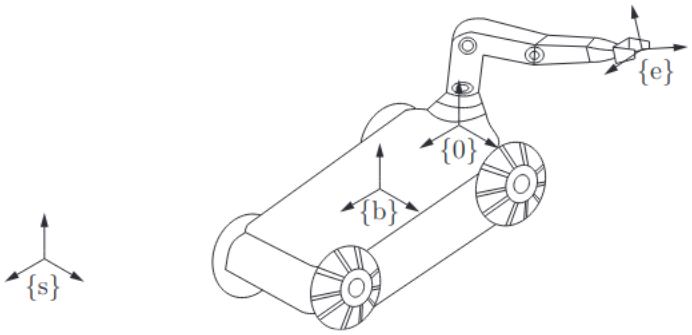


Figure 2.3: Visualization of all the frames related to a mobile manipulator. Taken from [14]

For kinematic control of the mobile manipulator we must define the end-effector twist matrix in respect to the end-effector frame as described above. Note that these formulations are generalized for a mobile manipulator with m number of wheels and n number of joints in the robot arm. The following 6xm Jacobians must first be defined:

- $J_{base}(\theta)$ maps wheel velocities to end-effector velocity
- $J_{arm}(\theta)$ is the arm's body Jacobian.

The following equations show the definition of the end-effector twist V_e and the Jacobian $J_{base}(\theta)$ where F_6 is the transformation from wheel velocities to the twist of the chassis, which is defined as $V_{b6} = Fu$. For more details please refer to [14].

$$V_e = \begin{bmatrix} J_{base}(\theta) & J_{arm}(\theta) \end{bmatrix} \begin{bmatrix} u \\ \dot{\theta} \end{bmatrix} \tag{2.16}$$

$$J_{base}(theta) = [Ad_{T_{0e}^{-1} T_{b0}^{-1}}] * F_6 \tag{2.17}$$

2.2.3 Motion Planning

Motion planning, as the name implies, is generating motions for a robot to move from a starting to end point or pose. Constraints are also taken into account, such as collision avoidance, velocity and acceleration limits, joint limits, and more. Some literature may also reference motion planning as trajectory planning, though in practice trajectory planning tends to have a smaller scope of constraints. To accomplish whole body motion control we employ a model predictive approach which is outlined in the next section. Here we will give an overview of the methods and foundations for motion planning.

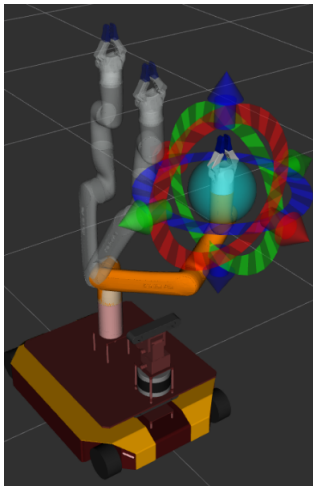


Figure 2.4: Motion planning done by MoveIt! in RViz.

First we define the concept of configuration space (C-space). Any configuration of a robot's linkages and joint angles can be represented in C-space, such that every point in C-space corresponds to a specific robot configuration. Assuming that the linkages of a robot does not change length we can then represent an n-joint robot's configuration with just its joint angles as a list:

$$q = (\theta_0, \theta_1, ..., \theta_n) \tag{2.18}$$

where C-space $\in \mathbb{R}^n$. Of all the possible configurations of the robot we consider the ones that do not violate a certain constraint, such as penetrating (or colliding) with an obstacle or joint angle limits. These desired configurations will fall under free C-space.

For robots that possess second-order dynamics, which includes robot arms, then the state of the robot needs more than its joint configurations to define itself. By including the velocity of the robot v then a defining state vector can be written,

$$x = (q, v) \tag{2.19}$$

where $v = \dot{q}$ and $q(x)$ refers to a corresponding configuration for state x. The simplest form of the equations of motion for a robot can be expressed with the following:

$$\dot{x} = f(x, u) \tag{2.20}$$

where u is the control input, which can be velocities or forces depending on the application. Some also prefer to express the equation of motion in an integral form with T being the final time and t_0 is the start time

$$x(T) = x(t_0) + \int_{t_0}^{T} f(x(t), u(t))dt \tag{2.21}$$

With all the pre-requisite definitions established we can formulate the generic motion planning problem. If given an initial state $x_0 = x(t_0)$ and a desired final state $x_d = x(T)$, we need to find a set of controls u that reaches the desired state within time $t \in [t_0, T]$ where $q(x(t)) \in$ free C-space.

2.2.3.1 *Model Predictive Control*

MPC is considered an optimal control method where the computed control action minimizes a cost function for a (constrained) dynamical system over a finite time horizon that is receding. The steps of MPC is as follows. At each time step, the MPC controller receives a measured or estimated state of the system. The controller then computes the sequence of control actions which minimizes a given cost function over a set time horizon for a constrained optimization problem. The optimization problem is dependent on an internal model of the dynamic system and depends on the current (measured) state of the system. The MPC controller finally applies the first control inputs to the system and then the process repeats for the next time step.

2.2.3.2 *Sequential Linear Quadratic*

In our purposes we use a Sequential Linear Quadratic MPC (SLQ-MPC) which is a type of MPC for nonlinear systems as seen and adapted from [17]. This work uses a continuous-time formulation of SLQ-MPC which is implemented by the open-source C++ toolbox Optimal Control for Switched Systems (OCS2) [18]. The SLQ differential dynamic programming algorithm is explained as follows from [19].

Consider a generic dynamic system expressed by a set of nonlinear differential equations

$$\dot{x}(t) = f(x(t), u(t), t) \tag{2.22}$$

where $x \in \mathbb{R}^n$ is the state variable vector, $u \in \mathbb{R}^n$ is the input variables, and $f(x, u, t) \in \mathbb{R}^n$ is a continuously differentiable system. The optimal control prob-

lem for a generic nonlinear differential system is setup as

$$\begin{cases} \min\limits_{u} \quad \Phi(x(t_f)) + \int_{t_o}^{t_f} L(x(t), u(t), t)dt \\ \text{s.t.} \quad \dot{x}(t) = f(x(t), u(t), t) \quad \forall t \in [t_0, t_f] \quad \text{and} \quad x(t_0) = x_0 \end{cases} \tag{2.23}$$

where $L(x(t), u(t), t)$ is the stage cost and $\Phi(x(t_f))$ is the terminal cost. Optimal trajectories are found by iteratively solving local approximations, which requires first-order derivatives of the dynamics and approximations of the objective function by quadrization around nominal trajectories $\hat{x}(t)$ and $\hat{u}(t)$.

$$\delta\dot{x} \approx \mathbf{A}(t)\delta x + \mathbf{B}\delta u \tag{2.24}$$

$$\hat{\Phi} = \frac{1}{2}x^T\mathbf{Q}_f\delta x + q_f^T\delta x + q_f \tag{2.25}$$

$$\hat{L} \approx= \frac{1}{2}\delta x^T\mathbf{Q}(T)\delta x + \frac{1}{2}\delta u^T\mathbf{R}(t)\delta u + \delta u^T\mathbf{P}(t)\delta x + q^T(t)\delta x + r^T(t)\delta u + q(t) \tag{2.26}$$

The Pontryagin Maximum Principle comes into play here (particularly the minimum formulation) to solve for the approximate optimal control problem by providing necessary considtions for optimality [20]. The Hamiltonian function $\mathbf{H}$ can then be defined where $\lambda(t)$ is the time-varying Langrange multiplier vector with its elements being costates of the system.

$$\mathbf{H}(\delta x, \delta u, \lambda, t) := \hat{L} + \lambda^T(\mathbf{A}\delta x + \mathbf{B}\delta u) \tag{2.27}$$

By minimizing the Hamiltonian function, the optimal control input variable can be obtained.

$$\delta u^* = arg\min_{\delta u} \mathbf{H} \tag{2.28}$$

Finally, a differential Riccati equation $\mathbf{K}(t)$ can emerge by some mathematical manipulation as seen in [21]. By cranking the integration of the Riccati equation backwards in time, an affine control policy arises.

$$\delta u^*(t) = \delta u_{ff}(t) + \mathbf{K}(t)\delta x(t) \tag{2.29}$$

This control policy is then sent back into 2.22 to generate the next nominal trajectories for the next iteration. The overall algorithm for the SLQ-MPC is detailed below.

Algorithm 1 SLQ-MPC

Initialization
for $k = 0, 1, ...$ **do**
  Rollout system dynamics $x_k(t)$ for a given time horizon
  Approximate the objective function around nominal trajectories
  Linearize dynamics
  Solve **Riccati** equations
  Compute **control policy** $\delta u^*(t) = \delta u_{ff}(t) + \mathbf{K}(t)\delta x(t)$
end

Application and formulation of this algorithm to our robot is covered in Chapter 3 under *Section 3.3.9 Implementation*.

2.3 MULTI-AGENTS

So far we have considered the control and coordination of a single robot. However, for deployment of robot fleets in the real world robots need to be able to coordinate among each other. Specifically, the ability for robots to work together to achieve a task is an important behavior for mobile manipulation fleets to be useful. Similar to how humans cooperate to lift objects heavier than an individual is capable of lifting, mobile manipulators will benefit from such cooperation. In addition, for

collaborative robot arms (co-bots) maximum lifting payloads tend to be light due to the requirement of operating under non-lethal torques around humans. Thus, for safe human robot collaboration strategies are needed for cooperation.

Past multi-agent manipulation research has either focused on low DOF systems without arms or on mobile manipulation systems with an already grasped object [22]. Thus, here we discuss approaches for grasping a rigid body object by two mobile manipulators.

2.3.1 Configurations

We start by discussing one of the most simplest examples, picking up a cardboard box. With the dimensions are seen as below with length l, width w, and depth d. We assume that the dimensions of the object are known, either due to being

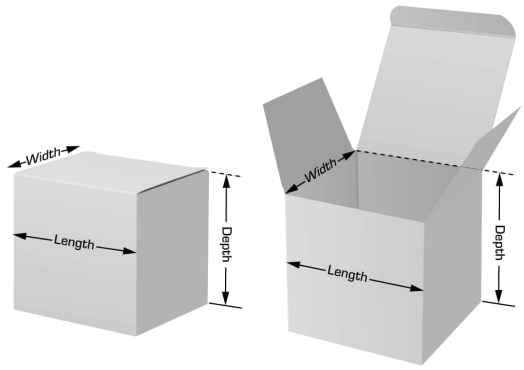


Figure 2.5: Generic box with dimension labels.

given or estimated through sensors. From there, the centroid of the object can be calculated. The gasp points then are determined through the relative distances of the centroid to the surfaces of the object. Ideally, corner folds of the object are ideal for two-fingered grasping. If the end-effector falls under the category of two-fingered grippers then the grasp points will be one corner of the object and the

corresponding furthest away graspable corner for the second robot, forming a line that crosses the centroid with some discrepancy allowed.

An alternative to finger grasping for manipulation is possible through the use of placing the end-effector along a side of the object, maximizing surface area contact. The second robot will then need to place its end-effector on the opposite end, once again crossing the centroid of the object, maximizing surface area contact. However, this method will also need to consider the forces the arm exerts on the object to avoid object deformation. As such, torque estimation is needed for the arms, which is possible through a simple equation dependent on torque $Torque = Kt * Current$ where Kt is the motor torque coefficient. The arm used in this research is capable of calculating estimated torque but this method is out of the scope of this thesis.

CHAPTER 3
Implementation Details

3.1 INTRODUCTION

In this section we go over the work and details of applying the previously discussed theory to our real robots. We begin with discussing the software frameworks that are essential to modern robotics development. Then we go over a development log, detailing issues and solutions encountered in practice. Finally, we cover the implementation of the high level theory of motion planning and control to the robots.

3.2 ROS AND GAZEBO

Robot Operating System (ROS) is an open-source framework designed for building, testing, and deploying robotic systems [25]. It provides a collection of open-source libraries and tools that enable roboticists to create complex robot applications in a streamlined fashion with standards and incompatibility. ROS is widely used within the robotics community, ranging from professional industry to research.

One of the central components of ROS is the physics simulator Gazebo. Gazebo is a multi-robot simulator that allows users to test and develop their robot software in a virtual environment. This simulator allows for rapid prototyping and testing of robot algorithms and applications. It provides an accurate physics engine, ode, (bullet, simbody, and dart are also supported) that simulates the behavior of robots and their interactions with the environment. Gazebo can simulate a wide range of sensors, such as cameras, LiDAR, and depth cameras, in addition to simulating the

body and actuators, allowing for a full stack simulation of robots.

ROS and Gazebo are often used together to develop robotic systems. Developers can create a virtual environment in Gazebo and simulate their robot's behavior. Due to ROS's platform agnostic structure, the same nodes used to control the virtual robot can directly be applied to the physical robot, allowing for direct sim-to-real transfer.

3.2.1 Robot Operating System

ROS is based on a distributed architecture, meaning that a robotic system consists of multiple independent nodes that communicate with one another over a network. This allows ROS to be platform agnostic and highly compatible with various robots, as long as there is an appropriate control node for the robot model.

At the core of ROS is the concept of a master node, ROS master, activated with the roscore command. This central node is responsible for managing the communication between nodes. Essentially, the master node acts as a domain name service (DNS) server for the nodes. The master node maintains a registry of all the nodes in the system and their respective topics where ros messages are published on. Nodes can register their topics with the master node, allowing other nodes to subscribe to or publish messages on those topics. A visualization of this system is shown in the next figure in the form of a ROS graph:

Publishers and subscribers are the two primary communication mechanisms in ROS. A publisher node publishes messages on a specific topic, while a subscriber node receives messages on that same topic. ROS messages are written in a simplified description language meant to describe data values. ROS provides a vast library of message types that can be used to communicate between nodes,

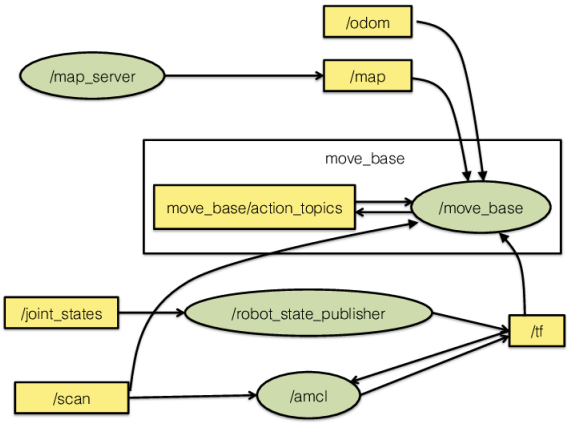


Figure 3.1: Generic ROS graph example generated by rqt_graph.

such as sensor readings, motor commands, and control messages. Publishers and subscribers can be located on different physical machines, as long as they are connected to the same network and are registered with the master node. This means that as long as a robot is able to communicate with a master node, then it has access to any topic and its respective messages across multiple platforms. When a publisher node sends a message on a topic, it is transmitted to the master node, which then forwards the message to all registered subscriber nodes for that topic. Subscriber nodes can receive multiple messages from multiple publishers, allowing for distributed processing and parallel computation.

Additionally, ROS includes a set of tools for debugging and visualizing messages, enabling developers to monitor and analyze the behavior of the system. This includes RViz (Robot Visualization) which is 3D visualization tool for a robot's state and sensors. One can think of RViz as what a robot believes its own state and sensor readings are.

In our discussion of ROS up til now has been in reference to ROS 1. In recent years there has been lots of development and advancement in ROS 1's successor

ROS 2 (the developers prefer people to not to refer to it as ROS2). ROS 2 is meant to build on the success of ROS 1 and further standardize the software framework. Some of the differences between ROS 1 and ROS 2 include its build system, middleware, and comptatible operating systems. ROS 1 utilizes a custom serialization format, transport protocol, and central discovery mechanism. ROS 2 does away with this and has an abstract middleware interface, which allows for better communication across networks. ROS 2 also no longer has a ROS master node, making it a decentralized system. Each ROS 2 node has the ability to discover other nodes and thus do not need a pre-requisite ROS master node to be run. Each robotic system will be independent of each other and also be able to start up, connect, and disconnect with each other. There are many other subtle differences not discussed in this thesis as it is out of scope of the abstract.

For the work detailed in this thesis we adhere to ROS 1 due to its stability and massive community support. In the near future most systems will be migrated to ROS 2.

3.2.2 Gazebo

Gazebo is able to simulate the dynamics of the robot and its interactions with the environment, including friction, gravity, and collisions. Gazebo's simulation environment is defined using a file format called URDF (Universal Robot Description Format), which describes the robot model, its sensors, and the properties of the environment. There are also other formats, such as SDF (Simulation Description Format) which has been gaining popularity due to being more streamlined than URDF. For the purpose of this thesis, we use URDF due to its more robust support and community.

Gazebo can be controlled using ROS, allowing for seamless integration with other ROS nodes. ROS nodes can publish commands to the simulated robot, receive sensor data, and visualize the simulation, which are all wrapped up in a large Gazebo node. This system allows for the relatively easy sim-to-real transfer as the Gazebo node simply needs to be swapped out for the real robot's controller nodes.

Gazebo also includes a plugin architecture, which allows developers to extend the simulator's functionality. Plugins can be used to add new sensor models, physics engines, and visualization tools.

3D dynamic simulators, such as Gazebo, are similar to game engines. In fact, advancements in game graphics and fidelity lead to the advent of robot camera simulations. Robot simulators in contrast to game engines focus and offer accurate real world physics simulations with a higher degree of fidelity.

3.2.2.1 Visualization

The current physical robots already have preliminary configuration models created in urdf format. Two possible placements of the arm and sensors upon the mobile platform are presented in the following figure 3.2 & figure 3.3.

Figure 3.2: Mobile manipulator configuration 1 rendered in RViz.

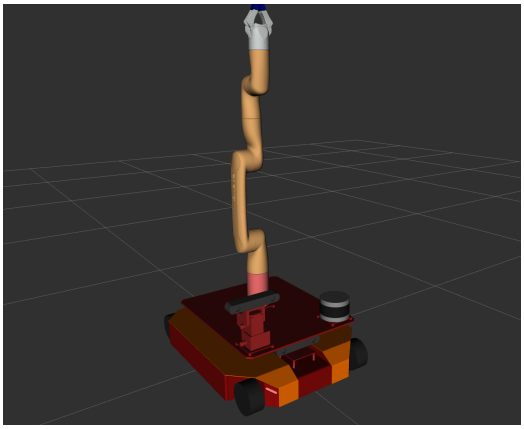


Figure 3.3: Mobile manipulator configuration 2 rendered in RViz.

It is important to note that these robot configurations are subject to change depending on physical workspace requirements, arm movement inertia, and various other factors that need to be tested. This also naturally applies to the real robot configuration as shown in figure 3.4. More configurations of the Dingo-Os in terms of sensors and arm placements can be found in the Appendix.

Figure 3.4: Mobile manipulator robot comprising of the Kinova Gen3 Lite arm and Clearpath Dingo-O holonomic mobile platform

3.3 TECHNICAL DEVELOPMENT

3.3.1 Initial Design Proposal of the Robots

The initial designs of the robots were first sketched up around one year ago in 2021. After numerous meetings with Clearpath Robotics, the first configurations of the arms and sensors were determined. Following meetings established the modularity of the sensors and actuators for future research with the robot fleet. It was also determined during this phase of the type of mobile base to use, which ended up being the Dingo-O due to its four wheel drive and omniwheel capabilities. The Kinova Gen3 LITE arm was chosen due to its lightweight body and compact form of not having a discrete control box as seen with other popular arms, such as the Universal Robots series. The ZED2 depth sensor was chosen over that of the ZED1 due to deprecation of the older version's support and libraries. The Velodyne VLP16 LiDAR was chosen due to its small form factor and its thorough documentation and support from its parent company. The actuated gimbal system for one of the two ZED2's is manufactured by FLIR and is meant for future experiments involving non-fixed view binocular vision. Finally, the batteries were to be hot-swappable lithium-ion as to ensure a long and continuous runtime through changing out battery units as needed. The batteries are lithium-ion with 8.5 Ah at 14.4V, so they should last around two hours and require eight hours of charge time.

3.3.2 Receiving Parts and Initial Setup

Unfortunately, due to the pandemic and supply chain shortage, the robots experienced numerous delays. It was not until autumn of 2022 that the shipments arrived

from Clearpath's offices in Canada. When the packages arrived at the university a survey of parts were first taken and then came the building of the robots. There are twelve individual battery packs that need to be installed and removed for charging on a separate dock. So hot swapping batteries seems to be feasible, but charging will be a bit tedious. The NVIDIA Jetson computer is Ubuntu 18.04 with ROS Melodic and thus there needs to be some caution taken when interfacting with the 20.04 ROS Noetic development laptop.

The HDMI output for the Jetson is also located underneath the metal covers and thus if we want to directly jack into the robot we will need to take apart the cover over and over again. Perhaps it will be worth the time to extend the HDMI female cable out through one of the wire holes for more convenient access. Unfortunately, the Kinova Lite 3 arm is missing some of its power cables and regulators and thus some time will have to be invested in either tracking them down or making them ourselves.

Finally, an initial bootup of the whole system needed to be performed. This involved booting up the robot and pairing a bluetooth controller to it to test out its drive train. It is important to note that it took about 1-2 minutes for initial startup and only 30 seconds or so for the bluetooth pairing of the controller. After that the system drove and handled fine. Though it is vital to note that the clearance on the bottom of the robot is quite low.

3.3.3 Testing Actuators

The booting up the arm on top of the robot did not have any issues and initiated its joints without any errors returned. When connecting the arm's static IP issues there were a misunderstanding in whether or not the address was already added to the

'/etc/sources' file or not in the NVIDIA Jetson. However, the robot automatically runs a script upon first bootup that already establishes another file to point the IP at. The arm's ROS node also is able to connect to MoveIt! as visualized by RViz (the default robot visualization program incorporated in ROS). The PTZ gimbal unit does properly boot upon startup and calibrate itself with its internal encoders. However, it should be noted that there's some grinding when testing its hard-stop position. Unfortunately, the encoders that are utilized are not absolute encoders and thus must be re-calibrated upon every restart of the robot.

Next, we analyzed the functionality of the omniwheel controller node. As with usual 'move_base' nodes in ROS systems, the node takes inputs of command velocities. By convention the command velocity command is a twist composed of linear and angular vectors of size 3x1 for each vector of float type elements.

$$V_{cmd} = \begin{bmatrix} T_l & T_a \end{bmatrix} \tag{3.1}$$

$$T_l = \begin{bmatrix} x \\ y \\ z \end{bmatrix}, T_a = \begin{bmatrix} \theta_x \\ \theta_y \\ \theta_z \end{bmatrix} \tag{3.2}$$

3.3.4 Testing Sensors

Each of the sensors were tested separately and visualized within RViz. As seen in the following figure, the LiDAR data is represented as the pointcloud2 data package. The ZED2 depth cameras are depth clouds with detected RGB data overlayed on top. In addition to the depth data, the ZED2 also features normal monocular camera data which can be seen on the right side of the figure. The robot has two

ZED2 cameras, and the top image is the one mounted on the actuated gimbal and the bottom image is from the static ZED2 camera. The robot that is visualized in the main image is a state estimation model of the robot based on the encoder positional readings from the wheels and arm joints.

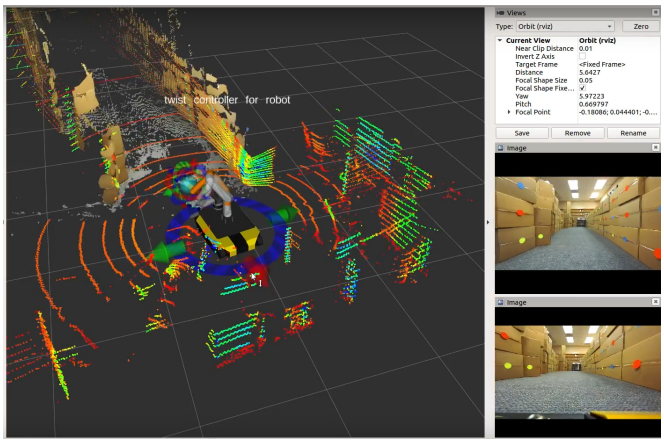


Figure 3.5: RViz rendering of the real robot's sensor readings and estimated state.

3.3.5 Setting up Development Computer

With the robot setup the next important milestone to work on is the setup of the development computer environment. The purpose of the development computer is to develop and test packages in simulation without the need of the real robot. This helps in speeding up development time among multiple people as they can have their own instances of the simulated robot to test custom packages on. Clearpath Robotics has their own Ubuntu repository dependencies that can be added to any Ubuntu machine. These Clearpath packages will take care of the basic programs needed to simulate the mobile base and corresponding sensors that have been mounted on it. Since the development laptop lacks an NVIDIA card, CUDA would not be able to be installed completely. Thus, emulation mode was the only option for the CUDA dependency requirements. It is important to note that the

ZED2 SDK dependency's installation script must not be used to install CUDA as the repository that it refers to is out of date. The Ubuntu universal apt repository has compatible binaries for CUDA made by NVIDIA (after adding their key), so that was the installation method chosen. After CUDA is installed, the ZED2 SDK is installed, with python bindings, answering Y to all the options when presented.

There are also custom ROS packages that need to be built from source. Most of the packages seen on the real robot need to be cloned over to the development computer's catkin workspace. Due to the Kinova arm, the ros_kortex package must be used. However, the CMakeLists.txt file options are outdated in comparison to the C++ compiler needed for the other (more updated) packages. Thus, the CMake files and the package xml files must be updated to include the C++17 compiler options. It is important to also make sure that the packages are the right versions corresponding to the operating system version of the development computer. For example, if the computer is running Ubuntu 20.04 then the ROS version would be Noetic and thus the ROS packages must be their Noetic branch. It is important to note that despite the real robot running on Ubuntu 18.04 with ROS version Melodic, most if not all packages are cross compatible between Melodic and Noetic.

3.3.6 Testing ROS system

it is important to test out all the sensors and actuators within the robot prior to further development. This includes an analysis between the simulated ROS DOT graphs and the real robot ROS DOT graphs to compare which nodes exist in each instance. After this it is important to view everything within RViz to see if the robot functions as intended in simulation. However, it is important to note that within

simulation the ZED2 SDK is closed source and thus not included as a gazebo plugin within the physics simulation. There is also a lack of PTU gimbal response and of course the Kinova arm is still in a fail state due to a bug. Next steps include creating the ROS nodes needed for simulating the ZED, FLIR PTU gimbal drivers, and also fixing the ROS_kortex package for the arm, which might not be possible without the aid of the Kinova company themselves.

3.3.7 Testing and Debugging Physics Simulation

The following sub-sections will cover the process in detail about the first prototype testing procedure and results. The testing revolved around the physics simulation of the robot in Gazebo.

3.3.7.1 First Prototype Testing Setup

The equipment that was used for this lab testing was divided into two parts, hardware and software. Since the testing revolved around the physics simulation for the mobile manipulation robot there are few hardware requirements needed to run the tests. As such, the only hardware necessary is a development computer with adequate enough specifications and a mouse. The computers specifications should be that of a mid-range CPU capable of a clock speed up to 3.7 GHz and at least 16 GB of RAM to be able to reliably run the physics engine, Bullet, used behind the Gazebo simulator. In terms of GPUs, an integrated graphics card was used during the in-person lab testing but frame-rate performance was less than ideal (though this does not affect the testing criteria), so a discrete graphics card will be used in any further live simulation demonstrations. The mouse is a necessary component for the testing demonstration due to the 3D environments of both

the simulation and visualization/planner that need to be navigated which cannot be done through touch nor trackpad controls. The software revolves around the framework known as Robot Operating System (ROS) version Noetic running on Ubuntu 20.04. ROS1 is chosen instead of ROS2 due to more support and features that are within the ROS1 community. Most of the scripts within the ROS1 packages that create the nodes are written in Python3 with configuration files written in yaml and launch scripts being written in xml. The packages needed are a combination of off-the-shelf packages, such as ros_kortex, and packages that are custom made for this project (and thus are the focus of the contributions and tests for this report), such as bosu08_custom. The full list of custom ROS packages used is presented below:

- bosu08_custom
- dingo_control
- dingo_description
- dingo_msgs
- dingo_navigation
- ros_kortex
- zed-ros-wrapper

It is important to note that there are also some basic ROS packages that are also being depended on for this test to function but should come standard with a full desktop installation of any ROS version and as such is not listed. The setup is relatively simple and revolves around making sure the packages are properly compiled within the catkin workspace of ROS. Once the packages are compiled and

the devels terminal scripts are sourced the next step of testing the robot arm and AMR (autonomous mobile robot) can commence.

3.3.7.2 First Prototype Testing Results

Due to some ros_kortex driver issues for the arm simulation, the test will focus on the mobile manipulars main components separately. Thus, the robot arm and AMR will be tested separately with separate terminal environment variables. The robotic arm will be tested on the physics model simulation and trajectory planning. The AMR will be tested on its physics simulation model as well as its pose estimation with its simulated lidar sensor. Measurable criteria are mainly determined by the performance of the packages initializations and runtime performance. More specifically, the physics simulation models must be rendered correctly in relation to the actual real robot and cannot have any fatal errors that lead to a crash. The visualizations of the robots pose estimations and transformation states must reflect what is observed in the physics simulation without any world frame errors or discrepancies. The robot arm should be able to move to any specified pose with the background trajectory planner not colliding with any of the components of the combined arm and mobile base. The AMR should be able to accurately create a point cloud map of its environment that changes in relation to the AMRs position in its virtual environment. The results of the in-person lab testing were all positive in the sense that each criteria was successfully performed and executed. All of the above were completed without any fatal errors and were accurate in simulation and the visualization. The objectives for the arm include that of the physics simulation and robot visualization programs running without fatal errors, correctly rendered robot models in both the simulator and visualizer, and the execution of

the optimized motion plan. The objectives for the AMR include that of the physics simulation and robot visualization programs running without fatal errors. The correctly rendered robot models in both the simulator and visualizer were also achieved with the sensor data reflecting the environment seen in the simulation and projected in the visualizations robot pose and data representation.

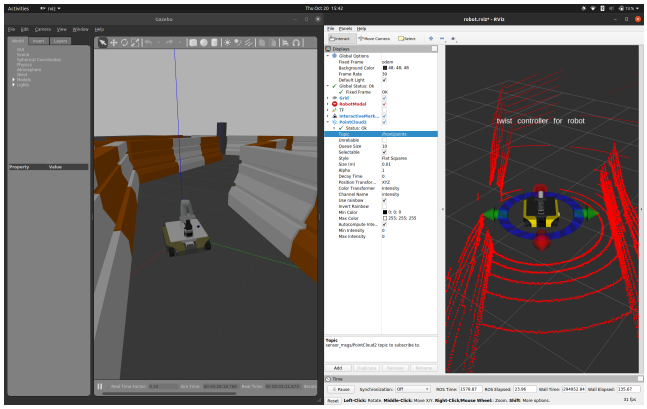


Figure 3.6: Gazebo physics simulation with RViz rendering of what the robot perceives within the simulator.

3.3.8 Testing and Debugging the Real Robot

After work on the simulation there were some discoveries about the other real robot units' hardware. In particular, the Dingo-O unit that is meant to be the second mobile manipulator has hardware issues that disables it from mounting its Kinova Gen3 LITE arm in a stable and reliable fashion. The problem lies with the mounting plate of the robot in that the mounting holes do not line up with the standoffs from the mobile base itself. This misalignment is particularly confounding due to the fact that all the robot plates are machined milled with the same base pattern. Thus, the three other Dingo-O units have their mounting plates perfectly aligned and screwed on. After a meeting with the engineers at Clearpath Robotics the following notes were discovered. The dingo actually has two central

steel plates, each with four contact points that connect the rest of the robot chassis to any add-ons, including the mounting plates. These two steel plates may have been wrapped during shipping, but regardless of origin of cause, the effect is that the standoffs are angled slightly off from ninety degrees vertical. Due to the nature of the plates being attached with only four contact points each over a long space it is susceptible to being tilted depending on the amount of screw turns made on the plates during installation. Thus, the skewed mounting plate and standoffs may be from the dingo plates being screwed in at different amounts between each plates' four screw holes. This issue has been recently resolved, but it is important to keep in mind in future testing with the real robots.

3.3.9 Implementation

A kinematic model is used in the MPC of the mobile manipulator where the base link is holonomic. Joint angle positions are used to describe the state of the arm $x_{arm} \in \mathbb{R}^n$ and the base falls within the special euclidian group $x_{base} \in SE(3)$. The base is described by its orientation q_{base}, which is a quaternion, and its position in the static world frame, which is a vector $r_{base} \in \mathbb{R}^3$. This full description of the base is considered so that the system is more generalizable to different robot platforms. Control inputs to the robot are defined as robot arm input controlled by joint velocities $u_{arm} = [\dot{\phi_1}, \dot{\phi_2}, ..., \dot{\phi_n}]$ and robot base input controlled by desired base twist, forward velocity, and turning rate $u_{base} = [v, \dot{\phi_{base}}]$.

For the SLQ-MPC algorithm the robot is described by the following mathematical system model:

$$x = \begin{bmatrix} x_{base} \\ x_{arm} \end{bmatrix} \tag{3.3}$$

$$u = \begin{bmatrix} v \\ \dot{\phi_1} \\ \dot{\phi_2} \\ \dots \\ \dot{\phi_n} \end{bmatrix} \tag{3.4}$$

$$\dot{x} = \begin{bmatrix} q_{base}[*]\frac{1}{2}k\dot{\phi_{base}} \\ q_{base}(*)[v,0,0]^T \\ v_{arm} \end{bmatrix} \tag{3.5}$$

where x is the state of the full mobile manipulator, $[*]$ represents the Hamiltonian quaternion product, and $(*)$ represents a vector rotation by a quaternion.

The cost function for the MPC integrates the end-effector tracking error which is denoted as $C_{ee\_tracking}$ and the weighted control effort $u^T Ru$ over a time horizon of T. The desired configuration state for the mobile manipulator is denoted as x_d as seen in the cost formulation:

$$J(x, x_d, u) = \int \tau = t_0 + T\tau = t_0 L(x(\tau), x_d(\tau), u(\tau))d\tau \tag{3.6}$$

$$L(x, x_d, u) = C_{ee\_tracking} + u^T Ru \tag{3.7}$$

Deviations of the end-effector pose of the robot $p_{ee}(x)$, which is a function of the current state, from that of the desired end-effector pose $p_{ee_d}(t)$, which may vary in time, are penalized. The translational error component is found by the difference between two position vectors. The rotational error is based on the orientation quaternions of the current and desired pose. Forward kinematics from the robot base to the end-effector is multiplied with the world frame to base transformation

based on x_{base}. The end-effector tracking term $C_{ee\_tracking}$ is thus defined as the sum of squares of the error functions:

$$C_{ee\_tracking} = ||e_{pos}||_2^2 + ||e_{ore}||_2^2 \tag{3.8}$$

where e_{pos} is positional error and e_{ore} is the orientational error of the end-effector. To achieve a stable control law from the convergence of the SLQ algorithm, the property of positive semi-definite must be held by the second derivative of the cost function $J(x, x_d, u)$ with respect to the state x. The robot's FK results are not always positive semi-definite and thus the second derivative of the cost function is approximated where the second derivative of $C_{ee\_tracking}$ is defined as:

$$\ddot{C}_{ee\_tracking} \approx 2\dot{e}_{pos}\dot{e}_{pos}^T + 2\dot{e}_{ore}\dot{e}_{ore}^T \tag{3.9}$$

The R matrix within the weighted control term of the cost function is set to be the appropriately dimensioned identity matrix. The MPC control module was implemented with the OCS2 library, which also features a SLQ solver that operated over a given time horizon T of two seconds. All of this operates a ROS node upon execution of the ROS launch file.

CHAPTER 4
Conclusions

In conclusion, this thesis has investigated the foundational principles of multi-agent mobile manipulation with a focus on kinematic redundancy. By exploring various approaches to resolve kinematic redundancy, consistently repeatable behaviors and predictable motions can be created. Furthermore, principles of dynamic task partitioning have been proposed to tackle tasks that require multiple agents' cooperation. The empirical validation using two advanced platforms demonstrated the effectiveness of the proposed principles. This study provides valuable insights into multi-agent mobile manipulation and contributes to the development of practical applications in this field.

4.1 FUTURE WORK

Further developments in this work centers around the formulation and coordination between the robot agents. This work describes a simple human defined grasping strategy for normal rigid body geometries. In the real world this is not the case, objects will have deformities and unusual geometries that are not conductive to easy grasping points. Multi-robot manipulation involve complex control entities that can be represented as independent dynamic parts. Research with reinforcement learning to solve this task has been promising, but still challenged by the sample complexity and large exploration requirements. Approaches, such as the one seen in [31], show promise in overcoming these issues by using latent actions spaces that are learned and shared across multiple robot agents. Mathematically, latent space is defined as an embedding of a set of items within a manifold where similar items are closer together in distance. Such multi-agent manipulation

learning methods tend to focus on static robot arms, so high dimensional mobile-manipulators promise to be an aspect of novel research and advancements.

BIBLIOGRAPHY

[1] Sandakalum, T.; Ang, M.H., Jr. Motion Planning for Mobile ManipulatorsA Systematic Review. Machines 2022, 10, 97. https://doi.org/10.3390/machines10020097

[2] Malone, B. George Devol: A Life Devoted to Invention, and RobotsIEEE Spectrum; IEEE: Piscataway, NJ, USA, 2011.

[3] Sereinig, M.; Werth, W.; Faller, L.M. A review of the challenges in mobile manipulation: Systems design and RoboCup challenges. E I Elektrotech. Inform. 2020, 137, 297308.

[4] Foulon, G.; Fourquet, J.Y.; Renaud, M. Coordinating mobility and manipulation using nonholonomic mobile manipulators. Control Eng. Pract. 1999, 7, 391399.

[5] M. Cefalo, G. Oriolo and M. Vendittelli, "Task-constrained motion planning with moving obstacles," 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2013, pp. 5758-5763, doi: 10.1109/IROS.2013.6697190.

[6] O. Brock and L. E. Kavraki, "Decomposition-based motion planning: a framework for real-time motion planning in high-dimensional configuration spaces," Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164), 2001, pp. 1469-1474 vol.2, doi: 10.1109/ROBOT.2001.932817.

[7] Dharmawan, A. G., Foong, S., and Soh, G. S. (April 5, 2018). "Task-Constrained Optimal Motion Planning of Redundant Robots Via Sequential Expanded Lagrangian Homotopy." ASME. J. Mechanisms Robotics. June 2018; 10(3): 031010. https://doi.org/10.1115/1.4039395

[8] J. Su and W. Xie, "Motion Planning and Coordination for Robot Systems Based on Representation Space," in IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), vol. 41, no. 1, pp. 248-259, Feb. 2011, doi: 10.1109/TSMCB.2010.2051025.

[9] J. Baillieul, "Kinematic programming alternatives for redundant manipulators," Proceedings. 1985 IEEE International Conference on Robotics and Automation, 1985, pp. 722-728, doi: 10.1109/ROBOT.1985.1087234.

[10] M. Galicki and M. Banaszkiewicz, "Optimal trajectory tracking control of omni-directional mobile robots," 2019 12th International Workshop on Robot Motion and Control (RoMoCo), 2019, pp. 137-142, doi: 10.1109/RoMoCo.2019.8787377.

[11] Z. Wang, M. Schwager, Multi-Robot Manipulation without Communication, InăInternational Symposium on Distributed Autonomous Robotic Systems(DARS 14), November, 2014.

[12] J. Alonso-Mora, R. Knepper, R. Siegwart and D. Rus, "Local motion planning for collaborative multi-robot manipulation of deformable objects," 2015 IEEE International Conference on Robotics and Automation (ICRA), 2015, pp. 5495-5502, doi: 10.1109/ICRA.2015.7139967.

[13] B. Wei, Y. Li, Y. Zhang, Y. Li and S. Shu, "Dynamic Modeling of Mobile Manipulator based on Floating-Like Base," 2022 IEEE 6th Information Technology and Mechatronics Engineering Conference (ITOEC), 2022, pp. 993-998, doi: 10.1109/ITOEC53115.2022.9734599.

[14] Kevin M. Lynch and Frank C. Park. 2017. Modern Robotics: Mechanics, Planning, and Control (1st. ed.). Cambridge University Press, USA.

[15] K. Nagatani, T. Hirayama, A. Gofuku and Y. Tanaka, "Motion planning for mobile manipulator with keeping manipulability," IEEE/RSJ International Conference on Intelligent Robots and Systems, Lausanne, Switzerland, 2002, pp. 1663-1668 vol.2, doi: 10.1109/IRDS.2002.1043994.

[16] J. Denavit and R.S. Hartenberg, 1955, "A kinematic notation for lower-pair mechanisms based on matrices." Trans ASME J. Appl. Mech, 23:215221.

[17] J. Pankert and M. Hutter, "Perceptive Model Predictive Control for Continuous Mobile Manipulation," in IEEE Robotics and Automation Letters, vol. 5, no. 4, pp. 6177-6184, Oct. 2020, doi: 10.1109/LRA.2020.3010721.

[18] F. Farshidian, M. Neunert, A. W. Winkler, G. Rey, and J. Buchli, An efficient optimal planning and control framework for quadrupedal locomotion, Proc. IEEE Int. Conf. Robot. and Autom., May 2017, pp. 93100.

[19] J.-P. Sleiman, F. Farshidian, and M. Hutter, Constraint Handling in Continuous-Time DDP-Based Model Predictive Control. arXiv, Mar. 26, 2021. Accessed: Apr. 06, 2023. [Online]. Available: http://arxiv.org/abs/2101.06067

[20] R. Stengel, Optimal Control and Estimation, ser. Dover Books on Advanced Mathematics. Dover Publications, 1994.

[21] F. Farshidian, M. Neunert, A. W. Winkler, G. Rey, and J. Buchli, An efficient optimal planning and control framework for quadrupedal locomotion, in IEEE International Conference on Robotics and Automation, ICRA Singapore, 2017, pp. 93100

[22] J. Alonso-Mora, R. Knepper, R. Siegwart and D. Rus, "Local motion planning for collaborative multi-robot manipulation of deformable objects," 2015 IEEE International Conference on Robotics and Automation (ICRA), Seattle, WA, USA, 2015, pp. 5495-5502, doi: 10.1109/ICRA.2015.7139967.

[23] R. W. Brockett, Finite dimensional linear systems, SIAM edition. in Classics in applied mathematics, no. 74. Philadelphia: Society for Industrial and Applied Mathematics, 2015.

[24] N. Koenig and A. Howard, "Design and use paradigms for Gazebo, an open-source multi-robot simulator," 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566), Sendai, Japan, 2004, pp. 2149-2154 vol.3, doi: 10.1109/IROS.2004.1389727.

[25] Stanford Artificial Intelligence Laboratory et al. Robotic Operating System [Internet]. 2018. Available from: https://www.ros.org

[26] Richard M. Murray, S. Shankar Sastry, and Li Zexiang. 1994. A Mathematical Introduction to Robotic Manipulation (1st. ed.). CRC Press, Inc., USA.

[27] R. Grandia, F. Farshidian, R. Ranftl, and M. Hutter, Feedback MPC for torque-controlled legged robots, in Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS), Nov. 2019, pp. 47304737, doi: 10.1109/IROS40897.2019.8968251

[28] S. Karaman and E. Frazzoli, Sampling-based algorithms for optimal motion planning, Int. J. Robot. Res., vol. 30, no. 7, pp. 846894, Jun. 2011.

[29] F. Farshidian, M. Neunert, A. W. Winkler, G. Rey, and J. Buchli, An efficient optimal planning and control framework for quadrupedal locomotion, Proc. IEEE Int. Conf. Robot. and Autom., May 2017, pp. 93100.

[30] M. Neunert et al., Fast nonlinear Model Predictive Control for unified trajectory optimization and tracking, in Proc. IEEE Int. Conf. Robot. and Autom., May 2016, pp. 13981404.

[31] Aljalbout, Elie Karl, Maximilian van der Smagt, Patrick. (2022). CLAS: Coordinating Multi-Robot Manipulation with Central Latent Action Spaces. 10.48550/arXiv.2211.15824.

[32] D. E. Whitney, "Resolved Motion Rate Control of Manipulators and Human Prostheses," in IEEE Transactions on Man-Machine Systems, vol. 10, no. 2, pp. 47-53, June 1969, doi: 10.1109/TMMS.1969.299896.

[33] C. A. Klein and C. -H. Huang, "Review of pseudoinverse control for use with kinematically redundant manipulators," in IEEE Transactions on Systems, Man, and Cybernetics, vol. SMC-13, no. 2, pp. 245-250, March-April 1983, doi: 10.1109/TSMC.1983.6313123.

[34] J. Baillieul, J. Hollerbach and R. Brockett, "Programming and control of kinematically redundant manipulators," The 23rd IEEE Conference on Decision and Control, Las Vegas, NV, USA, 1984, pp. 768-774, doi: 10.1109/CDC.1984.272110.

CURRICULUM VITAE

Roy Xing, MA

May 24, 2024

7 Spruce Street
Winchester, MA 01890
(781) 960 3202
royx@bu.edu

Boston University
Boston, MA 02118
royx@bu.edu

Academic Training:

12/2023(expected) BS Boston University, Boston, MA; Electrical Engineering

Undergraduate Research:

Title:	Multi-Agent Mobile Manipulation Methods
Thesis advisor:	John Baillieul, PhD
Defense date:	May 7, 2023
Summary:	Research into methods for coordination of multiple mobile manipulation agents to complete cooperation tasks.

Original, Publications (newest first):

1. Bahreinian, Mahroo Tron, Roberto. (2022). Output-Feedback Path Planning with Robustness to State-Dependent Errors. 4375-4380. 10.1109/CDC51059.2022.9993405.