5-2019

# Towards a Prototype Platform for ROS Integrations on a Ground Robot

Taylor Joseph Linville Whitaker
*University of Arkansas, Fayetteville*

Towards a Prototype Platform for ROS Integrations on a Ground Robot

A thesis submitted in partial fullfillment
of the requirements for the degree of
Master of Science in Computer Engineering

by

Taylor Whitaker
Univeristy of Arkansas
Bachelors of Science in Computer Science, 2016

May 2019
University of Arkansas

This thesis is approved for recommendation to the Graduate Council.

Xiaoqing Liu, PhD
Thesis Director

Christophe Bobda, PhD
Committee Member

Dale Thompson, PhD
Committee Member

## Abstract

The intent of this work was to develop, evaluate, and demonstrate a prototype robot platform on which ROS integrations could be explored. With observations of features and requirements of existing industrial and service mobile ground robots, a platform was designed and outfitted with appropriate components to enable the most common operational-critical functionalities and account for unforeseen components and features. The resulting Arlo Demonstration Robot accommodates basic mapping, localization, and navigation in both two and three-dimensional space as well as additional safety and teleoperation features. The control system is centered around the Zybo Z7 FPGA SoC hosting a custom hardware design. The platform is validated through an analysis of feature requirements and limitations and additional evaluations of a series of real-world use cases demonstrating high-level behaviors. In order to promote further development, this work serves as detailed documentation of the selection, implementation, and testing of this platform and complements initial binary releases for the Zybo Z7 control system and accompanying source code for the functionalities implemented. This prototype robot stack can be further developed to enable additional capabilities and validate its performance in other real-world scenarios or used as a reference for porting to alternative robot platforms.

## Acknowledgements

I would like to give special thanks to all members of my committee for their support and patience throughout this work. Additionally, I'd like to to thank the entire Computer Science and Computer Engineering Department at the University of Arkansas for an incredibly rewarding undergraduate and graduate career.

Having made available so many opportunities over the course of our collaboration, I owe much gratitude to Dr. Christophe Bobda. He has continuously been a reliable resource and mentor. Also, to the past and present members of the Smart Embedded Systems Lab with whom I have shared many ideas, thank you for always serving as sources of inspiration. I would like to acknowledge Dr. David Andrews for welcoming any and all questions I had during the development of this thesis. His input was extremely helpful in the writing process.

Finally, thank you to my partner, family, and friends that have given unfettering support and encouragement in my educational goals and in life. I would not have been able to reach this accomplishment had it not been for the foundation they have provided.

Contents

# Chapter 1

# Introduction

With the proliferation of robotics into almost all existing industries, the requirement for understanding the underlying systems of future infrastructure increases as well. This thesis intends to demonstrate the procedures of developing a new robotic platform prototype, purposed for functional exploration and feature demonstration, and integrating a widely accepted solution for control. It is meant to serve as a resource for researchers in the field attempting to extend the resulting platform or implement other unique platforms.

## 1.1 Motivation

The necessity for developers and maintainers of robotic systems has continued to grow along with the adoption rates of robotic systems throughout various industries. The increased use of robotics has consistently been replacing lower skilled laborers and used to complement higher skilled workers as these systems hold better economic viability. This replacement has aided in building infrastructures of computerized systems and created new requirements for development and maintenance of said infrastructure. The International Federation of Robotics has published an analysis of past industrial robotic sales estimates and predictions of future growth, seen in Figure 1.1. With 15% growth of industrial robot operation stock from 2016-2017 to 2.1 million units and a conservative prediction of 16% average yearly growth from 2018-2021, the market for industrial robotics will continue to be a leading focus of robotics technology [1]. Though other directions such as service robotics were a mere 5% of the global market value of $48 billion in 2017, this market is expected to grow to an estimated $37 billion between 2019 and 2021 [1, 2].

Thus, the need for a widespread general understanding of robotic systems arises. It is crucial for many industries that steps be taken to meet current and ensure future labor supply for these occupations. Such steps could be introducing robotics and related concepts to current workers, to those outside of the field, and to children from an early age in educational systems.

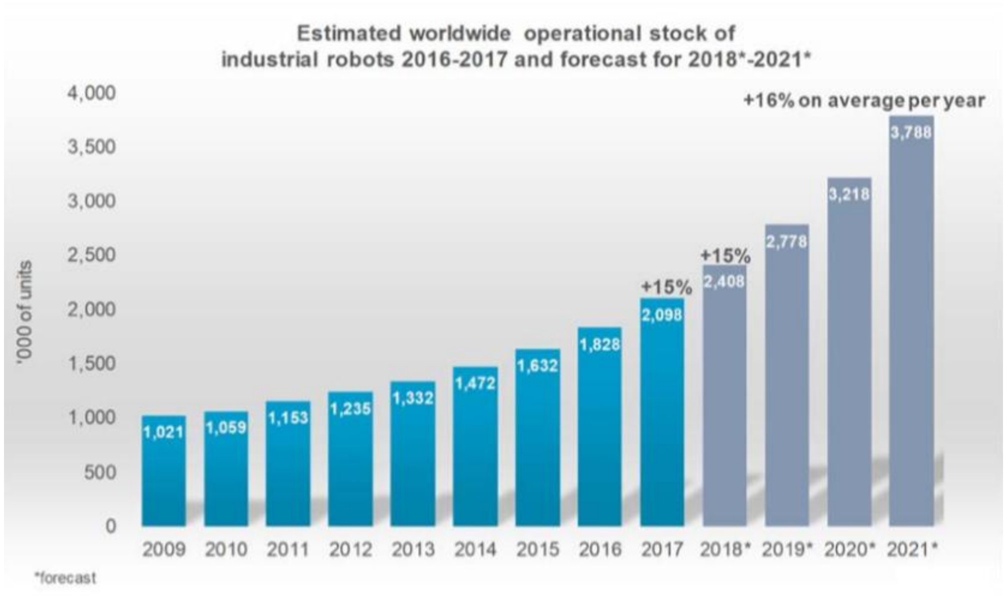


Figure 1.1: Estimated Annual Worldwide Operation Stock of Industrial Robotics, Source: International Federation of Robotics [1]

In order to facilitate the spread of understanding, the development tools and robotic platforms that are used for education and in industry should translate easily. This partially depends on development resources that can be used by both students and system designers. Additionally, it requires systems suitable for hosting industrial robots or hobby contraptions alike. One such development tool with heavy adoption, the Robotic Operating System (ROS pronouced "r-aws"), is working towards a versatile framework capable of setting up complex collaborative robot systems, as well as simple line-following robots. As the industry, open-source community, and ROS developers all serve a hand in the rapidly evolving complex of ROS, documentation and features are sometimes unable to keep up with the most recent standards. This thesis aimed to contribute to the collection of the most recent community documentation and open-source packages. In addition, the platform prototype was developed to contribute to the fleet of ROS compatible platforms that are suitable for educational experimentation and demonstrating features found in current commercialized robots.

Further, robots available today place a number of constraints on the control systems. As is typical in most emebeded systems, consideration must be given to the minimization of the size, weight, and power (SWAP) of the processing core. However, today's platforms also require more versatiltiy from control systems as they must be compututationally suitable for allowing efficient and optimized performance of advanced functionalities. This work intends to present a robotic platform prototype with which the control system can serves as an effective host for implementation and exploration of many existing features while still remaining suitable for future efforts. To facilitate this, the protoptype platform makes use of hardware and software partitioning with a system-on-chip development board serving as the basis of the primary control system.

## 1.2 Thesis Objectives

The below objectives were set out to guide the progress of this work. Each of these objectives was derived from the generalized goal set out in the introduction of this chapter: to demonstrate the procedures of developing a new robotic platform and integrating a widely accepted solution for control.

- Establish current state of the art in commercialized robotics
- Select a subset of operational-critical capabilities seen in survey of state of the art
- Select a robot platform suitable for demonstrating these capabilities
- Implement a control system capable of utilizing ROS
- Effectuate the selected capability set
- Evaluate and demonstrate the robot prototype
- Release an open-source ROS robot stack for a robot platform

In order to develop a suitable research platform, a first consideration was the intended demonstrations this prototype could provide upon the compeltion of this work. Taking from the motivation, the platform must contribute to the effort of bridging robotics in industry and those in educational settings. Conducting a survey of the current state of the robotics industry allowed comparisons to be drawn from the various applications and features implemented in differing fields. Though new applications are conceived on an almost daily basis, the fields most embracing robotics tend to be industrial and service. Industrial robots are used in logistics and manufacturing and service robots are being introduced into medical and educational facilities, homes, and even national defense armories [3]. Platforms that have been successfully commercialized and deployed in the aforementioned industries were the focus of this survey as the processes of commercialization and deployment can be considered fitting methods of product validation. These platforms were analyzed for their specific applications, capabilities, and complexities in order to produce a subset of common operational-critical functionalities to be demonstrated on a research grade robot platform.

With a set of intended features, a platform was chosen to host the functionalities and allow for additional resources for unforeseen features. The implementation process focused on exposing the low-level capabilities of the robot platform to a ROS centric control system. Upon completing the initial robot platform, a series of evaluations and demonstrations were to be chosen to analyze the viability of implementing industry standard features on a platform constructed with educational and hobbyist materials. Lastly, the resulting platform's firmware and source code were to be made available as a ROS robot stack.

## 1.3 Thesis Structure

With a fair number of abstraction layers being discussed in this work, mentioning the structure of the discussion is warranted. Chapter 2 briefly reviews the state of the art in the field of robotics with a look into currently available robot platforms and the latest developments of critical features. Moving to the actual implementation, Chapters 3-5 detail the process of each stage of developing the Arlo Demonstration Robot. Chapter 3 develops a baseline platform with Chapter 4 integrating the Robotic Operating System framework. Establishing the robot platform as suitable for demonstrating advanced features, Chapter 5 develops high-level functionalities and effective demonstrations. Discussions for the final implementation are placed in Chapter 6 along with an analysis of the platform itself. To conclude, Chapter 7 summarizes the contributions of this work and speculates on possible directions for future efforts.

# Chapter 2

# State of the Art

## 2.1 Robots in Industry

Two of the domains dominating robotics, industrial and service, have seen significant growth and the introduction of many technological developments [4].

### 2.1.1 Industrial Applications

In industrial and manufacturing settings, robots are quickly becoming commonplace. In this particular domain, the form factor of adopted robots varies widely, though this work will focus on the relevant ground vehicles currently in use. Three manufacturers of successful commercialization of ground robots are Amazon Robotics, Mobile Industrial Robots, and OTTO Motors.

One commonly known online shopping company, Amazon, has seen incredible growth over the years and is continuing to catalyze that growth with maximum efficiency in their warehouses and order fulfillment centers. Recognizing the benefits of automation through robotics, Amazon purchased Kiva Systems in 2012 for $775 million and renamed it to Amazon Robotics [5]. They have since deployed hundreds of thousands of ground robots for payload transportation in warehouses and fulfillment centers. With identification tags reading Amazon Drive #XXXXXX, each of these two-wheeled robots drives among several thousand others with extreme precision [6]. With computerized barcode stickers on the floor, the robots can navigate around the warehouse, sense and avoid other robots, and fulfill the task they are assigned. With these robots, they simply are sent to collect a payload from a predetermined position and transport it to another location. With the online purchasing service Amazon provides, the robot payload is typically a collection of items in a warehouse that must be delivered to a human operator for selecting the specific order [7]. Such a payload can be seen in Figure 2.1, courtesy of [8]. Amazon Robotics Vice President Brad Porter discloses robotic systems "employ multiple safety systems ranging from training materials, to physical barriers to entry, to process controls, to onboard" [9]. All measures are in an effort to ensure the safety across the board, for robots, the facilities, and most importantly the workers in close proximity. Human working zones are usually explicitly marked or barricaded to ensure no malfunctioning robot can cause harm. However, there are still many environments that lack this amount of protection, such as training facilities, and Amazon is aware of this. In fact, over the last year, Amazon has been issuing a vest to its workers to add yet another measure to improve safety. The vest acts as a sort of beacon allowing robots to "detect the human from farther away and smartly update its travel plan to steer clear," all without requiring additional inputs from the worker [9].

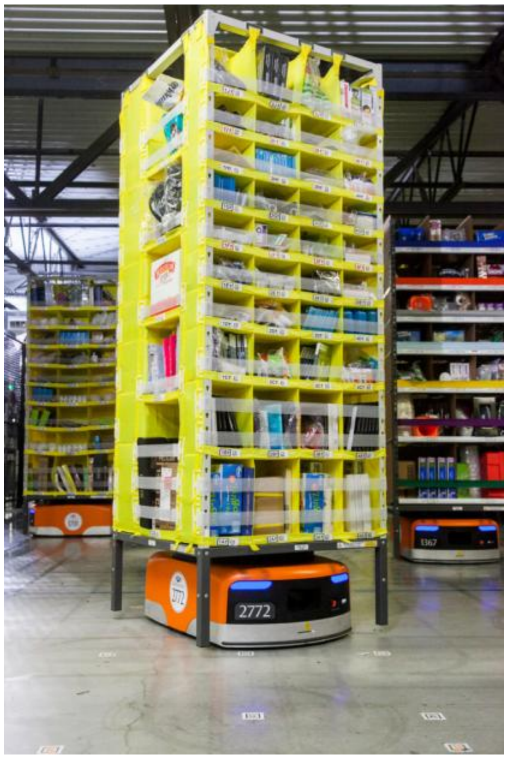


Figure 2.1: Early Kiva Robot by Kiva Systems,Predecessor to the current Amazon Drive Robots, Source: Architects Newspaper [8]

Both Otto Motors and Mobile Industrial Robots (MiR) are manufacturing robots for meeting the varying internal transportation and logistical needs of companies. Otto and MiR have developed an entire system of hardware and proprietary software to power their fleets of autonomous mobile robots. Each has its own fleet management software allowing for collaboration among robots and humans and global environmental awareness. Contrasting the Amazon Drive robots requiring a modification of their environment, Otto and MiR robots use integrated safety rated LIDAR sensors as the primary source of data for mapping their environment [10, 11]. The LIDARs are hosted at the front and rear of the Otto series robots, while the MiR robots place these at opposing corners for a true 360° field of view [12–15]. Additional sensors can be seen in the cutouts on the front and rear of the MiR robots, while Otto robots either do not include many additional sensors and/or are well hidden. Though no technical documentation has been released by either company, it can be assumed both utilize a LIDAR centric mapping method with additional sensor/camera support. Both systems allow for a close to non-invasive deployment of autonomous robots in nearly any existing warehouse setup. Human interaction with the robots can be done through programmable buttons, mobile device applications, or automation tied directly to the production line. With additional features such as the familiar indicator lights on the Otto robots and the real-time traffic management system of MiR, these companies are attempting to set standards for autonomous driving in industrial settings by verifying safety and ease of integration. These robots also enable flexibility for their applications with support for interchangeable modules that can transport various shaped payloads or extend the features of the robot [10, 11].

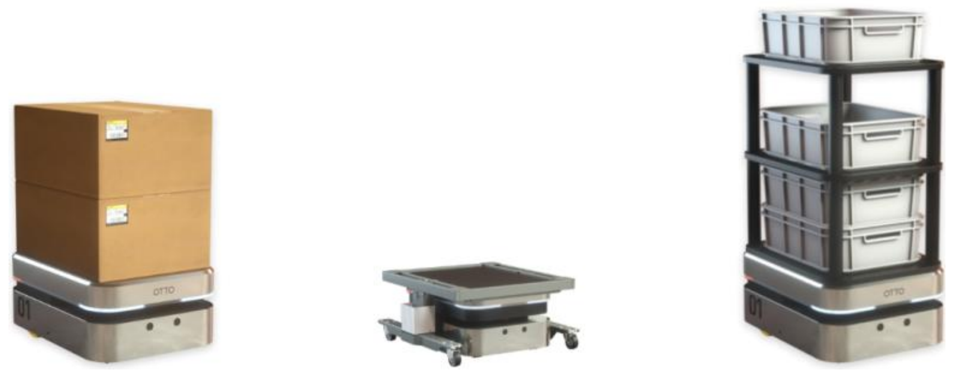


Figure 2.2: OTTO 100 Robot with Various Configurations, Source: OTTO Motors [13]

Figure 2.3: MiR 100 Robot, Source: Mobile Industrial Robots [15]

### 2.1.2 Service Applications

Though the introduction of robotics to the service industry required an initial infrastructure of industrial robotics to be laid, the influx of consumer and commercial service robots is rising [2]. The market for consumer home robots and commercial service robots has been heavily affected by the introduction of cheaper components and increased microcontroller speeds and complexities needed for sensing and control [16].

#### Robots in the Home

Unfortunately, there have not been any full-service robots to replace a maid service developed yet, though there have been a large number of robots targeted for specific tasks in the home.

Perhaps the most widely known and successful household robot is the Roomba vacuum manufactured by iRobot. This series of robots are tasked with the simple chore of vacuuming the home. iRobot has recently extended the offering of home robots to include the Braava mopping robot. With cleaning routes following no discernable path, the robots can successfully complete their purpose without advanced mapping. They are outfitted with bump sensors for corrections, infrared receivers, and cliff and wheel drop sensors to detect sudden terrain changes. They explore their environment and find their way back to an infrared emitting docking station. All sensors are added to ensure they are able to provide full coverage of the home's floors. They are packaged in an appropriately formed shell that has a low profile to the ground and efficiently placed wheels for maximum maneuverability in tight spaces and close to barriers. These iRobot products are developed to be minimally invasive to the space and routines of the home with features like scheduling and auto return to base for charging and continuous uptime [17, 18]. Interacting with these robots can be done with a set of basic action buttons on the physical units or in higher-end models through smartphone applications. With such a simple task, small footprint, and an inherent lack of potential hazards, these robots do not require much consideration for safety apart from avoiding falls down the stairs or mopping the carpet.

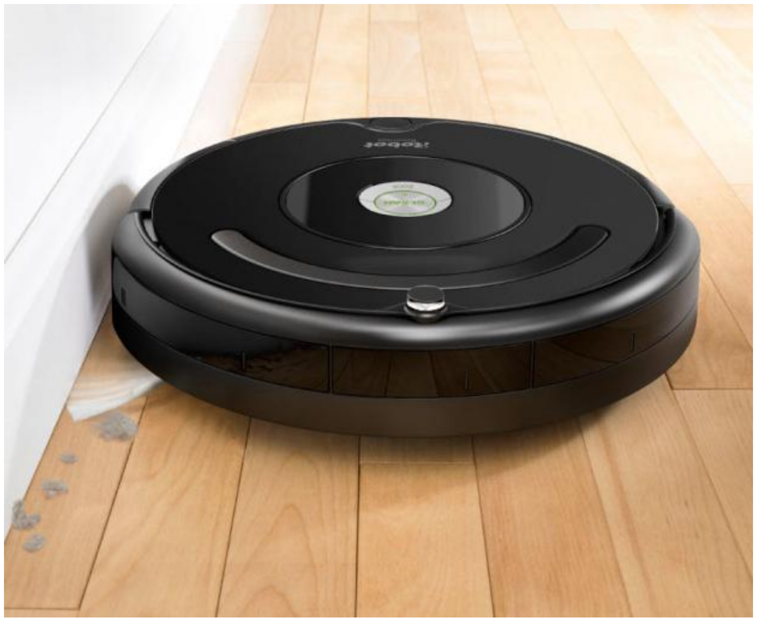


Figure 2.4: Roomba 675 Cleaning Robot, Source: iRobot [18]

Another tedious chore that has had its well-deserved opportunity for automation is lawn maintenance. Some companies in the lawn equipment business have released autonomous lawn mowers, such as Husqvarna with the Automower, removing the need for manually mowing the lawn. With iRobot performing a beta launch in 2019 of its Terra autonomous lawnmower [19], they will soon be providing worthy competition in the market. The operation of, and available methods of interacting with, these robots is quite similar to those purposed for vacuuming and mopping. The primary task is to provide a full coverage cut of the lawn. This task, though similar, introduces its own set of challenges. For instance, the vacuumming and mopping robots utilize bump sensors for sensing the environment as the walls of a home are easily detectable. However, in an outdoor environment, it would not be a good decision to allow the mower to bump into objects that could be moved or damaged by it. This application requires the use of high-velocity metal blades to perform its task and thus safety must be of prime importance. The Automower leverages a protective outer shell to enclose the blades, though still maintains an appropriate form factor. In order to map, localize, and navigate the outdoor environment safely, the Automower employs LIDAR sensors for hazard detection, physical boundaries for declaring a working space, and intertial readings to ensure the robot stays level with the ground. An emergency stop is also connected to the intertial measurement unit (IMU) to ensure that power to the blades is immediately cut upon the blades becoming exposed [20, 21].

Figure 2.5: AutoMower 310 Mowing Robot, Source: Husqvarna [21]

### Commercial Service Robots

With much influence from the development of the fields discussed thus far, service robots began to appear in stores, hospitals, hotels, and even in parking lots. These areas have given rise to robotics targetted for direct social interaction with humans and robotics to tackle specific mundane tasks.

The OSHBot, developed by Fellow Robots and Lowe's Innovation Labs, aimed to aid the consumer at the store and allow employees to devote more time to customers. It hosted an onboard display to present the catalog of merchandise the store supplies. Users were able to select the desired items and the OSHBot will direct the customer to the location in the store where the item may be found [22]. The robot builds a map of its environment when the store's are closes, roaming the aisles to ensure its map is always up to date and detecting environmental changes such as shelving layouts [23]. LIDAR and cameras are used for navigation, map creation, and map updates [24]. The latest version of this robot, the LoweBot pictured in Figure 2.6 [25], builds upon its predecessor and additionally includes the ability for voice interaction, in multiple languages, with customers and real-time inventory tracking [26]. With the success of these robot deployments, stores such as Walmart and Target are currently looking to implement their own robot solutions [27].

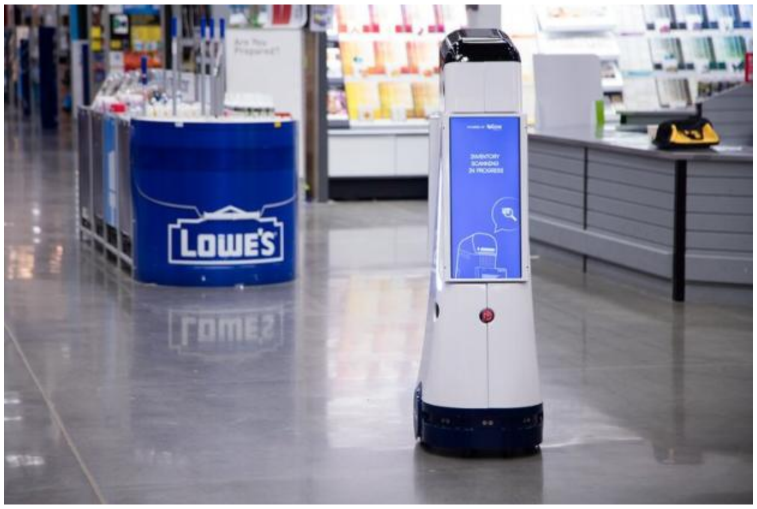


Figure 2.6: LoweBot Conceirge Robot, Source: Lowe's Innovation Labs [25]

Many businesses such as hotels, amusement parks, and restaurants are adopting concierge robots for more efficient service to their guests [4]. These robots complement current workers in order to better address the needs of hotel guests requesting room service, restaurant goers ordering meals, and theme park attendants needing general assistance. Additionally, hospitals can handle autonomous deliveries and uphold patient hospitality with concierge robots.

One such robot, Relay, is manufactured by Savoike and purposed for healthcare, hospitality, and logistics businesses [28–30]. It can be seen in Figure 2.7. It is an autonomous mobile robot capable of transporting small packages to some location in its known environment. With a footprint no larger than a human's, it can easily operate within an area with light human traffic. The robot stands at an appropriate height for easily interacting with the touch screen display that allows control for sending and receiving deliveries. They can also be controlled via a proprietary management system hosted by Savoike, in addition to general management of the site the robot operates in. Though no official technical reports have been made for the Relay robot, assumptions drawn from information freely available on the internet can be made. The lower black strip on the robot can be presumed to be a range finding sensor array, as the window would provide a 180° field of view and could be used for collecting data for a simple two-dimensional map of the environment. The window below the display could house a camera for additional hazard detection. The robot demonstrates its consideration for safety with its immediate halt of motion when an object obstructs its path or detects a collision. However, there appears to be no available emergency stop button which indicates the low amount of risk this system introduces to a facility.

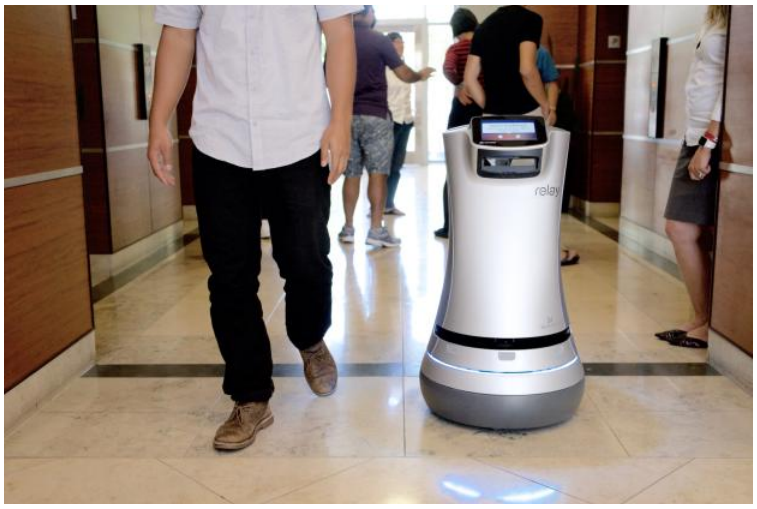


Figure 2.7: Relay Conceigre Robot, Source: Savoike [30]

## 2.2 Operational-Critical Functionalities

With the above review of current successful commercializations of various ground robots, this work determined the most commonly adopted features. Navigation, mapping, and localization are utilized by all of the discussed platforms. This section looks further at these capabilities and introduces some of the efforts towards their utilization on robotic platforms.

### 2.2.1 Navigation

A ground robot simply implies the use of high-level processing and sensing on a ground-based platform, whether its position be static or dynamic in an environment. Of the ground robots this work considered, all were mobile ground robots that allowed for movement within the hosting environments, typically via teleoperation. However of particular interest, many can be classified as autonomous mobile robots due to their abilities to manage the navigation through their environment without manual control or supervision. This ability opens the opportunity for robot deployments that impose little to no restrictions or modifications of the environment. The infrastructure needed for implementing some system of autonomous mobile robots is thus minimized.

Enabling autonomous navigation is a feat that has dominated the field of robotics for many years. First attempts with DARPA's Shakey robot in the 1960's were initially considered a failure as autonomous operation was never reached. Though, it did allow for sensing with a steerable TV camera and ultrasonic and touch sensors as well as manipulation of objects in an environment, sending research efforts into the areas of planning and vision processing [31]. A complex stereo vision system was used on the Stanford Cart with a single camera taking nine images as it was translated on a small slider was developed by Stanford University AI Lab between 1973 and 1981 [32]. It used feature extraction and image correlation for researching navigation and obstacle avoidance, though achieved mediocre results with slow movements taking up to fifteen minutes per meter [31]. However, it can be attributed with the first attempt at map-building [32]. Beyond these early attempts at navigation, it is difficult to pinpoint exact development in autonomous navigation in brevity without considering the research of various universities and government agencies.

For the discussed robots, the most common approach to navigation is a map-building approach. In regards to their applications, each of the industrial and home service robots is deployed in environments with extreme variance amongst them and so no pre-built map could be appropriate for all scenarios. Additionally, these robots could not operate completely without a map as each includes a general return-to-home feature that would require some capacity for remembering previous environmental observations.

With the discussed commercialized autonomous mobile robots, it is expected that none have released technical articles reviewing their methods for navigation. However, a glimpse into the inner workings of these systems can be had with reviewing current standards for navigation in open-source robotics. The most important abilities needed for navigation are locomotion, mapping, and localization. Locomotion is platform dependent though mapping and localization can be generalized across many platform applications. The navigation controller of any robot must be able to provide these capabilities or utilize underlying systems. Luckily there has been a considerable contribution to mapping and localization in the robotics community, with an overwhelming majority focus on SLAM which is discussed in Section 2.2.2.

With regards to open-source solutions for navigation, the ROS framework defines a basic navigational stack that can be implemented to support basic autonomous navigation and obstacle avoidance. This setup requires the previously mentioned capabilities of locomotion, mapping, and localization to enable navigation in the environment. Assuming the ROS suggested setup, the core of the navigation stack is a ROS library, Move_Base. This package employs the use of a global and local planner to define optimal paths for reaching a navigational goal [33]. These global and local planners use a known map of the environment, a set of tuning parameters, and persistent localization updates to build a path in the environment. The global known map is used to build an occupancy map that represents a weighted cost of traveling through a particular region and aids in planning routes. This navigation controller is discussed further in Section 5.2.1.

### 2.2.2 Simulataneous Localization and Mapping

Early research efforts towards navigation and obstacle avoidance with the Shakey robot from DARPA and the Stanford Cart project laid a foundation for research into feature extraction from video and images [31]. Since, vision-based navigation has seen substantial development of systems with majority efforts focused on either indoor or outdoor navigation. Both environmental types can be navigated with map-building based approaches, as well as some requiring no map. Indoor environments have the advantage of a mostly static floorplan and layout of landmarks, thus alternative methods also developed that supplied the robot with known environmental models [32]. However, this section focuses on perhaps the most predominant approach for mapping and localization, Simultaneous Localization and Mapping (SLAM).

SLAM is a generalized term for the process of sensing an environment and localizing a target object within that environment at the same time. It has become a fundamental challenge for the field of robotics as it is centric to many robot navigational systems. At the core, SLAM simply attempts to estimate the position, or state, of the robot and landmarks in the environment calculated by the observations made and the model of robot kinematics. This requires an ability to collect environmental samples, most often distance readings of some sort, and understand how the robot moves.

With sensors and actuators introducing noise and uncertainty to data collected, sensor readings cannot be deemed 100% reliable nor can robot movement be expected to directly translate to movement commands. Since the 1990's the most dominating approaches to addressing these issues have been probabilistic to enable reliable readings to be gathered. The primary approaches are Kalman Filtering, Particle Filtering, and Expectation Maximization [34]. Though the mechanics of SLAM are beyond the scope of this work, these approaches attempt to model both the noise of sensors and errors introduced by motors while moving or transitioning state [34]. Each of these approaches produces dependable data and state estimations through probablistically modeling uncertainty in prediction and update phases in order to filter it out.

SLAM does not discriminate against differing input sensor types, for the internal model of observations is aggregated after filtering sensor readings with the appropriate models of error for that sensor. Thus, SLAM has been implemented for many types of rangefinding sensors and cameras alike.

**Vision-Based SLAM**

Vision-based approaches were the first attempted by the robotics community to achieve autonomous navigation. With failed or partially successful attempts, some justifications could be placed on the lack of foundation in the realm of mapping and localization. With various approaches for SLAM, the data for vision-based systems is given a new opportunity to aid in mapping and localization. However, vision-based systems must rely on specialized sensors or a setup of multiple cameras in order to extract the most meaningful information for SLAM, depth readings.

One approach for gathering depth from a vision system utilizes the depth perception of the human visual system as motivation. With two camera sensors separated by a known translation, corresponding points can be found in the produced images. The disparities of these corresponding points enable a depth to be attributed to these points. Disparity is defined as the "difference in image location of the same 3D point when projected under the perspective of two different cameras" [35]. Triangulation can be applied to the set of disparities and depth can be retrieved for the set of three-dimensional points captured by both camera sensors [36]. This approach for depth requires prior knowledge of constants imposed by the camera's calibration in addition to effective methods of finding corresponding points [35].

Today there is a wide availability of specialized vision systems providing stereo cameras as well as other depth enabling components. For example, the well-known XBOX manufacturer, Microsoft, developed a low-cost vision system for sensing the movements of players in three dimensions for game interaction, the Microsoft Kinect. The first generation sensor employed an RGB camera and a depth sensor. The depth sensor consisted of an infrared projector and monochrome CMOS sensor that projects and detects a set of projected infrared points [37].

With the wide availability of low-cost vision systems, there has been a significant effort in applying SLAM to the depth information, typically a point cloud, generated by these systems. One such effort is that of Huang et. al [38] that describes a system for visual odometry and SLAM mapping using an RGB-D camera and makes the argument for the rich data from the sensors to be a viable alternative to other SLAM data sources. Another such work by Endres et. al presents an RBG-D SLAM method with the discussed Microsft Kinect sensor. Initial results supported trajectory estimations with a 2.1cm to 4.1cm root mean square error from ground truth [39].

The open-source community has also made attempts to progress this front. The RTAB-Map project of [40] and [41] by Labbe and Michaund contribute to the ROS community with support for online appearance based SLAM with loop closures. In this approach, robot locations are represented by an image signature, time index, and a weight. Neighboring locations are linked and additional links are created by loop closure. Extensions to RTAB-Map were made in [41] to allow for laser-based SLAM to complement RGB-D SLAM in environments not conducive for visual mapping. RTAB-Map is discussed further in Section 5.1.3.

**Laser-Based SLAM**

An alternative to extracting environmental features with video and image processing appeared with the introduction of low cost, high precision ultrasonic, infrared, and laser rangefinders. Though most of the robots discussed in this work utilize an array of available sensors, the most commonly implemented is LIDAR. Light Detection and Ranging (LIDAR) has become an established method of collecting precise spatial data of the environment in the field of view of the sensor. This spatial data was initially purposed for mapping particles in the atmosphere from a ground-based platform, though was later utilized for mobile platforms as developments in GPS allowed accurate transformations of sensor and global frames of reference [42]. In the applications and robots considered in this work and many other platforms, the host environments are not suitable for use of localization with GPS as the facilities often obstruct accurate readings or applications require finer-tuned positioning than what GPS can offer. Thus, developments were made to utilize the high-density outputs of LIDAR sensors for SLAM to manage robot localization.

LIDAR sensors vary widely in achievable accuracies, fields of view, and scan rate. However, each produces a stream of data points each corresponding to a distance reading that can be attributed to a particular point in one, two, or three-dimensional space. The cardinality is determined by the technology and form factor of the sensor. One dimensional sensors rely on a single beam of a laser pulse to retrieve a single distance reading or point. A one dimensional LIDAR that is rotated about one axis or a specialized sensor with a 360° field of view can produce two-dimensional point maps or grids. More advanced LIDARs implement further specialized sensors that are able to measure three-dimensional space with static or moving sensors and output point clouds.

Though it is difficult to understand the state of the art of SLAM with a review of commercialized robots, there have been substantial strides in open-source libraries for handling this task. To this community, one well-known library is GMapping. The approach improves basic grid mapping with the additional consideration of the most recent observations to complement the robot movement [43]. With considerable improvements in reducing uncertainty with the prediction phase of state, or pose, estimation and heavy integrations with ROS, this library became a standard for the mapping and localization features of the navigational stack of ROS itself [44, 45]. However, the rapid deployment of ROS updates has forced this library to lose active maintainers.

A current, at the time of this writing, and actively maintained SLAM library is Google Cartographer. Its system provides real-time SLAM in both two and three dimensions across multiple hosting platforms and sensor configurations. Though RGB-D SLAM is supported in theory, Cartographer has been optimized for the noise models of laser rangefinding sensors. In addition to real-time mapping, this library provides the ability for loop closures and support for multiple agents. That is, it allows for a memory of visited space to which multiple sensing platforms can contribute. The developers and open-source community have enhanced the library significantly, even to include features such as the removal of requiring a known model of kinematics for the hosting platform. This library is freely available to download and use and furthers its support for open-source robotics with integration to ROS [46, 47]. Google Cartographer is discussed further in Section 5.1.2.

# Chapter 3

# Arlo Demonstration Robot

For this work, the goal was to evaluate the current methods and components that can be found in commercialized ground robots and construct a platform that would be suitable for demonstrations of these methods and components. This chapter discusses in detail the selection and implementation of the platform itself. The Arlo Demonstration Robot (ADR) prototype implementation was done in stages that clearly organizes the developments at all levels of abstraction. This work chose to take a bottom-up approach, with the first two foundational layers tackled being the physical robot platform and the primary control system. These are highlighted in Figure 3.1 alongside the other developmental goals for later chapters. It was vital that these stages be given proper attention as the replacement of any systems at this level would disrupt the high-level systems. Building upon the result of this chapter, the selection and implementation of methods and components chosen for evaluation and demonstration with ROS are discussed in Chapter 4.

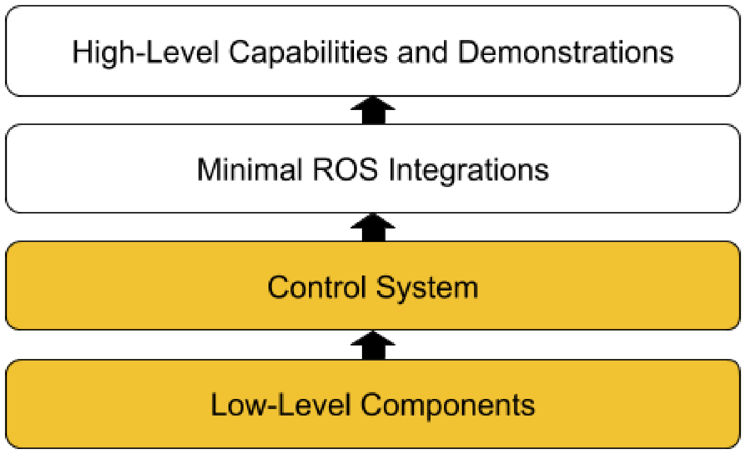


Figure 3.1: Chapter Three Development Goals for the Arlo Demonstration Robot

## 3.1 Robot Platform

In order to accurately be classified as a prototype platform, the robot must allow for a fair amount of flexibility in regards to applicable use-cases, additional resources for unforeseen features, and physical space for varying payloads. No specific payload constraints were imposed on the platform other than locomotion-critical components. As the focus of the review of commercialized robots was on wheeled-platforms, this work intended to use a ground-based wheeled robot. These loose provisions were revised to a set of requirements in order to shrink the search space of possible robot platforms and allow meaningful comparisons to be drawn between the considered options.

### 3.1.1 Requirements

The following requirements were imposed to narrow the search space. The robot platform must:

1. Have an appropriate form factor
2. Include essential hardware components for basic locomotion
3. Enable access to odometry information
4. Provide mounting surfaces for peripheral components
5. Provide an accessible power supply for peripheral components

The robot's form factor must allow for easy transportation as well as enable demonstrations in environments ranging from small conference rooms to entire facilities. In order to allow for easy maintenance and replacement of parts, it was determined the drivetrain and motor components ought to be included with the basic platform. This typically enables the use of forums and manufacturer support for any issues that may arise. Another motivation for only considering platforms with an included drive control system and justification for requirement 3 was to reduce the amount of error introduced to the platform during the development of the basic physical platform. Robot platforms with included odometry can usually ensure a reasonable amount of testing has been done during development. As the precision of high-level processing can be affected by even minute sources of error, it was imperative to consider this in choosing a platform and before any development began. Requirements 4 and 5 were enacted to further maximize the flexibility of what can be effectively hosted on this platform.

### 3.1.2 Considered Platforms

Even with the set constraints on the search space, there were many viable options all serving the purpose of generic development platforms and suitable for the needs of this work. The comparison of robots in Table 3.1 details four of the most attractive options of those considered. Each of these platforms provides the required basic drivetrain and control system and all are non-holonomic wheeled-robots with differential drive systems.

Table 3.1: Considerd Ground Robot Platforms

| **Platform** | **TurtleBot3 Burger** | **TurtleBot3 Waffle Pi** | **iRobot Create2** | **Parallax Arlo** |
|---|---|---|---|---|
| **Dimensions** | 138mm x 178mm x 192mm | 281mm x 306mm x 141mm | 340mm x 92mm | 457mm x 254mm |
| **Motor Controller** | DYNAMIXEL (XL430-W250-T)<br>Built-in with motors | DYNAMIXEL (XL430-W250-T)<br>Built-in with motors | Proprietary Built-in | Parallax DHB-10 |
| **Odometry** | Magnetic Encoders | Magnetic Encoders | Magnetic Encoders | Optical Encoders |
| **Mounting Surfaces** | Small Tiered Waffle Plates x3 | Large Tiered Waffle Plates x3 | Top surface | Tiered Round Plates x2 |
| **Power Source** | 11.8V 1800mAh LiPo | 11.8V 1800mAh LiPo | 14.4V 3800mAh | 12V 7.0Ah SLAs x2 |
| **Power Supply** | 12V 5A | 12V 5A | N/A | 12V 20A |
| **Aux Power Source** | 3.3V 1800mA<br>5V 4A<br>12V 1A | 3.3V 1800mA<br>5V 4A<br>12V 1A | N/A | 5V 2A x2<br>6.5V 2A x2<br>12V 4A x4 |
| **Drive System** | Differential Drive | Differential Drive | Differential Drive | Differential Drive |
| **Price** | $549 | $1399 | $199 | $999 |

Anyone familiar with ROS is most likely familiar with the TurtleBot series of robots, as it is a ROS standard platform robot. The TurtleBot3 was developed to reduce the size of previous versions and lower the price while maintaining functionality and quality [48]. It has a unique offering for expandability as it is built with a set of tiered waffle plates. With three models of the TurtleBot3, two of the actively manufactured models were considered as this platform has the advantage of extensive documentation and ROS community support. The Burger offers the smallest overall footprint with small waffle plates and the Waffle Pi model offering a much more spacious plate system for its tiers. Both of these platforms supply a Raspberry Pi 3 as the control system and include an additional laser scanner, inertial measurement unit, and set of GPIO expansion pins. Actuators consist of a speaker, set of status and programmable LEDs, and set of buttons and dip switches.

Early versions of the Roomba vacuum robot, as discussed in Section 2.1.2, were the inspiration for many projects that attempted to hack the control system, often ending with the unintended destruction of the platforms. With numerous publications written to that regard, iRobot released the Create series of robots to satisfy the apparent need for such a platform. The Create2 model is based on a popular model of the small home helpers, the Roomba 600 series. The Create2 exposes the iRobot control system through a simple serial connection. This enables commands to be issued for controlling actuators and retrieving updates from onboard sensors. The included actuators consist of the wheel motors, vacuum brush motor, vacuum motor, speakers, LEDs, and buttons. It also includes infrared receivers, four cliff sensors, bump switches, and wheel drop sensors [49].

Ultimately, it was decided to not use the TurtleBot series or Create2 robot. As to the reasoning behind their dismissal, the Create2 poses a number of constraints with the pre-assembled placements of components and simply does not offer enough flexibility for additional payloads. The TurtleBot series of robots offer flexibility with the tiered structures, though the Burger could suffer from a raised center of gravity with additional payload tiers. Also, both the Burger and Waffle Pi don't offer much space between their tiers. Targetting demonstrations of robot abilities at higher-levels than basic locomotion can require any number of peripherals to be added to the platform each varying in size. Additionally, the TurtleBots offer poor battery capacity at 1800mAh. Considering the Roomba vacuum can require a charge before it accomplishes cleaning in a large home, it can be said that the 3800mAh offering of the Create2 is also a disadvantage. Payloads requiring power for additional motors or other power hungry components would not be well suited on these small capacities.

### 3.1.3 Arlo Complete Robotic System

The Arlo Complete Robotic System by Parallax provides a solution to the insufficiencies presented in the previous section.

**Overview**

The Arlo Robot developed by Parallax is a complete unassembled kit that ships with all of the mandatory mechanical and electrical components needed to assemble the robot platform and supply the first command to move. Its round and symmetrical physical footprint places the wheels at the sides of the robot and establishes the center of rotation at the center of the robot, providing many benefits for maneuverability. It measures approximately 18in in diameter and 10in tall in the default configuration and rides on 6in pneumatic tires to ensure traction in various terrains. The robot has a tiered structure with a battery bay tucked underneath the primary platform and large 5in standoffs to elevate the second platform. The kit ships the materials for the default configuration of two platforms, though more platforms may be added according to the application requirements. The primary platform provides mounting holes for the provided control board, motor controller, and power distribution board leaving ample space for any peripheral components.

The kit supplies two 12V motors along with appropriate mounting hardware for attaching the axle, wheels/tires, and provided encoders between the battery plate and primary platform. The encoders are 36-position disk Quadrature Encoders utilizing two onboard sensors for 144 ticks per revolution. The mounting hardware accommodates the encoders in a machined recess to ensure the encoder disk is aligned properly and reduce possible encoder readings. The optical encoders are beneficial due to their inherent resistance to magnetic fields disrupting accurate position readings, which prevents undesired environmental constraints from being imposed.

To provide commands to the motor controller, the Arlo kit comes with a Propeller ActivityBoard WX. This board gives access to a Propeller microcontroller, RF module, a small breadboard, three pin headers, A/D and D/A converters, and a microSD card holder. This board is capable of many standard control functions and applications, though it is not suitable for the work of this thesis. In addition, it was decided preceding this thesis, to replace the central control board with a platform that supports a much wider range of potential applications. Thus, there were no requirements set for selecting an appropriate robot platform as listed in Section 3.1.1.

The kit also ships with a set of four ultrasonic Parallax Ping))) sensors. These are disregarded as obstacle avoidance was intended to be accomplished with more sophisticated sensors. The Arlo default configuration can be seen in Figures 3.2 and 3.3.

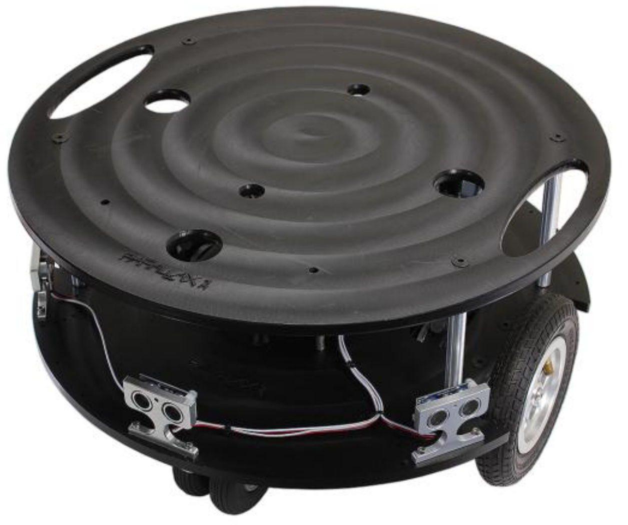


Figure 3.2: Arlo Robot Default Confiuration with Top Plate, Source: Parallax

### Power

The Arlo kit comes supplied with two 12V 7.0Ah sealed lead acid (SLA) batteries for supplying power to the whole system. This is nearly eight times the capacity of the TurtleBot robots and almost quadruple the capacity of the Create2 robot. Additionally, this enables support for higher current draws and more options for power conversion. Parallax claims this platform, in its default configuration, will consume 1.5A - 3A depending on the terrain [50]. With this rating, the Arlo could sustain operation at full speed on the worst terrain

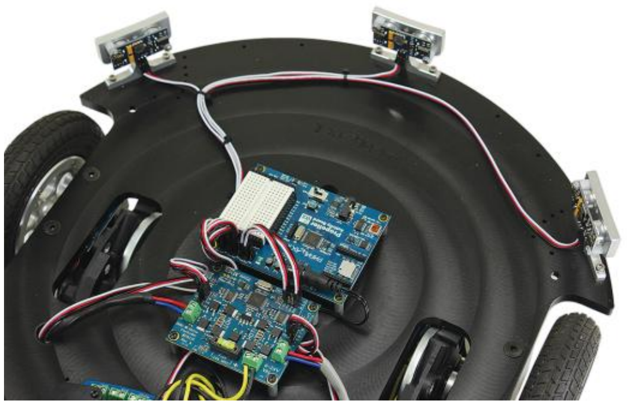


Figure 3.3: Arlo Robot Default Confiuration without Top Plate, Source: Parallax

for approximately two hours and twenty minutes. The schematic of components is shown in the standard system layout in Figure 3.4.

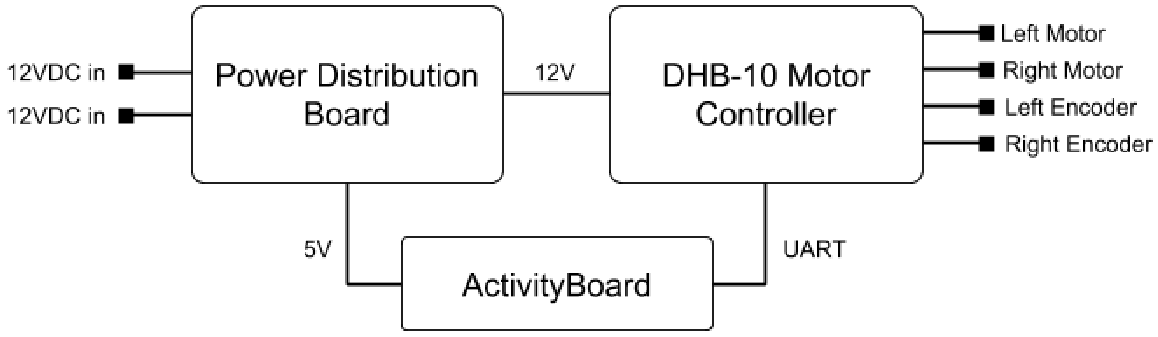


Figure 3.4: Schematic for Default Configuration of Arlo Compelete Robotic System

The included power distribution board (PDB) is shipped unassembled but provides an array of output power supply options. This includes a set of ten terminals for electrical connections, two 12V battery inputs, four 12V outputs, two 6.5V outputs, and two 5V outputs. Two of the 12V outputs are labeled for a motor controller connection and have a secondary in-line switch that must be in the on-position to supply power to the motors and motor controller. The primary switch manages system power with the exception of the motor controller terminals. The available PDB terminal connections are listed in Table 3.2. A charging terminal is also available on the PDB for connecting an appropriate 120VAC 3A battery charger, which the kit ships with. The circuitry for properly charging the SLA batteries is included in the PDB.

Table 3.2: Arlo Power Distribution Board Terminals

| **Direction** | **Voltage** | **Total Fuse Protection** | **Terminal Count** | **Type** |
|---|---|---|---|---|
| Input | 12V | 20A | 2 | Battery |
| Output | 12V | 15A | 2 | Motor Controller |
| Output | 12V | 4A | 2 | Auxiliary |
| Output | 6.5V | 2A | 2 | Auxiliary |
| Output | 5V | 2A | 2 | Auxiliary |

**Motor Controller**

The Arlo Complete Robotic System utilizes the DHB-10 Dual H-bridge Motor Controller with two motor control channels rated at 10A each. It requires an input voltage in the range of 6-24VDC, though the Arlo PDB suggests the use of 12V PDB terminals as the supply can be switched separately from main power. At the core of the controller is a Propeller P8X32A microcontroller which is preloaded with open source firmware customized for the Arlo platform configuration.

The Parallax DHB-10 Firmware Guide [51] describes in detail all of the movement commands, information requests, communication modes, and constants used in the Arlo configured firmware. All commands and requests must be issued via a serial connection. Summaries of relevant commands and requests available in this guide are available in Table 3.3 and Table 3.4, respectively. Section 4.3 elaborates on the use of these commands for the software interface for the motor controller.

## 3.2 Central Control Board

It was decided, preceding the selection of a physical platform, to replace the central control board of the chosen robot with one that supports a much wider range of potential applications. Even so, the Arlo Robotic System does not provide a suitable control board considering the ActivityBoard is centered around a Propellor microcontroller. In order to find a suitable replacement, a set of requirements were enacted to allow comparisons to be

Table 3.3: DHB-10 Motor Controller Command Subset, Sourced from: [51]

| Commands | Description | Parameters | Parameter Description |
|---|---|---|---|
| **TURN** | Turn in place, rotating by motor positions | Total Motor Movement | Positions from -32767 to 32767 |
| | | Top Speed | Positions per second from 1 to 512 |
| **ARC** | Travel along the arc of a circle | Radius | Positions from 1 to 32767 |
| | | Top Speed | Positions per second from 1 to 32767 |
| | | Arc Angle | Degrees from -32767 to 32767 |
| **GO** | Set and hold the motor output power | Left Wheel Power | 1/127th of total power from -127 to 127 |
| | | Right Wheel Power | 1/127th of total available power from -127 to 127 |
| **GOSPD** | Accelerate and sustain a speed | Left Wheel Speed | Positions per second from -32767 to 32767 |
| | | Right Wheel Speed | Positions per second from -32767 to 32767 |
| **MOVE** | Accelerate, travel, and decelerate across a distance in positions | Left Wheel Distance | Positions from -32767 to 32767 |
| | | Right Wheel Distance | Positions from -32767 to 32767 |
| | | Speed | Positions per second from 1 to 32767 |
| **TXPIN** | Set and switch terminal transmit pin | Transmit Pin | Either CH1, CH2, or PROG, Default: CH1 |
| **BAUD** | Set and switch terminal baud rate | Baud Rate | Baud from 19200 to 115200, Default: 19200 |

Table 3.4: DHB-10 Motor Controller Request Subset, Sourced from: [51]

| Requests | Description | Returns | Return Description |
|---|---|---|---|
| **SPD** | Report current motor speeds | Left Wheel Speed | Positions per second from -32767 to 32767 |
| | | Right Wheel Speed | Positions per second from -32767 to 32767 |
| **HEAD** | Report current heading | Heading | Degrees from 0 to 360 |
| **DIST** | Report accumulative distance each motor has traveled | Left Wheel Distance | Positions from -32767 to 32767 |
| | | Right Wheel Distance | Positions from -32767 to 32767 |
| **RST** | Reset wheel travel distance to zero | N/A | N/A |

drawn between the considered control boards.

### 3.2.1 Requirements

The following constraints were placed when selecting a replacement central control unit for the prototype platform and are listed in order of importance. The board must:

1. Be compatible with available Arlo PDB power supplies
2. Enable the use of the latest release of ROS (Melodic Morena)
3. Enable the use of TCP/IP network communication
4. Enable the use of a standardly available camera
5. Enable the use of USB peripherals
6. Enable the use of Serial/UART peripherals
7. Enable the use of I2C peripherals
8. Enable the use of GPIO peripherals
9. Provide flexibility for unforeseen functional requirements

The central control board must be compatible with the chosen Arlo robot power supplies. With Arlo's numerous available voltages, this requirement should be easily met. Perhaps the most restricting requirement is the facilitation of the latest release of the Robotic Operating System. As ROS is the most commonly used robotics platform among students and developers alike, as well as this work's goal of contributing to the state of the art, its support is essential for a control board. The latest official release of ROS is supported on AMD64, ARMHF, and ARM64 architectures running Ubuntu or Debian Linux. Thus the control board ought to employ one of the mentioned architectures. Another requirement imposed by ROS is the availability of a TCP/IP network stack.

The remaining requirements were put in place to expand the available interfacing options of the control system and ensure its suitability for various demonstratable functions. With a desire for remote monitoring, camera support must be enabled via specialized ports or a typical USB interface. With the availability of USB, UART, I2C, and GPIO controllers, nearly any other sensor or peripheral would be able to interface with the control board. Additionally, the flexibility of boards with specialized components would contribute to the ability to serve multiple applicable use-cases and better manage unforeseen functional requirements.

### 3.2.2 Considered Platforms

The search space for control boards focused on small-scale development boards with minimal footprints. With an abundance of boards in the category of SoC centric single board computers, this search space was extremely large. With a simple survey of equipment in labs at the University of Arkansas and the suggestions of online forums, the three most popular boards used in robotics projects and other general uses were compiled and compared in Table 3.5.

The Raspberry Pi series is an extremely affordable set of single board computers that have seen a large surge in popularity since their introduction in 2012. In 2017, it achieved the status of the world's third best-selling general purpose computer in just five years, as reported by MagPi, Raspberry Pi's official magazine [52]. Just as the other boards considered, the Pi is centered around an ARM-based SoC. With the flagship model, the Raspberry Pi 3 Model B+, it provides a basic set of peripherals for development purposes, including four USB ports along with other general inputs and output [53]. This board is quite comparable to others with similar features, though the Pi's advantages come from the unprecedented community support and documentation targeted for education.

The DE0 Nano SoC is another capable development board, though it also exposes the hardware development domain with the Altera SoC FPGA. It offers a more limited set of input and output ports though it has double the number of GPIO pins available on the Raspberry Pi boards [54]. A limited set of reference designs, tools, and documentation is available for both hardware or software focused development with this board. The board manufacturer, Terasic, also offers robotic platforms utilizing the DE0 as central control. Though they did not meet the platform requirements of this work, these resources can easily be applied towards generic development on the DE0.

Table 3.5: Considered SoC Development Boards

| **Board** | **Raspberry Pi B+** | **DE0-Nano-SoC** | **Zybo Z7-10** |
|---|---|---|---|
| **Recommended Power Supply** | 5V 2.5A | 5V 2.5A | 5V 2.5A |
| **Ethernet** | Gigabit Ethernet | Gigabit Ethernet | Gigabit Ethernet |
| **WiFi** | 2.4GHz, 5.0GHz | No | No |
| **CSI Camera Port** | Yes | No | Yes |
| **USB** | Four USB 2.0 | One USB 2.0, One USB OTG | One USB 2.0, One USB OTG |
| **UART** | One dedicated controller via GPIO | One dedicated controller via USB, or custom controller via PL+GPIO | One dedicated controller via USB, two dedicated controllers via PL+GPIO, or custom controller via PL+GPIO |
| **I2C** | One dedicated controller via GPIO | One dedicated controller via PL+GPIO, or custom controller via PL+GPIO | Two dedicated controllers via PL+GPIO, or custom controller via PL+GPIO |
| **GPIO** | One 40-pin GPIO Header | Two 40-pin GPIO Headers | Five PMOD Ports |
| **Processor** | 1.4GHz Quad-core Cortex-A53 (ARMv8), 64-bit | 925MHz Dual-core ARM Cortex-A9, 32-bit | 667MHz Dual-core ARM Cortex-A9, 32-bit |
| **RAM** | 1GB DDR3 | 1GB DDR3 | 1GB DDR3 |
| **FPGA** | N/A | Altera Cyclone V SE 5CSEMA4U23C6N | Xilinx XC7Z010-1CLG400C |
| **SD Card Boot** | Yes | Yes | Yes |
| **CSI Camera Port** | Yes | No | Yes |
| **Price** | $35 | $99 | $99 |

With all of the considered boards meeting the set requirements and providing affordable small-scale development platforms, the flexibility factor determined the selected board. With almost identical specifications, the DE0 allows more flexibility over the Pi with the inclusion of programmable logic (PL). However, the Digilent Zybo Z7 provides further flexibility with the inclusion of programmable logic and numerous dedicated hardware interface controllers for IO. The next section discusses this development board in more detail as well as presenting the implemented hardware design and operating system on which the robot platform is built.

### 3.2.3 Zybo Z7-10

The Zybo Z7 development board, manufactured by Digilent, hosts a Xilinx Zynq-7000 series SoC and numerous input and output peripherals. The Zynq family of SoCs is based on the Xilinx All Programmable System-on-Chip (AP SoC) architecture, which tightly integrates a dual-core ARM Cortex-A9 processor with Xilinx 7-series Field Programmable Gate Array (FPGA) logic [55]. The Zybo Z7 supports two configuration variants with either the XC7Z010-1CLG400C or XC7Z020-1CLG400C SoC chips, differing primarily in FPGA size. The variants are referred to as the Zybo Z7-10 and Zybo Z7-20, respectively. For the purpose of this work, the Zybo Z7-10 was selected to host the control system processes for the robot platform.

To enable exploration in both hardware and software domains, a baseline hardware reference design was implemented and used in the configuration of an embedded Linux deployment on the Zynq processor.

#### Baseline Hardware Design

As the Zybo hosts a Xilinx SoC chip, development software from the Xilinx collection of design tools was required. In order to ensure this work is able to maximize its contribution to future research efforts, the latest available tools are utilized. The Vivado Design Suite - HLx Edition - 2018.3 was the most recent release at the time of this work and thus was utilized for developing the hardware design to be implemented in PL.

The block design developed for this platform, seen in Figure 3.5, is simple but serves as an adequate baseline design that can be extended with applications requiring strict timing constraints and parallel processing. The block design can be seen to include a Zynq processing system IP that is available as a standard IP from Xilinx. The ZynqPS IP has a number of internal configuration options to customize the processing system according to the needs of the application. A single serial port, UART0, is enabled by default with the ZynqPS to serve as a console interface with the Zynq processor. With the requirement for an implementation of a motor controller interface, an additional serial port with a dedicated controller was enabled for the ZynqPS. The controller for this serial port, UART1, allows the alteration of the serial protocol options from software allowing different baudrate support. The UART1 controller was attached to a set of specialized MMIO ports, one for transmission and one for receiving, exposed on a PMOD connector of the Zybo. For reasons discussed in Section 4.3, an additional UART controller was added in the PL, the AXI-UARTLite IP available from Xilinx. This serial controller adds a third port, UART2, that exposes its transmission and receiver ports on another PMOD connector. This IP does not support dynamic baudrate as the rate is fixed in hardware and was set to a baudrate of 19200, the fastest speed compatible with the Arlo DHB-10 motor controller.

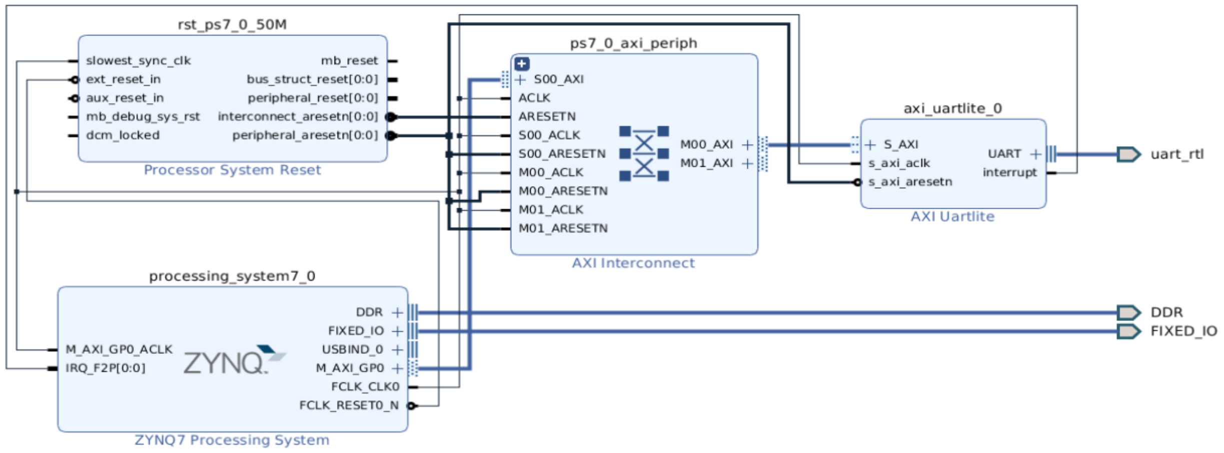


Figure 3.5: Zybo Z7-10 Hardware Block Design for ADR Platform

Upon the completion of the baseline block design, Vivado was used to synthesize the design and create a bitstream for the Zybo Z7-10 development board. The bitstream and generated hardware description file were exported for use in proceeding steps.

**Embedded Linux**

As it is necessary that the Zybo host an Ubuntu or Debian Linux operating system, the Zybo must be set up to configure hardware and boot Linux from an SD card. As implementing an embedded Linux platform has become more common in the hardware development community, many of the primary FPGA manufacturers also distribute tools for streamlining the process. Xilinx provides the PetaLinux Tools that offer the necessities for customizing, building, and deploying Linux on Xilinx systems [56]. Integrations with Xilinx hardware design tools simplify the process of exposing the hardware design in PL to the boot loader, kernel, and software applications.

PetaLinux first requires an input of a hardware platform description generated with Xilinx Vivado. This description includes details such as custom IP memory addresses, hardware system configuration parameters, and processor configuration parameters [57]. With this input, PetaLinux is able to accurately inform a Linux kernel of all the available hardware components available to it. The next steps take configuration options for setting up the kernel itself as well as filesystem and system-level options. Kernel customization involves the selection of kernel modules that should be included with it, thereby expanding or limiting the capabilities of the core Linux processes. A Linux operating system could be considered as a particular organization and collection of system packages that provide a set of functionalities to enhance usability and further abstract low-level processes. Thus the Linux root filesystem is the source of many of the high-level processes that together define the operating system. PetaLinux utilizes an open-source collaboration project, Yocto, for building and customizing the root filesystem. Yocto simply enables templates and methods for creating custom Linux-based systems for embedded application. The final options handled by PetaLinux are system-level configurations that specify parameters for changing boot behaviors, boot image locations, and image packaging. With the collection of customization specifications, PetaLinux will utilize these to compile a Linux kernel that can be packaged with a first stage bootloader and hardware bitstream into a bootable Linux image. Additionally, PetaLinux will automatically generate a root filesystem that can be mounted after the kernel has booted.

For the platform, there were not many customizations made to the Linux kernel. Replacing the control board on the default Arlo robotic platform setup, discussed in Section 3.1.3, required no specific kernel modules. However, the integrations explored in Chapter 4 introduced additional sensor payloads that did require addition kernel modules. To add these modules or any other common module, the PetaLinux kernel configuration must be run and the appropriate module selected. A rebuild of the kernel and regeneration of the boot image finalizes the kernel update and enables the new component drivers upon next boot. At the time of this writing, the latest release of the PetaLinux Tools was 2018.3. Additionally, the current default kernel version that PetalLinux supports is 4.14.0-xilinx-v2018.3. As the name suggests, this is a Xilinx custom version of the Linux kernel version 4.14.0.

The most versatile method for configuring and booting from the PetaLinux produced images is via an SD card, thus this was the approach taken by this work. The bootloader and Linux images are required to be stored on a FAT32 formatted partition of the SD card, while the root filesystem was to reside on an EXT3 or EXT4 formatted partition. A significant change introduced to the embedded Linux deployment for this platform was the replacement of the PetaLinux generated filesystem as it could not support ROS. Instead, the filesystem was overwritten with that of a ROS Melodic supporting Ubuntu Bionic 18.04.02.

# Chapter 4

# ROS Integrations

Continuing with the bottom-up approach, the development stage discussed in this chapter is highlighted in Figure 4.1. It introduces the Robotic Operating System (ROS) framework that serves as the foundation of all software processes and extends the bare physical platform to complete the proof of concept Arlo Demonstration Robot (ADR). The below sections discuss the framework itself, the minimal requirements for ROS integration of a general robot, as well as the implementation of some of the additional features found in the robots discussed in Chapter 2.

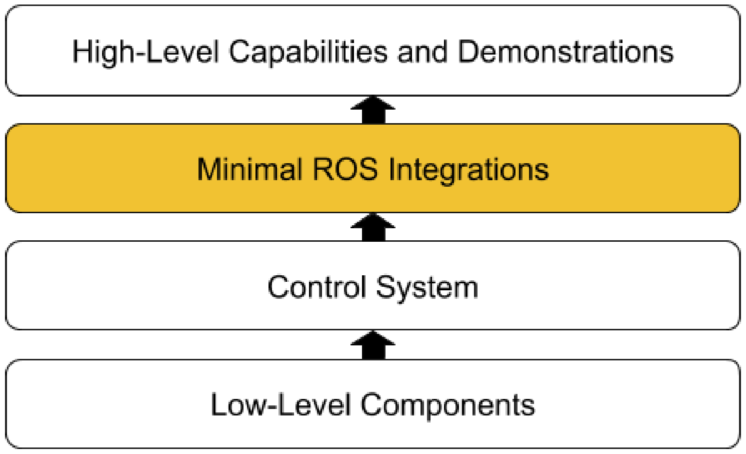


Figure 4.1: Chapter Four Development Goals for the Arlo Demonstration Robot

## 4.1 Robotic Operating System

The Robotic Operating Sytems was introduced by Quiqley et. al in the work of [58] in 2009. The authors put heavy emphasis on allowing large-scale integrative research in robotics that made it applicable for a variety of use-cases that may or may not require scalability. Development was fueled by a number of design criteria that set it apart from other existing frameworks of the time. These criteria included the enabling of peer to peer topologies communication policies, multi-lingual development, a partitioned tools-based development environment, separation of complex driver and algorithm libraries and core functionalities, and lastly a framework that is completely open-source and free of use restrictions. The peer-to-peer topology of ROS avoids the issue of unnecessary traffic over network links as it provides a *master* module to manage the lookup mechanisms or name services for processes requiring communication with another. This supports a central directory of information of various processes that are exposed by the ROS communication system and is known by each included process. Multi-lingual development ensures that developers can utilize their own preferences in programming languages and still be able to leverage the benefits of ROS. The specification of any ROS process is only at the messaging layer. That is, it defines an Interface Definition Language (IDL) that simply describes the messages that can be sent between various modules and the subfields of them. Additionally, message definitions can be composed to ease scalability and the addition of new message types. The microkernel design of ROS uses a fair amount of small tools as separate modules for the building and running of ROS components. This yields a large reduction in overhead of the ROS framework by enforcing complex drivers and algorithms to be developed as stand-alone libraries that can interface with ROS through the IDL specifications as ROS packages. Most notably, the final criteria set by the authors for a free and open-source framework promotes robotics community involvement for input on revisions at all levels of the software stack.

ROS establishes a set of concepts that warrant a review of the nomenclature it uses. Processes that are run and exposed to the ROS master node through an IDL specification are called **Nodes**. The term is taken from the graph-based method ROS utilizes with visualizing the collection of running processes. Nodes may communicate with one another by passing **Messages**, strictly typed data structures of standard primitive types. As mentioned above, these typed messages can also be nested to any depth. Any message that is sent must be published to a particular **Topic** to provide easy management of publishers and interested subscribers. Synchronous transactions require well-structured request and response types, thus ROS provides a **Service**. They are provided by a singular entity and defined by a string name and request and response message structure (Quigley 2009)[58]. With the foundation that the community and ROS developers have laid, it was the ideal choice for an underlying framework for all software processes of the Arlo Demonstration Robot.

### 4.1.1 Minimal ROS Functionalities

As discussed in the previous section, ROS only provides a communication template with which applications can use for designing simple or complex robot systems. With the flexibility of the framework and the variety of platforms supported, it is difficult to establish a bare minimum set of functionalities to fully enable a robot with ROS. Here, this work attempts to organize a set of minimalistic ROS functionalities for interfacing with the low-level aspects of the physical platform of the robot used. As this work primarily focused on replicating features seen in commercial mobile ground robots, this section limits its consideration for only this class of platform. The three basic functionalities that must be enabled with ROS are:

- Control of drive systems
- Retreival of odometry
- Publication of platform configuration specifications

With a mobile ground robot, it can be assumed there exists some method of mobility and a corresponding mechanism to send commands for control. Additionally, in order to have an awareness of how the robot reacts to any movement commands, some form of odometry is usually utilized. These two basic functionalities would be sufficient for nominal remote control of the platform, though they don't offer enough information for determining the position of the platform in the environment. For example, wheeled platforms typically utilize disk encoder that allows the rotation of the motor to be monitored. Direct readings from the encoder would not be sufficient in measuring the distance the platform has moved without an understanding of the physical platform characteristics. For this reason, a description of the robot platform is also needed for a bare minimum integration of ROS.

The lower levels of the ADR control system comprise of only a set of motors and encoders that are accessible through interfacing with the motor controller. In order to let any ROS node access these low-level components, a ROS abiding process to handle the interaction with the motor controller was needed. It must publish odometry information gathered from encoders for any interested subscribers and allow publishers to send commands to control the motors. To do so, encoder readings must be translated to meaningful values and motor control commands must be translated to values with standard measurement units. These translations are only possible if a know platform configuration is specified. The implementations of these basic functions for the ADR are detailed below.

## 4.2 Robot Description

To unlock the true potential of ROS and many other complex ROS packages, a model of the platform must be available. A fundamental concept of ROS and almost any other robotics framework is the notion of coordinate frames. A general robot description will include detailed measurements and positioning of onboard components so as to define the relationships among the included coordinate frames.

### 4.2.1 Transformations

In the work of Quigley et. al [58], the authors included the consideration for clearly defining spatial relationships and coordinate frames with the transformation library, or *TF* system. This system is utilized to continuously update a transformation tree, or TF tree, that establishes a relation between all frames of reference present in the system. This can include the relationship of the platform in regards to its environment or the placement of a wheel with respect to the robot body. The TF system helps to eliminate the errors that are common in manual transformation between frames. Transformations can be defined and published in a number of ways such as explicit definitions in a ROS package, dedicated ROS nodes for broadcasting static transformations, or dedicated description files that are discussed shortly. All transformations that are published must be assigned to the ROS topic, /tf.

To elaborate on the necessity of known spatial relationships with an example, the ground on which a robot sits will typically be a static coordinate frame as it does not tend to

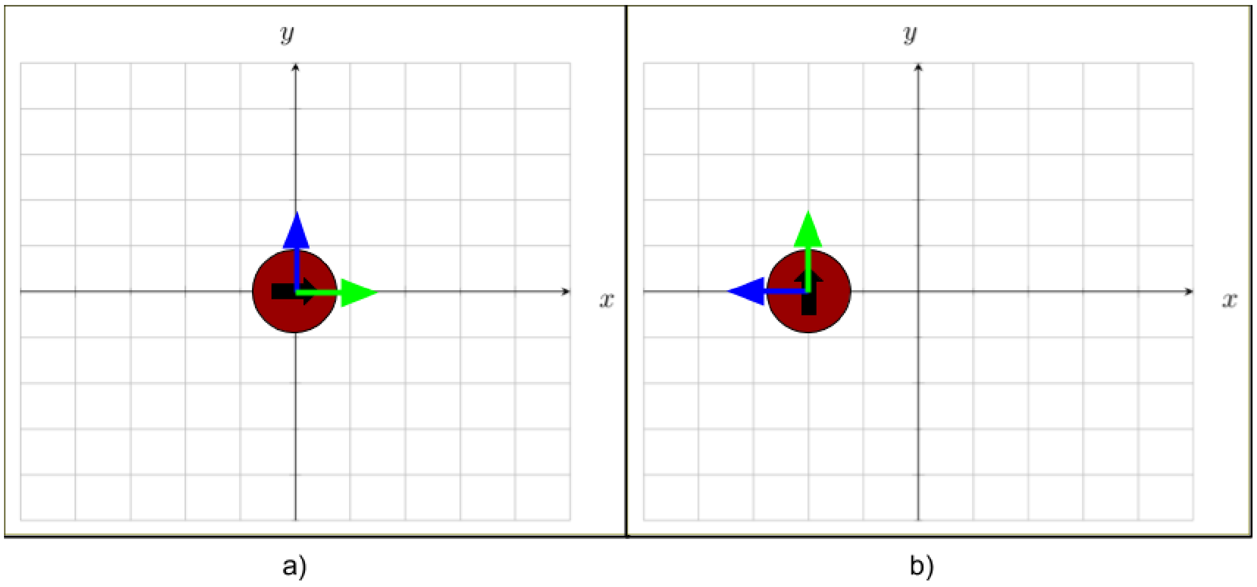


Figure 4.2: Example of Coordinate Frame Transformation

move. However, the robot, equipped with wheels and appropriate actuators, can move along the ground changing position and orientation with respect to the global coordinate frame. There must exist some known transformation between these two frames in order to accurately track the robot's movements in the global frame. The grids in Figure 4.2 demonstrates this scenario of how the robot's frame, marked by the red circle, can change with respect to the global frame, denoted by the grid. The robot's orientation is denoted with the green (positive X) and blue (positive Y) axes. In grid a), the robot is centered at the origin and its orientation in line with the ground frame, thus no transformation is needed to change between frames. In grid b), the robot has moved to position (-2,0) on the grid and has also turned 90° counterclockwise. Here, a translation and rotation must be applied to change between the global and robot frame.

In order to effectively apply transformations and quickly change between frames, ROS allows for a robot model or description to be given as a Unified Robot Description File (URDF) of links and joints. Links are physical structures of the robot such as the wheel, while joints denote connections or transformations between links. An example of both would be an axle joint making the connection of the wheel and robot frame links. This description must be in the form of a valid TF tree such that every child link must have only a single parent link. With a URDF file, joints can be static or dynamic. However, as this work does not employ any actuators that would supply meaningful visualizations, thus the axle joints are static in the ADR model. A ROS package, *robot_state_publisher*, takes as input a URDF file of a robot description and will manage the broadcast of all transformations available in the input. This package decreases the complexity of the transformation publishing task with a dedicated ROS node.

### 4.2.2 ADR Model

In order to satisfy the first basic ROS functionality, a robot model was developed and broadcasted with the robot_state_publisher package. The TF tree in Figure 4.3, generated by the rqt_tf_tree visualization tool, denotes the relationships of coordinate frames written in the robot description URDF file for the ADR platform. The URDF file for this platform was adopted from the work of [59] and simplified to explicitly define all links and joints. The root frame of the ADR TF tree is seen to be **base_footprint**, though this is only necessary for legacy ROS packages. Actually, **base_link** represents the primary platform of the robot. The child links represent the components that are directly attached to this platform such as the front and rear caster wheels. The *battery_box* link consolidates the components located underneath the primary platform as a three-dimensional volume. The branches extending from this link denote the axles and wheels on both sides of the robot. The remaining branches extending from the base_link are the standoffs, not shown here, and the top plate.

Utilizing the ROS visualization tool, RViz, the robot model can be rendered in 3D from the descriptions given in the URDF. This rendering can be seen in Figure 4.4. It can be noted that the rendering and the TF tree lack the Arlo system's power distribution board and motor controller as well as the Zybo primary control board. This is due to the fact that these served no purpose in the TF tree other than visualizing them. There was never going to exist a case where they required a frame of reference, as they simply don't provide any

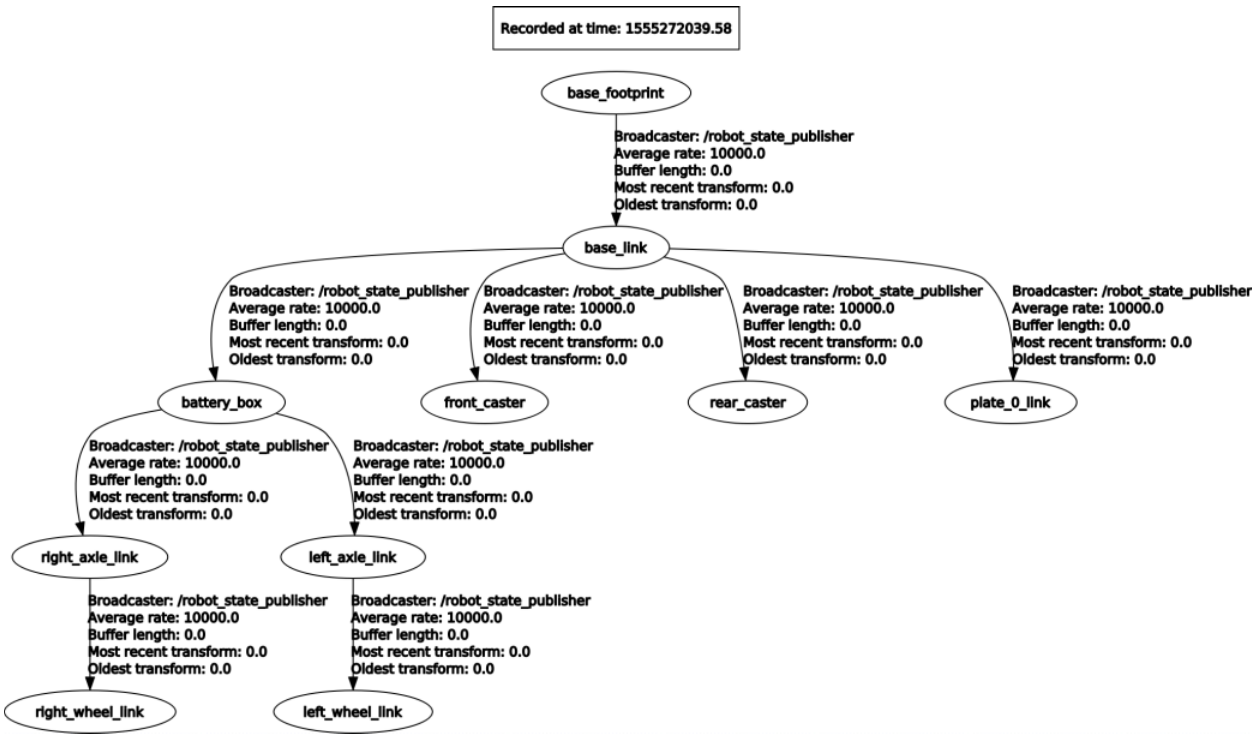


Figure 4.3: TF Tree for the Arlo Robot Model

sensory information that would require a transformation to another coordinate frame.

## 4.3 Base Controller

With ROS fully aware of the robot configuration, it must next be given access to the basic controls and sensing components available. As the platform chosen for the ADR includes a motor controller that directly interfaces with the wheel encoders and motors, it was appropriate to group the tasks of motor control and computing odometry into a single module, the *base_ controller*.

Commands can be issued to the motor controller over a serial interface, thus one of the serial ports exposed in Section 3.2.3 was to be assigned for this purpose. The DHB-10 motor controller by default enables its serial port on a single pin, CH1, and at the baudrate of 9600. However, it allows for the segregation of transmission (TX) and receiving (RX) pins by changing the assigned pins. The most obvious choice was the UART1 port as it allowed for a dynamic baudrate though utilizes separate pins for TX and RX functionalities. For reasons unbeknownst to the authors, initial attempts of interfacing with the motor

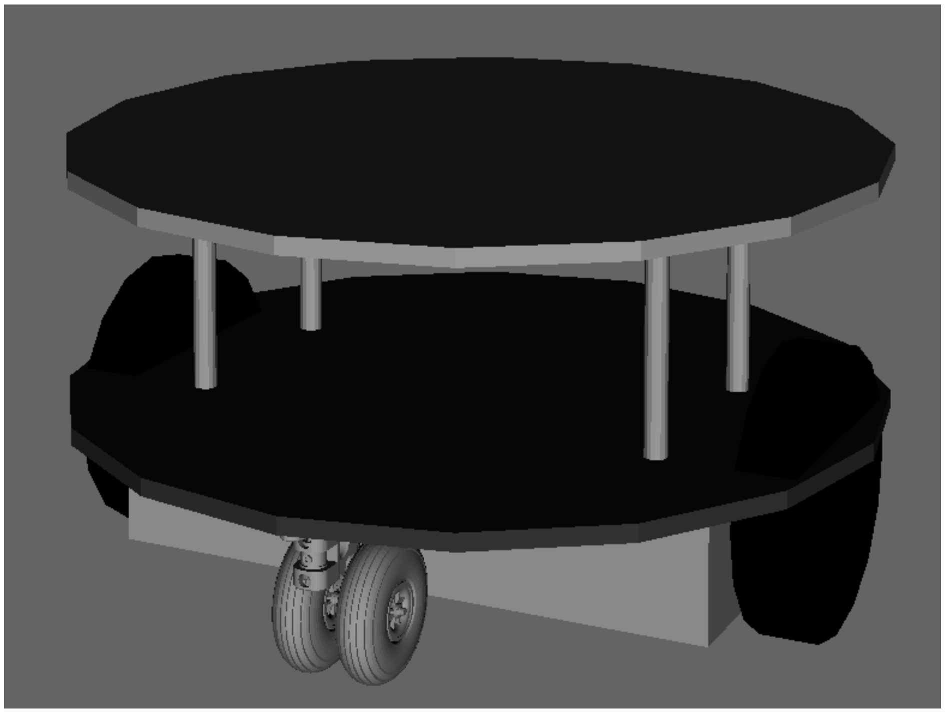


Figure 4.4: RViz Rendering of the ADR Base Platform

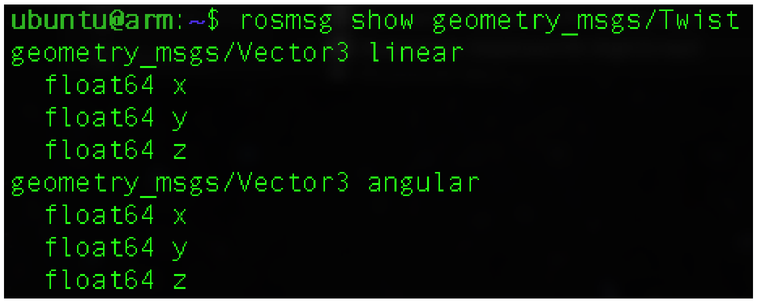
controller were restricted to issuing commands without response as the RX port of the UART1 controller did not register signals from the motor controller. To counteract this, an additional serial controller, UART2, was added to the Zybo hardware to ensure commands could be received from the motor controller board. However, an additional challenge arose with the consideration that UART2 was restricted to a static baudrate. To maximize the throughput of communications with the motor controller, it was decided to utilize the fastest supported baudrate of the controller, 19200, thus the UART2 port was set up for this in hardware. With a UART1 controller with variable baud and a single working TX pin as well as a fully-functioning UART2 with static baud, the enabling of the 19200 baudrate required the use of both UART1 and UART2 serial controllers.

The applicable commands for the motor controller's firmware are listed in Section 3.1.3. In order to configure the motor controller for the discussed serial interface, the controller is issued a command to separate the TX and RX serial pins, "TXPIN CH2". Secondly, the baud rate of the motor controller is adjusted from the default rate with "BAUD 19200". A

Linux program like Picocom or Screen can be used to open a serial console from the Zybo and send and receive these motor controller commands. Utilizing these methods do not however enable a ROS integration to be made. Instead, it was decided to implement a process capable of accessing the serial ports, handling commands and responses, and registering with ROS. Since ROS supports nodes written in many languages and there exists a well-established serial library in Python, a simple script was determined to be suitable for the base_controller node. The base controller script, in general, manages the two serial ports, initializes the motor controller with the desired configuration, and bridges the ROS system with the motor controller board.

### 4.3.1 Motor Control

The first function the base_controller node serves is a subscription to the /cmd_vel topic in order to translate these messages to motor controller commands. This message topic is standardly used for platform movement and is very rarely issued anything other than a Twist message. The ROS core libraries include a set of built-in message types, with the set Geometry Messages purposed for common geometric primitives such as points, vectors, and poses. The Twist message type, included in Geometry Messages, is composed of a linear and angular three-dimensional vectors denoting velocities for linear and angular motion in the (x,y,z) coordinate plane. The vector values are in standard units with linear motion in meters per second and angular rotation in radians per second with negative values corresponding to the counterclockwise direction. The message components are seen in Figure 4.5 and produced with the rosmsg command line tool that can display message structures.



```
ubuntu@arm:~$ rosmsg show geometry_msgs/Twist
geometry_msgs/Vector3 linear
  float64 x
  float64 y
  float64 z
geometry_msgs/Vector3 angular
  float64 x
  float64 y
  float64 z
```

Figure 4.5: Twist Message Included in Geometry Messages Set

With an input of these formats, the conversion to motor controller commands next considered the motor controller interface for movement commands. The Arlo kit included a set of disk encoders that resolved 144 distinct positions in one full wheel rotation. The firmware guide for the motor controller reveals the distance of 15.5 inches of travel corresponds to approximately 118.4 encoder ticks [51]. This was used to establish a conversion constant that allowed meters per second ROS commands to be converted to the accepted encoder positions per second motor controller commands. With a number of available options for actually supplying the motor controller with the desired velocity, the "GOSPD" command was the most appropriate for the simple reason that this command could be streamed. That is, the motor controller will automatically apply smooth transitions between issued commands specifying different velocities. This was beneficial as many ROS teleoperation methods issue commands on the ROS /cmd_vel topic in high-frequency control loops.

The next step for converting ROS commands involved consideration for the robot platform itself. As it is a non-holonomic robot, with a left and right motor enabling differential drive, the Twist messages containing linear and angular velocities must be converted to wheel-independent velocities. Being a non-holonomic ground robot, the platform is only able to rotate and move forward or backward, thus limiting the applicable Twist component velocities to $v_{LinearX}$ for linear movement and $v_{AngularZ}$ for rotational movement. For the calculations of the left and right wheel velocities, respectively $v_{LeftWheel}$ and $v_{RightWheel}$, the conversion constant $C$, discussed above, and the radius of the differential drive wheelbase, $r_{Wheelbase}$, must be known. With these known values and velocities from ROS control messages, equations 4.1 and 4.2 are used to provide speed commands to the DHB-10 motor controller in the units of encoder positions per second.

$$v_{LeftWheel} = (C \times v_{LinearX}) - (C \times v_{AngularZ} \times r_{Wheelbase}) \tag{4.1}$$

$$v_{RightWheel} = (C \times v_{LinearX}) + (C \times v_{AngularZ} \times r_{Wheelbase}) \tag{4.2}$$

The base controller implements these equations in a received message callback function. Upon receiving a message on the subscribed /cmd_vel topic, the callback function is entered to parse the message, apply the conversions to derive motor controller commands, and lastly issue the appropriately calculated velocities with the "GOSPD" command over the serial TX port. This script enabled the callback function as a secondary priority to the primary control loop handling odometry.

### 4.3.2 Odometry

The base controller script employs a control loop that that firstly handles the retrieval of odometric information from the motor controller, and secondly handles the motor control commands being received if any. Upon the script being run or after an integrated auto-reset, the motor controller is initialized to ensure it is ready to receive commands or requests and subsequently gives way to the control loop. This loop is managed by the rospy library that enables a best effort attempt to run a loop at the desired frequency.

Determining the theoretical maximum operating frequency of the motor controller involved an analysis of the exact commands and responses that are sent over the serial connections. Though motor speed commands are not always available, this estimation was made under the assumption a request/response for current speed values and single command for motor speed would be issued in one iteration of the control loop without any overlapping transmissions. Utilizing Tables 3.3 and 3.4, the upper limits of lengths of these commands can be determined and in the iteration of a single cycle a maximum of 37 bytes is needed to be communicated over serial lines. With the operating baud rate of 19200bps, an ASCII message requiring one byte per character, and a total of 37 bytes, this control loop could operate at a maximum frequency of approximately 64Hz or over the period of about 15ms. As this controller interface is based in a software environment which introduces large overheads, this upper limit is difficult to reach. For this reason, the control loop was set to run at a still adequate 20Hz.

In addition to simply retrieving the current speed of the motors, the base controller script is responsible for the generation of odometry information that can be published to ROS on the /odom topic. This topic is almost exclusively used for Odometry messages that are defined in the standard set of Navigation Messages, or nav_msgs, included with ROS. This message type, shown in Figure 4.6, includes information about the pose, orientation, and odometry of the platform as well as an additional covariance matrix for each. In order to publish this information, it must be calculated. The first step requires the current motor speeds of the left and right motors, $v_{LeftWheel}$ and $v_{RightWheel}$, and a constant, $C$, used for converting velocities in encoder positions per second to the appropriate meters per second. These values as used in Equation 4.3 will produce the current linear velocity in the X dimension of the robot's coordinate frame, $v_{LinearX}$. Additionally including the diameter of the platform wheelbase, $d_{Wheelbase}$, allows the use of Equation 4.4 to find the current angular velocity around the Z-axis, $v_{AngularZ}$ of the platform.

$$v_{LinearX} = [(v_{RightWheel}/C) - (v_{LeftWheel}/C)]/2 \tag{4.3}$$

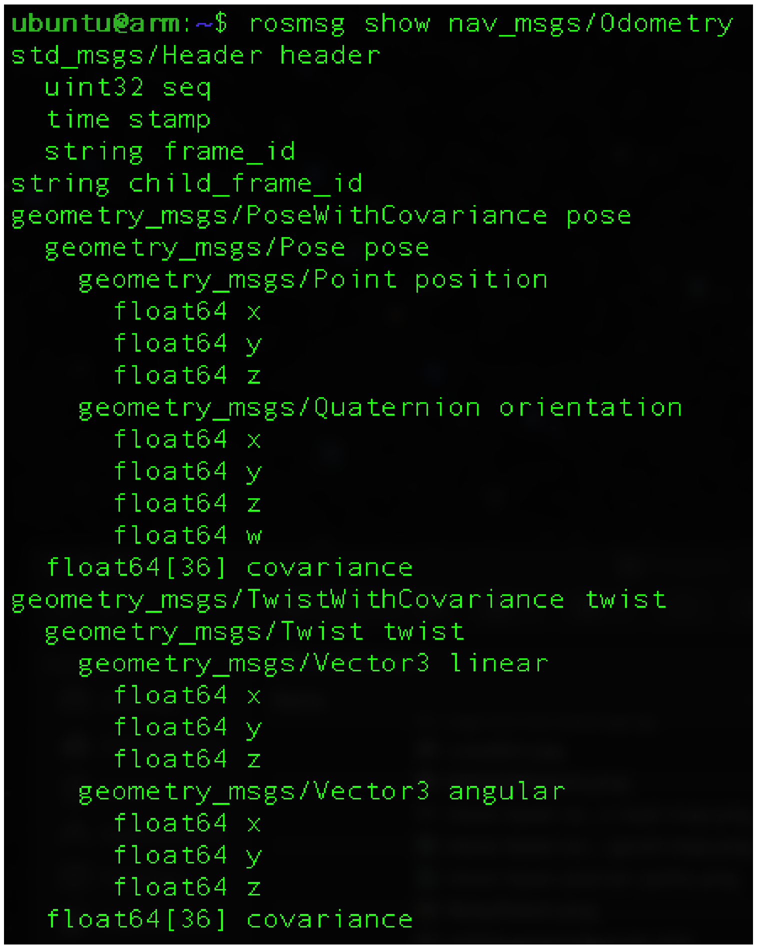
$$v_{AngularZ} = [(v_{RightWheel}/C) - (v_{LeftWheel}/C)]/d_{wheelbase} \tag{4.4}$$

These velocities calculated are in terms of the robot's coordinate system. A reference frame transformation must be made to determine the velocities of the platform in the global coordinate system, which is standardly referred to in ROS as the *odom* frame. These are denoted $V_X$ and $V_Y$ with the Z dimension being ignored due to the platform's inability to move about this dimension. Equations 4.5 and 4.6 can be used with the current robot heading, $\theta_{Robot}$ to make this transformation.

$$V_X = v_{LinearX} \times cos(\theta_{Robot}) - v_{CurrentLinearY} \times sin(\theta_{Robot}) \tag{4.5}$$

$$V_Y = v_{CurrentLinearX} \times sin(\theta_{Robot}) + v_{CurrentLinearY} \times cos(\theta_{Robot}) \tag{4.6}$$

With velocities now expressed in the global frame of reference, forward kinematics can be applied to make an estimation of the current position and orientation of the robot. Leveraging



```
ubuntu@arm:~$ rosmsg show nav_msgs/Odometry
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
string child_frame_id
geometry_msgs/PoseWithCovariance pose
  geometry_msgs/Pose pose
    geometry_msgs/Point position
      float64 x
      float64 y
      float64 z
    geometry_msgs/Quaternion orientation
      float64 x
      float64 y
      float64 z
      float64 w
  float64[36] covariance
geometry_msgs/TwistWithCovariance twist
  geometry_msgs/Twist twist
    geometry_msgs/Vector3 linear
      float64 x
      float64 y
      float64 z
    geometry_msgs/Vector3 angular
      float64 x
      float64 y
      float64 z
  float64[36] covariance
```

Figure 4.6: Odometry Message Included in Navigation Messages Set

Foward Euler Integration would allow the velocities to be integrated over time to return a current positioning. However, integrations over discrete, non-continuous time intervals force the use of the explicit version of this method and introduce error at every time step. The best countermeasure to this is minimizing time steps. This method is shown in Equations 4.7 through 4.9 with current positions denoted by $X_{Pose}$ and $Y_{Pose}$, previous positions labelled $X_{Pose_{t-1}}$ and $Y_{Pose_{t-1}}$, and elapsed time between current and previous position estimations denoted by $\delta t$. Orientation estimations use the previous orientation, $Z_{Orientation_{t-1}}$, and current angular velocity, $v_{AngularZ}$ to integrate the current orientation, $Z_{Orientation}$. With the Arlo platform's motor controller, it is not necessary to calculate the current heading as there exists a command for retrieving the motor controller calculated heading. However, this work faced issues when issuing both a "SPD" and "HEAD" request in quick succession as the control loop of the base controller required. For this reason, the only readings taken from the motor controller were the current speeds of each motor. Demonstrations in later sections have been successful using these odometry estimations.

$$X_{Pose} = X_{Pose_{t-1}} + V_X \times \delta t \tag{4.7}$$

$$Y_{Pose} = Y_{Pose_{t-1}} + V_Y \times \delta t \tag{4.8}$$

$$Z_{Orientation} = Z_{Orientation_{t-1}} + v_{AngularZ} \times \delta t \tag{4.9}$$

Having used the above equations to establish odometry estimations, the control loop will proceed with the publication of the Odometry message on the /odom topic. Additionally, a transformation must be broadcast to expose the *odom* frame as the parent to the robot coordinate frame, the base_link frame exposed by the robot model. This simply entails the publication of a transform message to the /tf topic.

## 4.4 Teleoperation

The above sections describe how the minimal functionalities were enabled for the ADR. Though one motivation behind establishing these features was basic manual remote control or teleoperation of the platform, no mechanism of actually issuing control commands has been discussed yet. A ROS node meant for teleoperation will almost always utilize the *cmd_vel* message, which was detailed in the previous section. Unfortunately, with the standard releases of ROS, there is no provided support for teleoperation. However, there are numerous packages available for this purpose with the most common method of teleoperation is via a keyboard.

The official ROS tutorials based around the simulation environment, TurtleSim, include the turtle_teleop package that allows the control of movement of the turtle image using the keys of a keyboard. Fixed velocities are associated with key presses to issue cmd_vel messages to control the forward/backward or rotational movement of the turtle. Once a key is pressed, the hard-coded values are issued in a velocity command followed by a subsequent command to stop movement after one second. Another widely used package, *teleop_twist_keyboard*, expands the basic functionality of keyboard control with modifiable linear and rotational velocities [60]. This package was used for developmental testing of the ADR platform, though with its targetted use-case as a prototype robot platform sutibale for basic demonstrations, additional teleoperation systems were included.

A demonstration platform ought to have numerous methods of control to further increase the flexibility of the applications that can be implemented and shown. For this reason, the control of this platform was enabled on a smartphone. With no particular justification for the operating system choice other than availability, a manual control application and a set of high-level integrations were developed for iOS. This smartphone OS offers numerous methods for developing custom mobile applications, one of which is provided by an application developed by OMZ Software, Pythonista. It provides a full-support interpreter for Python 3.5 and Python 2.7 and a complementing set of python packages exposing many of the APIs available on an iOS device [61]. With the ability to easily utilize the touch interface of the smartphone, it was determined that a simple teleoperation python script would be beneficial for the ADR. Though most typical python implementations for teleoperation would be hosted on a ROS compatible environment, no known ports of ROS to iOS are known. This restricted the python script from utilizing the ROS communication framework directly. However, implementing a simple listening server on either the controlled platform or another remote master server would suffice in issuing cmd_vel messages from any platform not supported by ROS. Thus, the iOS control python script was implemented in this fashion.

More specifically, the iOS script publishes control commands via a UDP socket to a corresponding UDP server that listens for incoming commands. Hosted in a ROS compatible environment, the server registers itself as a publishing node and redirects incoming UDP control commands to the cmd_vel topic. The iOS script begins to broadcast commands once a single touch is registered on the display and held there allowing a stream of commands for smooth accelerations. Dragging a finger up or down on the screen from the initial touch will respectively increase or decrease the linear velocity of the command issued. Dragging to either side will increase rotational velocity in the corresponding direction. An example command can be seen in a screenshot of the graphical user interface for this iOS script in Figure 4.7. This figure shows a touch that originated at the center of the screen and moved upward and left, corresponding to a cmd_vel message defining a linear and rotational velocity.

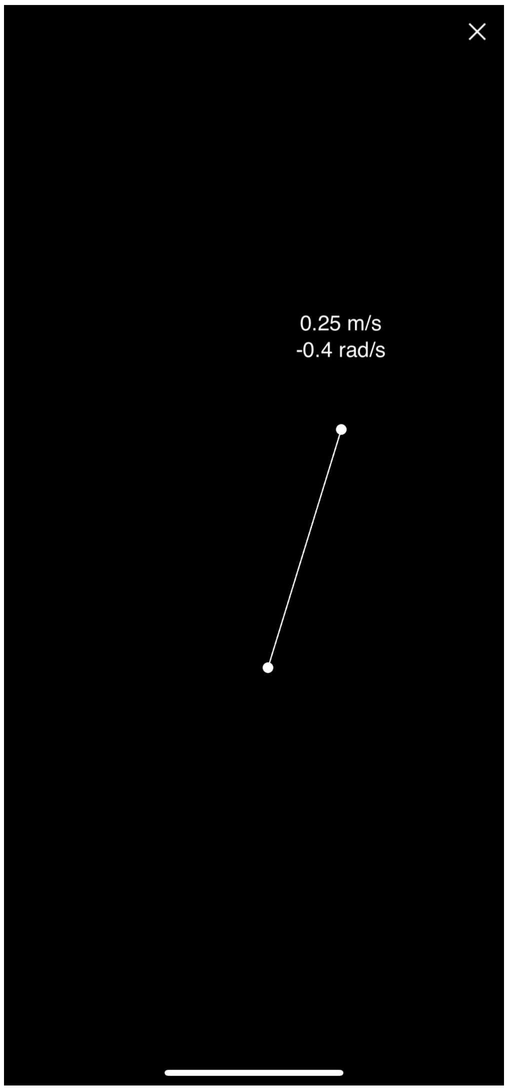


Figure 4.7: Teleoperation Interface for iOS

Another tool that developers have available to them when working with iOS is the personal assistant, Siri. This enables the use of the integrated personal assistant for triggering actions within iOS with custom voice commands, known as Siri Shortcuts [62]. For defining more complex actions than what is available by default, the Shortcuts app executes a sequential set of actions [63]. Other iOS applications can provide an API which is accessible from Shortcuts which facilitates specific triggers even within applications. Pythonista provides such an API and allows users to run a script in Pythonista just by asking Siri to do so. As the python script for teleoperation of the ADR already establishes the feasibility of using a smartphone for control, higher-level commands issued from a voice-triggered python script could add significant value to the platform as a whole. Specific implemented voice controls are discussed further in Section 5.3.3.

# Chapter 5

# High-Level Capabilties and Demonstrations

The final developmental stage of this work, highlighted in Figure 5.1, is covered in this chapter. As this work developed the Arlo Demonstration Robot's operational-critical features of the control system in previous sections, attention could be turned to adding more advanced functionalities leveraging these underlying mechanisms and extending the capabilities of the robot platform. The review of some state of the art features helped reveal the abilities that successful commercializations of robot platforms typically include. Enabling some of these features on a platform such as the ADR was intended to help prove the feasibility of hosting advanced features found in industry on small-scale development platforms. Sharing and demonstrating validations such as this can encourage future development efforts to be made in the field of robotics.

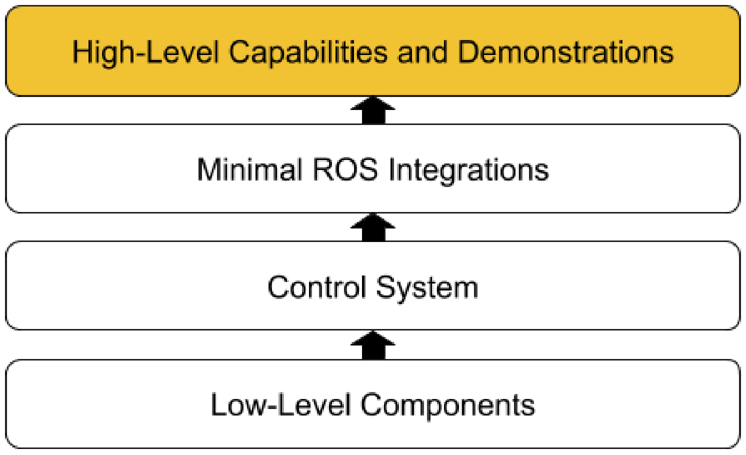


Figure 5.1: Chapter Five Development Goals for the Arlo Demonstration Robot

By far the most common functions seen in many mobile ground robots in industry is mapping, localization, and navigation which are enabled through various outfittings of peripheral sensors. This chapter discusses the implementations of mapping and localization on the ADR with SLAM and two unique sensor types to allow for autonomous navigation. Demonstrations for each of these features as well as some high-level demonstrations are also presented.

## 5.1 SLAM

With the proprietary nature of commercialized robots, the implementations of mapping and localization features found on many of these platforms are not available. In order to enable some form of environmental awareness, a map must be created and used for determining its location in regards to its environment. As discussed in Section 2.2.2, Simultaneous Localization and Mapping is a commonly adopted solution for handling these functionalities and has seen numerous research efforts directed towards it. With the wide range of packages offered with ROS, SLAM implementations are not difficult to come by. This section discusses a set of chosen sensors whose outputs are applicable to SLAM algorithms. It further presents the implementation and demonstration of a set of actively maintained SLAM algorithms capable of leveraging the chosen sensors.

As discussed up to this point, the ADR platform has the ability to perform an environmental scan that can be transformed into the robot coordinate frame and publish odometry for position estimations and telemetry. In this setup, the odom frame is the root of the TF tree of the robot model. This only reveals the estimations of where the robot believes it is in relation to the point at which odometry measurements began. In order to create a true map frame, SLAM will generate and broadcast a map frame. This new frame is listed as the parent of the odom frame and is defined by a transformation that SLAM will apply. With this configuration, SLAM can apply its estimation of robot localization in a self-produced map while also accounting for the error produced from odometry estimations.

### 5.1.1 SLAM Sensors

SLAM requires input data from sensors that can produce some kind of distance measurements of the environment. Using this, SLAM attempts to maintain a continuously updated state of the platform and its environment. Maximizing the input data can lead to better representations of state and enable more accurate predictions of the localization. For this reason, sensors that can quickly and accurately measure the environment are essential to maximizing the performance of SLAM. Though this work does not have access to industrytested components, low-cost derivatives of the sensors seen in industry are utilized for the ADR platform to enable SLAM.

For the discussed sensors, each produces data as an environmental scan. The placement of the sensor in reference to the robot frame of reference is typically essential in the processing of any produced data. For this reason, the ADR robot description was updated to include the placement, dimensions, and parent TF frame for each sensor. Figure 5.2 shows this updated model with a Kinect camera added on top and a laser scanner, denoted by the green box, that is mounted upside down just underneath the top plate.

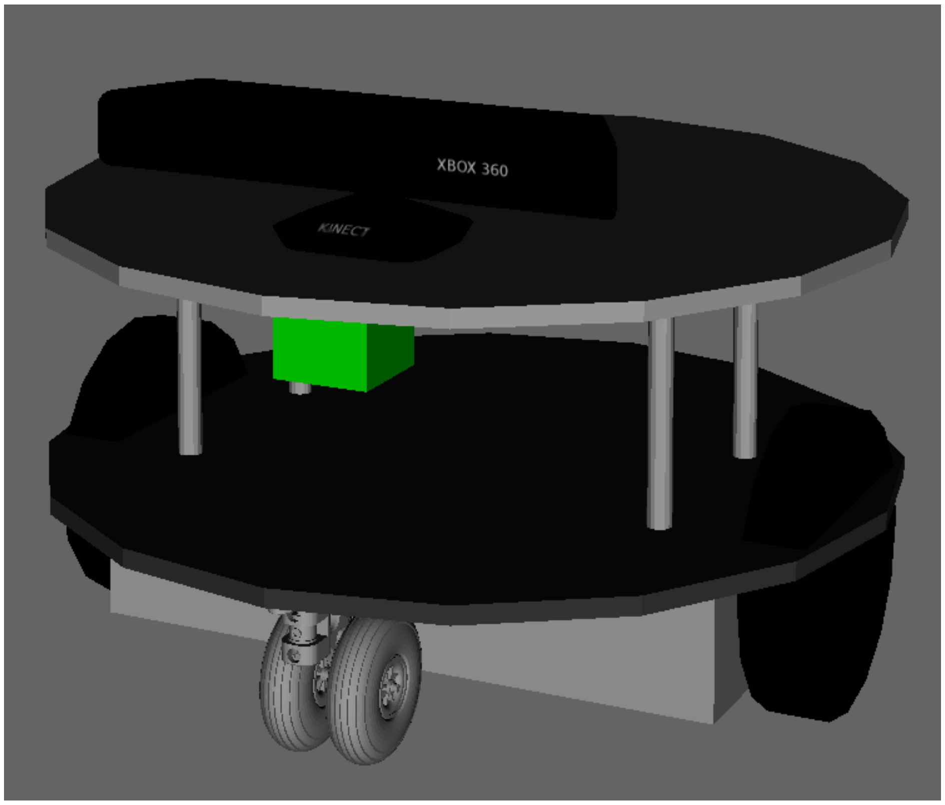


Figure 5.2: RViz Rendering of the ADR Base Platform with Sensors

### LIDAR

With the increased proliferation and decreasing costs of sophisticated laser rangefinders, LIDAR units have found their way into many of the platforms discussed in Chapter 2. With the variety of available range-finders being quite immense, inspiration was taken from the TurtlBot3's selection of LIDAR sensor. The included LIDAR is a simple two-dimensional rangefinder with a single senors that is rotated around the Z-axis for a 360° field of view. As most navigation and obstacle avoidance scenarios could be effectively implemented with only a two-dimensional awareness of the environment, this work also determined to implement SLAM with a two-dimensional LIDAR and evaluate its performance.

The LIDAR unit selected was the RPLidar A1 unit, manufactured by Slamtec. This sensor, seen in Figure 5.3 [64], is of a similar form factor to the TurtleBot3 LIDAR, though offers more support as it was not developed for a specific platform. With an adjustable full scan frequency of 2-10Hz, these sensors can generate up to 8000 point measurements per time sample [65]. This offering is currently the highest density scan for current economical LIDAR units. It can produce accurate readings up to 12 meters and at an approximate 1° angular resolution and 0.2cm distance resolution with the default configuration settings. With a serial interface exposed over USB, the sensor draws only 5V and is compatible with the ADR control system.

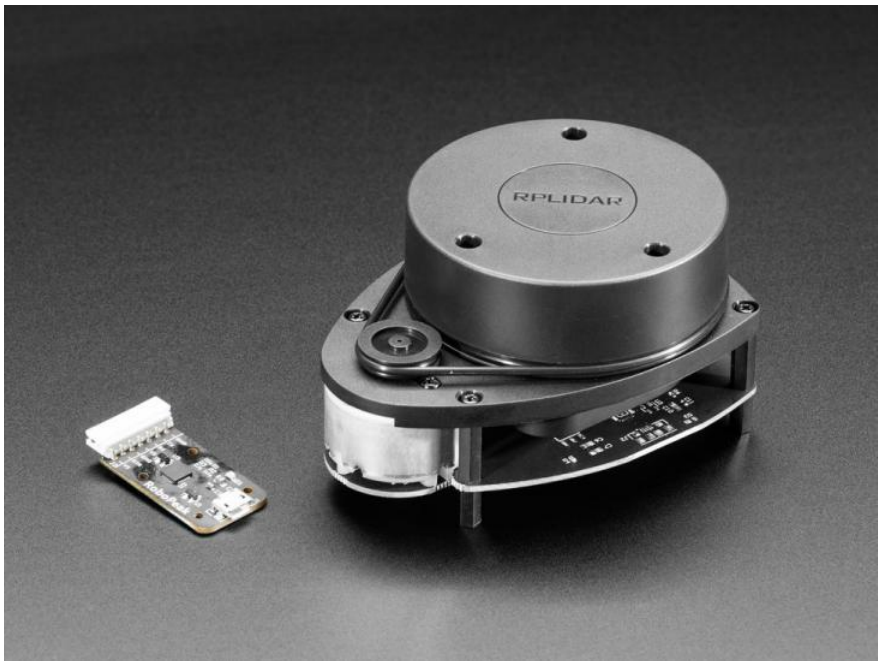
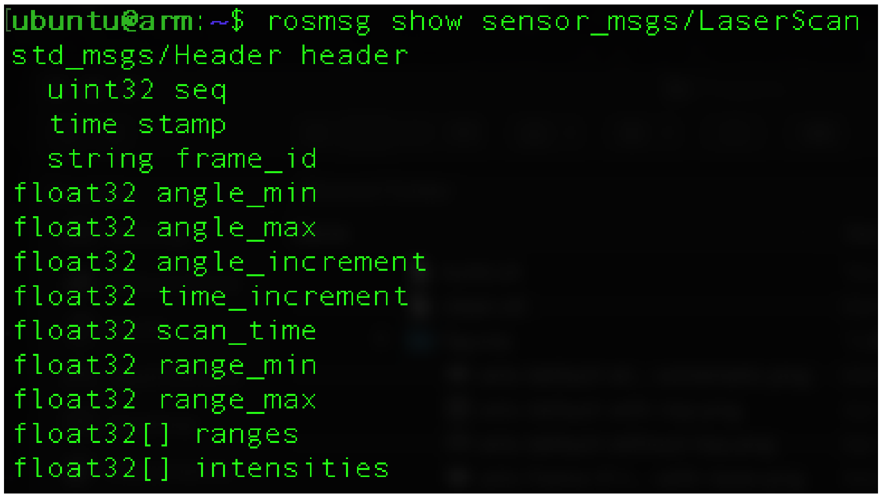


Figure 5.3: RPLidar A1, Source: Adafruit [64]

With a minimal kernel configuration generated for the Zybo control board, an additional kernel module, the CP210x USB to UART driver was needed to allow Linux to mount the sensor as an expected Linux USB device at the file descriptor /dev/USBx. For use of the LIDAR in software, Slamtec developed an SDK and additional ROS integration for this sensor allowing an interface to be set up quickly. The ROS package made available, rplidar, implements a simple driver for the USB-connected sensor utilizing the ROS independent RPLidar SDK. It enables simple commands to start and stop the LIDAR rotational motor and also modify the current scan mode used by the unit. The primary function of the rplidar ROS node is to publish the retrieved scan to the standard laser scan topic, /scan. This topic receives message of the LaserScan type, a message type provided by the set of Sensor Messages, sensor_msgs, and listed in Figure 5.4. Messages with this structure allow the packetization of measurements that are included with additional metrics of the specific measurements contained within a message.



```
ubuntu@arm:~$ rosmsg show sensor_msgs/LaserScan
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
float32 angle_min
float32 angle_max
float32 angle_increment
float32 time_increment
float32 scan_time
float32 range_min
float32 range_max
float32[] ranges
float32[] intensities
```

Figure 5.4: LaserScan Message Included in the Sensor Messages Set

A visualization of the data retrieved from the LIDAR rangefinder can be seen in Figure 5.5. The image presents a scan that was performed in an indoor location. It was produced by running the robot_state_publisher node with the ADR robot description with added LIDAR frame, the rplidar ROS node, RViz, and a static transformation publishing node broadcasting a transformation from the map to base link frame. The additional transformation allows the exposure of a map frame so RViz is able to link its visualized frame to that of one containing the robot and onboard LIDAR. For actual mapping and localization, the nodes performing

SLAM are responsible for declaring this transformation.

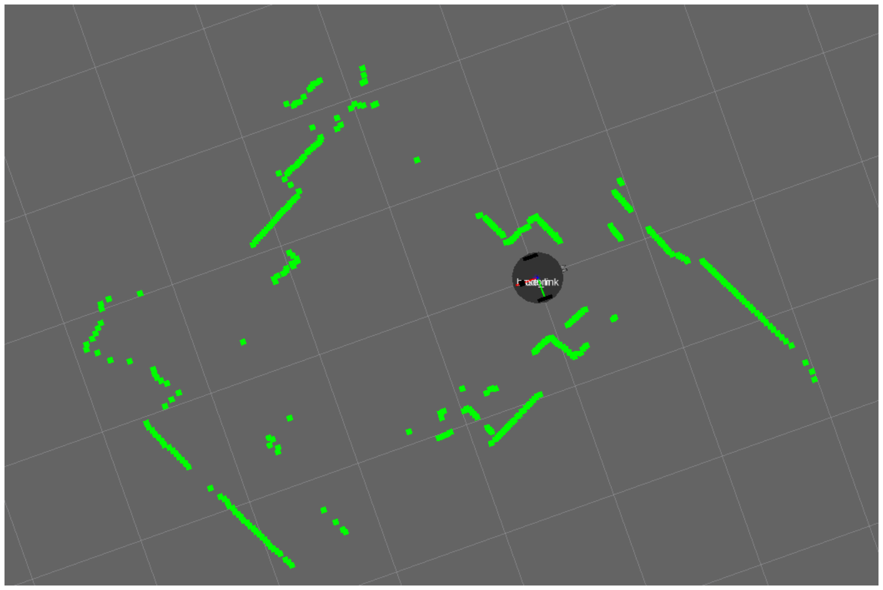


Figure 5.5: LIDAR Scan in an Indoor Environment

### Depth Cameras

With laser scanners being introduced as alternatives to the original methods of sensing the environment with basic TV video cameras in the work of the Shakey robot, modern technology has allowed more sophisticated cameras to capture higher definition images with some enabling distance sensing. One such camera that allows the capture of depth from the field of view is the Microsoft Kinect, introduced in Section 2.2.2. With the ability to produce an RGB image with an RGB camera sensor as well as a disparity image with an IR emitter and detector, these can be broadcast to ROS for further processing with, say, a SLAM algorithm. With a three-dimensional field of view, it can enable SLAM in three-dimensions further enhancing the platform's ability to perceive the environment.

These sensors have since been discontinued, but their mass production and multiple versions, allows them to be found for sale on the internet as well as local gaming shops, usually for under $20USD. This work utilized a pre-owned Kinect Sensor V1 pictured in Figure 5.6 [66]. The sensor uses a proprietary connection that was originally purposed for direct connection to an XBOX gaming console for a single port enabling USB and 12VDC power. Adapters are available to convert this to the separate USB and power components. The adapter includes an AC power converter for supplying the required DC voltage. For use on the ADR platform that does not provide an AC power source, the adapter was modified to pull 12VDC from one of the available PDB terminals.

Figure 5.6: Kinect V1 Sensor, Source: Engadget [66]

In order to interface the Kinect from the Zybo control board, the appropriate kernel modules and drivers must be installed. The Kinect's USB interface also requires the use of the CP210x USB to UART driver enabled for the LIDAR. In order to access the Kinect's control unit, additional drivers were needed. Luckily, with the large adoption of the Kinect by the open-source community, many such libraries exist. However, the availability of these drivers for Ubuntu was not as great. Further, the open-source support for these sensors has grown to a minimum with the available drivers not being maintained for supporting newer kernel or OS versions.

One such library that was successfully implemented for interfacing the Kinect with the Zybo was libfreenect, developed by the OpenKinect community. This library provides basic access to functions such as retrieving the camera feeds and controlling the internal tilt motors for the device. However, this library did not allow any exposure to the ROS communication framework. There are ROS packages that utilize the libfreenect library, though there are no existing packages for the latest release of ROS. Another library that enables a complete interface with the Kinect is the OpenNI2 library, though attempts of utilizing a direct interface with this library failed on the ADR platform. There are also ROS packages that utilize the OpenNI2 library for interfacing with the sensor in a ROS node, however, initial attempts with these packages also failed as the OpenNI2 driver was not successful on the Zybo's Ubuntu system.

It was eventually determined through substantial trial and error that the OpenNI2 library allowed the use of external drivers. In addition, the OpenKinect community has made available the OpenNi2-FreenectDriver, that interfaces with libfreenect to expose the sensor to the OpenNI2 library. With a successful interface setup, the images needed to be published to ROS. The ROS, openni2_launch, utilizes this interface to retrieve the camera feeds from the sensor [67]. Under the hood, this package actually utilizes another, rgbd_launch, which manages a set of ROS nodes that handle the processing of raw RGB or depth streams from a sensor to produce depth, disparity, and basic RGB images [68]. These nodes publish the processed images for higher-level algorithms to use. In this work, these products are used for SLAM.

### 5.1.2 Laser-Based SLAM

Most of the functionalities implemented for the ADR platform that have been discussed thus far have only been simple additions to the capabilities of the platform. The ADR robot description, base_controller, and LIDAR sensors expose all the necessary ROS nodes and topics to enable SLAM, though there has been no implemented process that actually utilizes these. Section 2.2.2 introduced some known open-source SLAM algorithms, but it was determined to use the Google Cartographer library to facilitate laser mapping and localization for the ADR.

#### Cartographer

The underlying mechanisms of Cartographer are out of the scope of this work, though there are some concepts that should be explained as the implementations introduce a number of new terms. Cartographer utilizes the input data of environmental scans that are made available over the appropriate topics, in this work's case, the /scan topic. These scans are collected and accumulated to determine a robot trajectory and optimize the pose estimation. For local SLAM, the laser scans are also used to build a submap that are consolidated with best estimates for offsets. A submap is complete after a configurable amount of scans have been made. Global SLAM computes the generated map which is a set of submaps that are merged with scan matching that considers all submaps for loop closure and submap overlaps [31, 32].

Cartographer can run in real-time or operate from recorded sets of ROS topics. As this work sought to demonstrate the SLAM functionality, only real-time configurations are discussed. A pure localization configuration that also available uses previously generated Cartographer SLAM maps to determine the robot's current location within that map based on the current scan. Though this is useful in the scope of this work, it was not explored to much depth.

The ROS cartographer package includes a number of default configurations purposed for various applications, both two-dimensional and three-dimensional. With a two-dimensional LIDAR, only these configuration setups were applicable. They are further separated by the intended platform and function, such as mapping on a robot in realtime or using a remote development machine for visualizing with RViz as the algorithm runs. For this work, real-time mapping was enabled for the robot by duplicating the backpack_2d launch file for the Cartographer package. This configuration was run on both the robot platform and a powerful development machine. The launch file simply runs a Cartographer ROS node with a configuration file that is specialized for the ADR robot platform. There are numerous configuration options, though only a few need be mentioned.

Options that must be configured according to the hosting platform include the specification of the tracking frame, number of laser scanner inputs, and number of points per laser scan. This tells Cartographer the frame that is to be localized in the generated map and the format of the laser scan sources. With the root TF frame of the platform is base_link, the tracking frame is set to this link. Additionally, the single utilized laser scanner outputs approximately 550 points per ROS message publication to the /scan topic, so the corresponding configuration options were set accordingly.

Upon launching Cartographer with the ADR custom launch script, the node will begin to listen for the necessary incoming messages published by the various peripheral components. After a while of collecting laser scans, a two-dimensional map will begin to be created. One such example map is seen in Figure 5.7. This visualization reveals a bit more of what exactly is represented in a map. The green outlines are current laser scan points, just as in Figure 5.5, but the dark outlines denote occupied space that has been detected by past scans. The interior of the map is shaded with varying intensities of grayscale. This is due to it representing the certainty of Cartographer that the space is free with larger shading intensities holding higher confidence. Confidence is increased as more scans are performed and matched to the submap set of scans. The blue line simply represents the robot's path of travel during this run.

### 5.1.3 RGB-D SLAM

Though the above section establishes a method of generating quality two-dimensional maps, there are many situations that could benefit from three-dimensional mappings of the environment. With an on-board Kinect camera added to the ADR platform, this work intended to explore the possibility of generating three-dimensional maps from RGB-D SLAM, introduced in Section 2.2.2. The Real-Time Appearance-Based Mapping (RTABMAP) project implements such an RGB-D approach. Much like Cartographer, this library provides the ability for real-time SLAM with considerations for global loop closure. Additionally, a ROS package, rtabmap_ros, is provided as a wrapper for this library enabling the use of ROS topics for gathering the necessary input data.

#### RTABMAP

Though the latest development of RTABMAP in [41] allows the use of a variety of sensors, it was originally proposed for appearance-based mapping with RGB-D inputs in [40]. With a

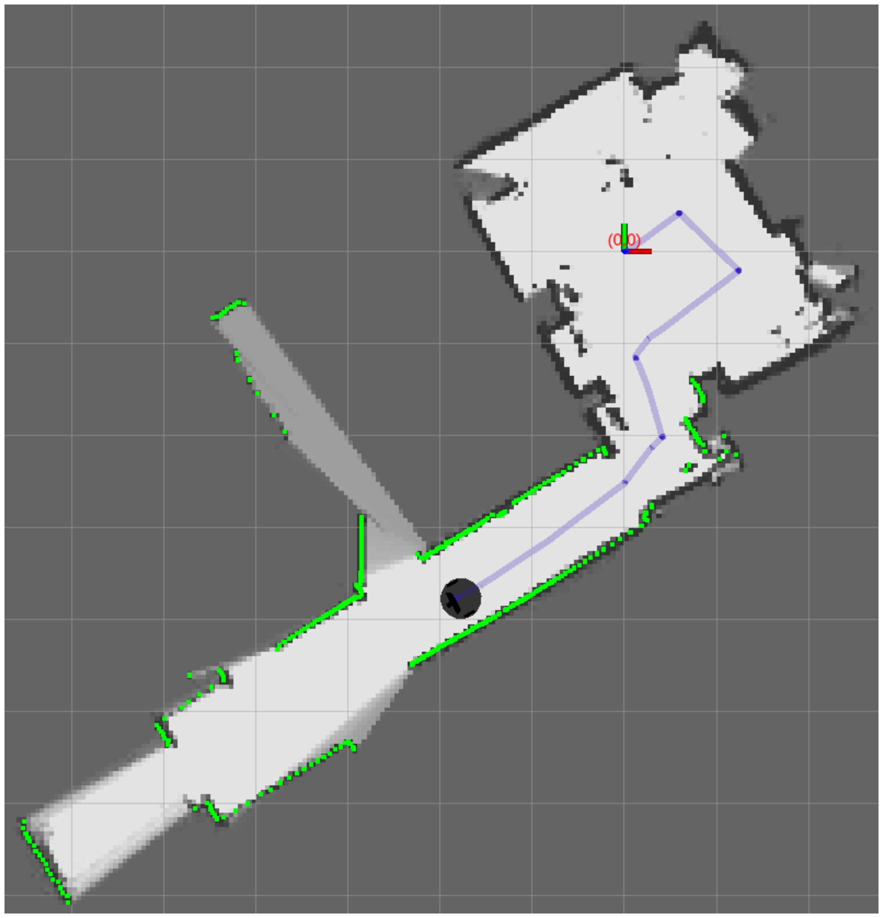


Figure 5.7: Example Cartographer-Generated Map

direct comparison of RGB-D and laser-based SLAM being outside of the scope of this work, it was determined to only utilize RTABMAP for the purposes of three-dimensional SLAM mapping with the Kinect sensor and delay the comparison for future work.

The RTABMAP ROS package has a rather large set of configuration options for the nodes run by the default launch file. As expected, parameters dictating the correct frame ID for the tracking frame and desired map frame ID are included. Running the OpenNI2 launch file as well as the RTABMAP launch file, with these two modified configurations, to process the Kinect images will produce a MapGraph. This custom definition of a point cloud is the collection of environmental scans captured by the input camera and stitched by the SLAM mechanisms of RTABMAP.

This work was successful in using all of the nodes for RGB-D SLAM, though a proper map was not achieved. With the numerous configurations of RTABMAP, a sufficient configuration was never found to resolve the issues faced with the scan matching component of RTABMAP. As an example, Figure 5.8 shows the result of one such attempt to create a three-dimensional map, where two point clouds are visualized in RViz. The two point clouds were the resulting product of RTABMAP's attempt to stitch the images captured as the robot was turning to the left. The primary issue faced was the segmentation of point clouds when doing a scan of one contiguous area. It should instead produce a single merged point cloud without missing portions. Additionally, the point clouds are presented to RViz upside down, or inverted in the Z dimension. The dense collection of points near the top of both point clouds are actually measurements of the floor. Speculation of the underlying causes of these problems is discussed in Section 6.2.

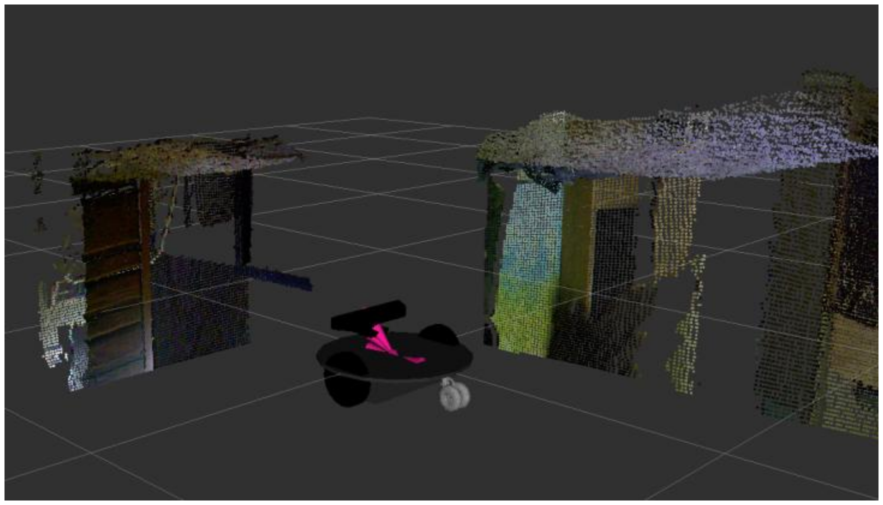


Figure 5.8: RTABMAP Visualization Attempted with RViz

With the issues faced, no RGB-D SLAM implementation to generated a reliable map or platform localization was brought to fruition. However, the ADR platform enables all the necessary components needed for such an implementation and even facilitates basic three-dimensional sensing. A deeper dive into the configuration options of RTABMAP is needed to duplicate the successful results of other works utilizing this library.

## 5.2 Navigation

The ADR platform's capabilities have grown substantially up to this point, however, the robot is still unable to operate outside of manual control setups. To enable autonomy for the platform, navigation and obstacle avoidance was to be implemented next. Simple navigation would facilitate the movement of the robot from one point in the environment to another. This can be achieved with relative ease when in an environment that contains no obstacles. However, in an environment with features like corridors and rooms, the walls cannot be navigated through and must be avoided in the traversal to a target point. Obstacle avoidance requires the more basic function of obstacle detection. With the SLAM algorithm that was implemented, the laser range finder helps generate a map that designates detected free and occupied space. Simply avoiding the areas of occupied space would be appropriate for many scenarios, though specifications of the robot platform itself should be known, such as the dimensions of the platform, the placement of the laser scanner, and maximum and minimum readings that can be collected during a scan. These would allow for more robust handling of obstacle avoidance as passages that are too small can be avoided and sensor readings outside the reliable limits can be ignored.

### 5.2.1 Move_Base

The move_base ROS package [33] is provided as an implementation of ROS actions, exposed in the ROS package, actionlib. Actions are simply preemptable tasks such as moving to an environmental target location. The move_base package simply attempts to complete the requested navigational action. With actions such as providing a navigation goal, a local and global planner are both used to build a path in order to reach the target. It is meant to be utilized on top of the ROS navigation stack. As mentioned earlier, the suggested navigation stack was broken for ROS Melodic at the time of this writing. In the case of this work, it used move_base as the controller for navigating a Google Cartographer built two-dimensional map with the ADR platform, where Cartographer was used to replace the broken navigation stack nodes.

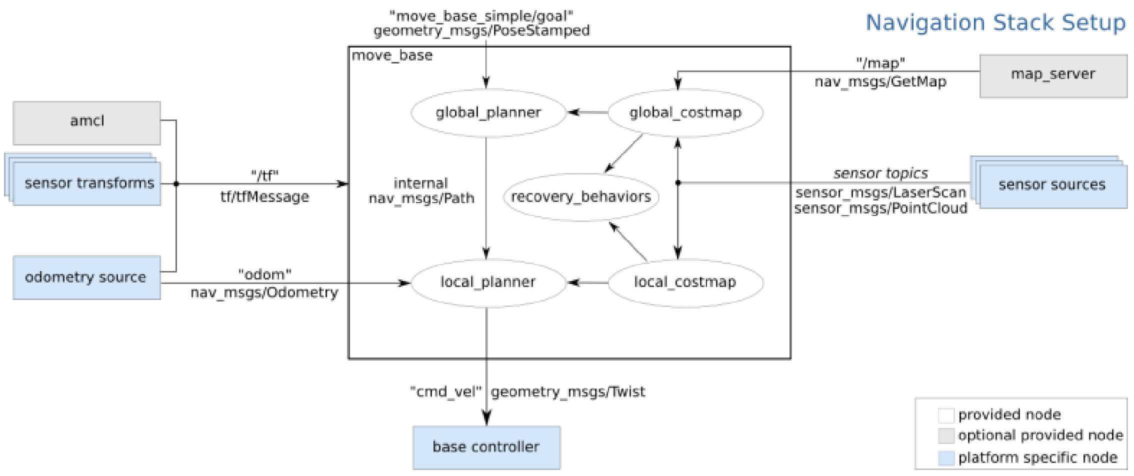


Figure 5.9: Move_Base ROS Node Overview, Source: wiki.ros.org

A high-level view of the components and interactions included in the move_base node are shown in Figure 5.9. Both of the planners included with move_base utilize a separate two-dimensional costmap for path planning that is generated based on specific configuration files for each costmap. The parameters in this file specify the dimensions, resolution, and update frequency for the costmaps generated by the planners. Additionally, the global and robot frames must be specified for gathering the map to build a costmap and retrieving the location of the robot in regards to the costmap, respectively. Each of the planners is also configured with their own specification files for setting parameters such as robot acceleration and speed limits, forward simulation granularities, and path scoring weights, and lastly the controller frequencies. The optimization of these parameters for the ADR platform was outside the scope of this discussion, though configuration files are included with the ADR stack of ROS nodes.

To show the visualized products of move_base, Figure 5.10 shows the costmap calculation on a generated map from Cartographer's SLAM processing. As the map used in this image is well-established, the costmaps, shown overlaid on one another here, are also well-defined. The map denoted by the grayscale shading has had sufficient sensor readings to rate these areas with high confidence. In turn, the obstacles denoted by the blue and pink costmaps are finer tuned as a result of this map confidence.

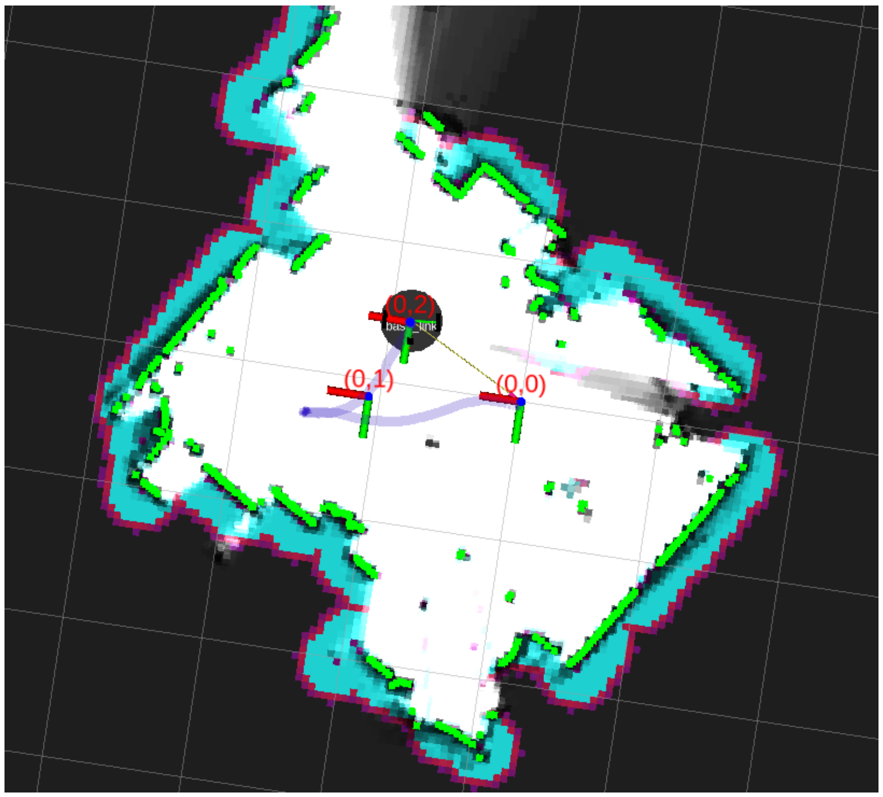


Figure 5.10: Costmap Generated on Well-Developed Map

Figure 5.11 shows the path calculations during an attempt to reach a navigational goal in a poorly mapped environment. The paths are labeled *Global* and *Local* for the global and local planner produced paths, respectively. It can be noticed in both figures that the costmaps are inflated significantly from the map-denoted occupied space. This inflation is an inherent feature of the move_base planners and can be sized appropriately to ensure the platform will never attempt to navigate a passage smaller than the platform itself.

## 5.3 Demonstrations

Having set a goal for the development of a prototype mobile robot platform, this work introduced a number of minimal functionalities required for ROS as well as some high-level processing layered on top of the basic capabilities hosted on the Arlo Demonstration Robot platform. Setting a series of basic demonstrations with the platform was intended to inspire future research developments towards the platform itself and other robotics projects in general as the ADR has comparable specifications to industry robots but no sole application

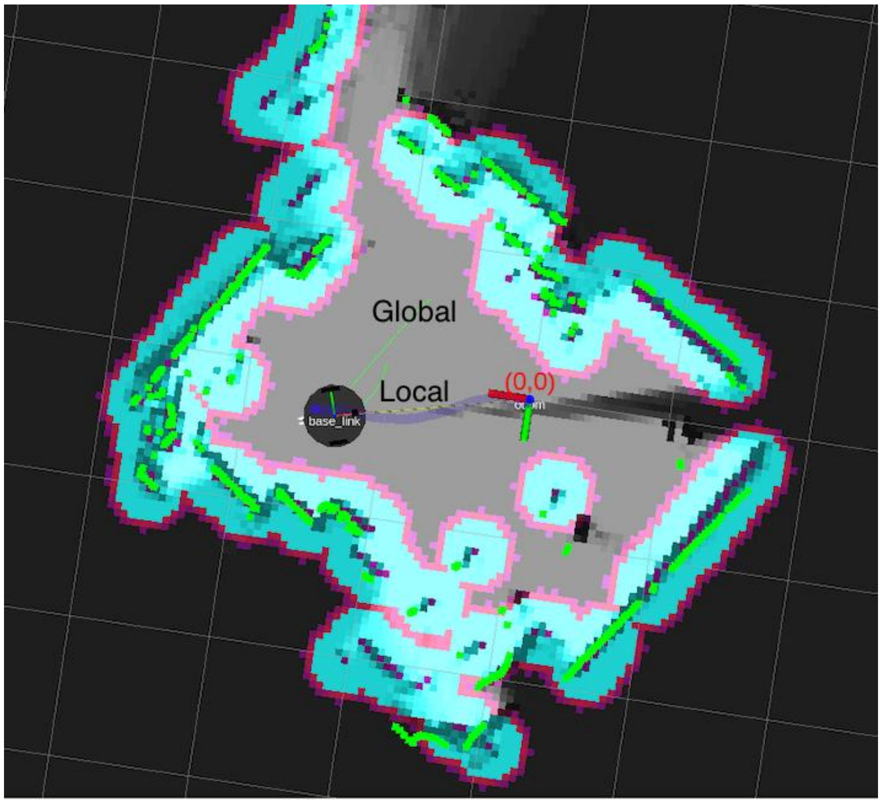


Figure 5.11: Navigational Goal and Corresponding Local and Global Paths

was set for it. This facilitates a platform that can encourage the imagination of the demonstration audience and enable the realization that only a small effort would be required to fit this platform to any number of use-cases, even commercial applications.

### 5.3.1 Generating Floorplans

One simple application the ADR can be used for is basic mapping. Allowing the robot to move throughout an entire indoor environment while processing laser scans with SLAM will produce a map as discussed in earlier sections of this chapter. However, the map generated can be useful beyond the robot's localization purposes. Construction contractors, facility managers, and even search and rescue teams can utilize the data contained in a basic floorplan.

In order to validate the feasibility of using this platform for such as purpose, the floorplan of an entire single family was developed and analyzed. The generation of the floorplan shown in Figure 5.12 was created with laser-based SLAM using Google Cartographer. The platform was manually controlled with the iOS teleoperation app for moving through the environment.

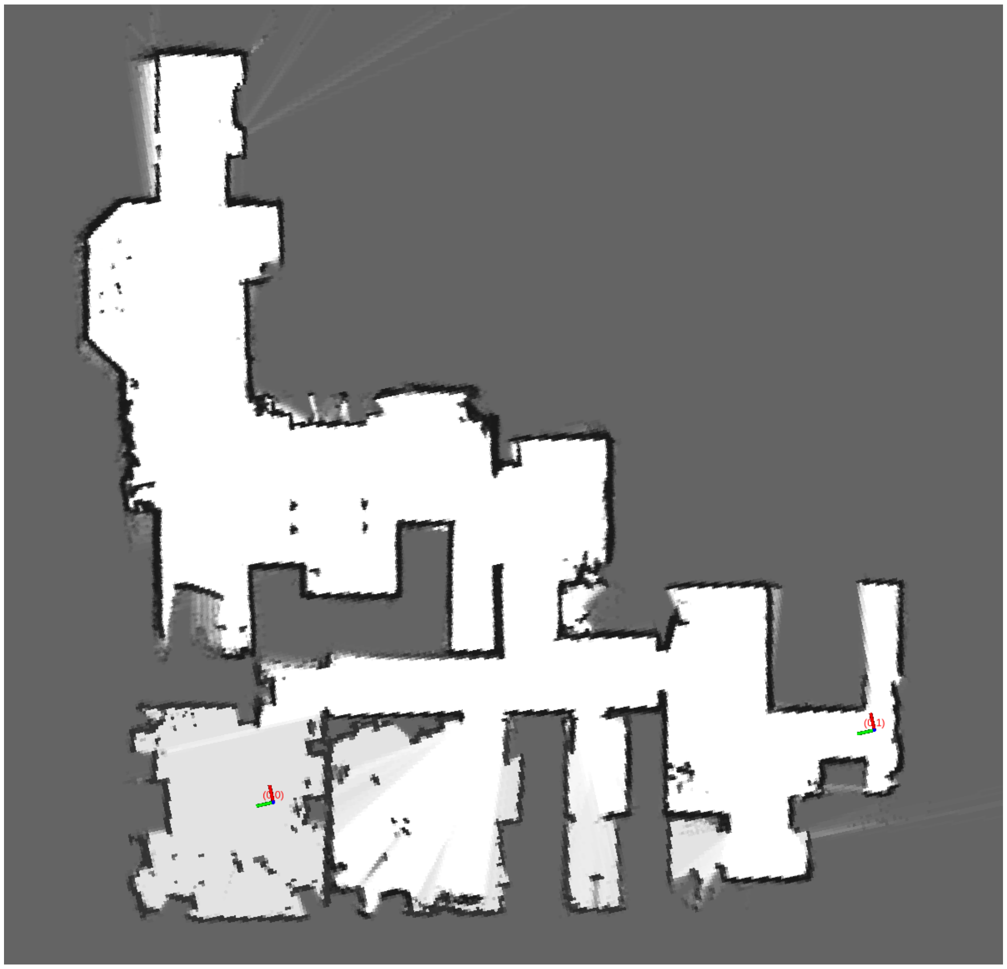


Figure 5.12: Home Flooplan

### 5.3.2 Semi-Autonomous Exploration

Though the basic map generation application represents a valid use-case, manual control of every robot movement is tedious and should be alleviated. The introduction of the move_base ROS package allowed current laser scans and the known map to be used for planning paths between navigational goals. However, it was not previously mentioned that map creation and basic navigation could coexist, thus allowing move_base to manage the teleoperation of the platform.

Figure 5.13 shows an example of this where the platform was placed in an indoor environment with SLAM and navigational ROS nodes being run. The green path was visualized

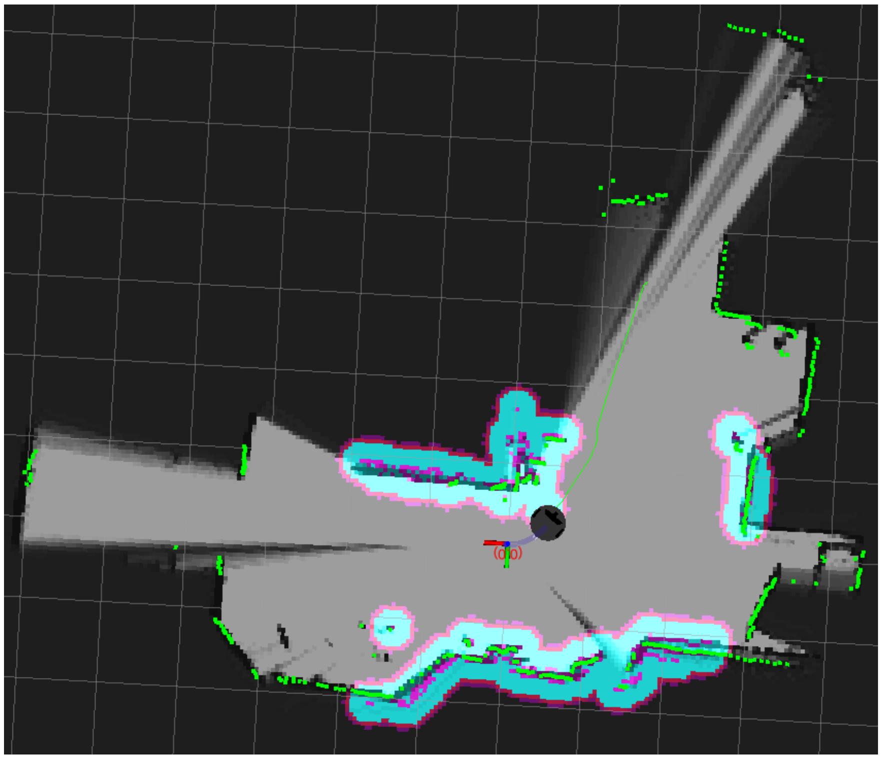


Figure 5.13: Semi-Autonomous Exploration with Move_Base and Cartographer

after setting a navigational target to move to with the RViz tool. It can be noticed from the intensities of the map that shading is very low at this target location and this was due to the platform not having mapped this area yet. Even still, the move_base node attempts to reach that position, mapping and recalculating the best path for reaching the target. This particular setup and use-case introduced new concerns. For instance, had the navigational goal not been known to exist in the reachable environment, the platform would still attempt to reach its goal though obstacles may be discovered to be in the set path. Thus, this environmental exploration setup still requires an operator to manually issue navigational commands and monitor the robot's progress toward the goal. Additionally, this setup, as run on a remote development computer, faced issues with data inputs not being able to keep up with the high-frequency control loops of the nodelets of move_base and Cartographer. The map shown in Figure 5.13 is not very well-established as a result.

There are existing ROS packages for handling environmental exploration, though demonstrating such on the ADR platform for this work and other intended venues could face limitations as imposed by various demonstration environments. For this reason, no further efforts were made to enable exploration capabilities.

### 5.3.3 Voice Control

Effective demonstrations should not be constrained by the environment in which they are given and should not require much overhead to support them. With an established method of utilizing the iOS integrated voice assistant, Siri, and the Pythonista application for communicating to a ROS system, a set of simple routines could be developed and run on the ADR platform with only a smartphone necessary for interaction and control.

To enable a flexible solution for this, a single python script was written for iOS. This script accepts a number of subcommands that are defined within the script itself. Utilizing a TCP connection, the iOS script will establish a connection to a corresponding server script on the robot platform, issue a command, and terminate to be ready for another run iteration. This server application simply listens for connections and parses the incoming subcommand to execute the appropriate ROS node(s) that actually implements the demonstration. Using the Shortcuts application along with Siri Shortcuts allows any number of voice triggers to be used to run the iOS script with any number of subcommands. A few examples of these voice-activated demonstrations that have been implemented and are summarized in Table 5.1. Each is defined as a unique subcommand in a custom ROS node that acts as a resource manager.

With the relatively large set of capabilities exposed by the ADR platform and even ROS in general, an effective demonstration would require the ability to precisely control the components and processes that were being utilized at any given point. Thus, this work has enabled the ability to control the LIDAR unit, the Kinect sensor, the motor controller power supply, and even the Zybo control board itself. The LIDAR and Kinect sensors can be enabled and disabled for scanning and the Kinect additionally allows for control of the internal tilt motor. This was especially useful as it allows the robot to greet its audience.

Table 5.1: Actions Enabled for Voice Control

| **Subcommand** | **Description** |
|---|---|
| MOTORS ON | Triggers the motor controller power relay to close |
| MOTORS OFF | Triggers the motor controller power relay to open |
| MOTORS RESET | Triggers the motor controller power relay to toggle |
| LIDAR ON | Enables the LIDAR ROS node to enable the sensor motor and begin publishing collected data on the /scan topic |
| LIDAR OFF | Kills the LIDAR ROS node and disables sensor rotation |
| GOTO WP_ID | With a preset waypoint on a saved or current map, ensures SLAM and naviagation processes are running and will attempt to autonomously navigate, with obstacle avoidance, to a waypoint specified by WP_ID |
| GREET | Enables a sequence of actions roughly resembling a robot greeting |
| FIGURE8 | Triggers a simple figure-eight maneuver |
| REBOOT | Triggers the control board to reboot and run the subcommand server |

Stepping up a level, the voice control system also enables the triggering of full behaviors. That is it can force the spawn of ROS processes and provide high-level actions like supplying navigational goals or forcing a system reset. In an unknown environment, the ADR platform could be commanded to begin or end a SLAM mapping session to create a local map. With an environment that can be mapped preceding a demonstration, programmable waypoints can be set allowing for the robot to demonstrate its SLAM and navigation abilities. Using these waypoints, routines such as moving to center stage when demonstrating or settling to the side when not being used can be commanded and handled without any extra input. Commands were also put in place to trigger basic demonstrations of the robot's maneuverability such as simple spinning and figure-eight maneuvers.

# Chapter 6

# Discussion

The Arlo Demonstration Robot platform prototype that was developed and discussed over the course of the previous chapters proved to be a successful undertaking. However, even with the use of well-developed tools and platforms, the resulting complexity of the system was much higher than originally hypothesized preceding this work. This chapter performs a brief review and analysis of the ADR platform and discusses some of the concerns and problems faced in development. As development followed a bottom-up approach, this discussion was structured in a similar manner.

## 6.1 Final Platform Implementation

The development process comprised of a set of stages that were roughly separated by abstraction level. The first two stages developed the foundation of the ADR with the physical robot platform, primary control components, and central control board. The third stage made additions to the ROS support of the platform. With the primary control functions and physical platform configurations being exposed to ROS at this stage, teleoperation was enabled for a number of methods of control. The final stage implemented the high-level capabilities of mapping, localization, and navigation with the support of ROS-official and third-party libraries. Additionally, a set of demonstrations were developed to show off the functionalities and capabilities of the platform. A detailed image of the Arlo Demonstration Robot and final schematic, as of the final stage of development, can be seen in Figures 6.1 and 6.2, respectively.

Complementing this work, the source code for all of the implementations done will be available as open-source. This was with hopes that the platform can serve as a resource for others implementing similar features or platforms and even those implementing on this described platform. The sources include all of the ROS nodes that were developed and the custom configurations for third-party ROS packages. Also included are all of the hardware design files, Petalinux board support package for the ADR processing system, and supple-

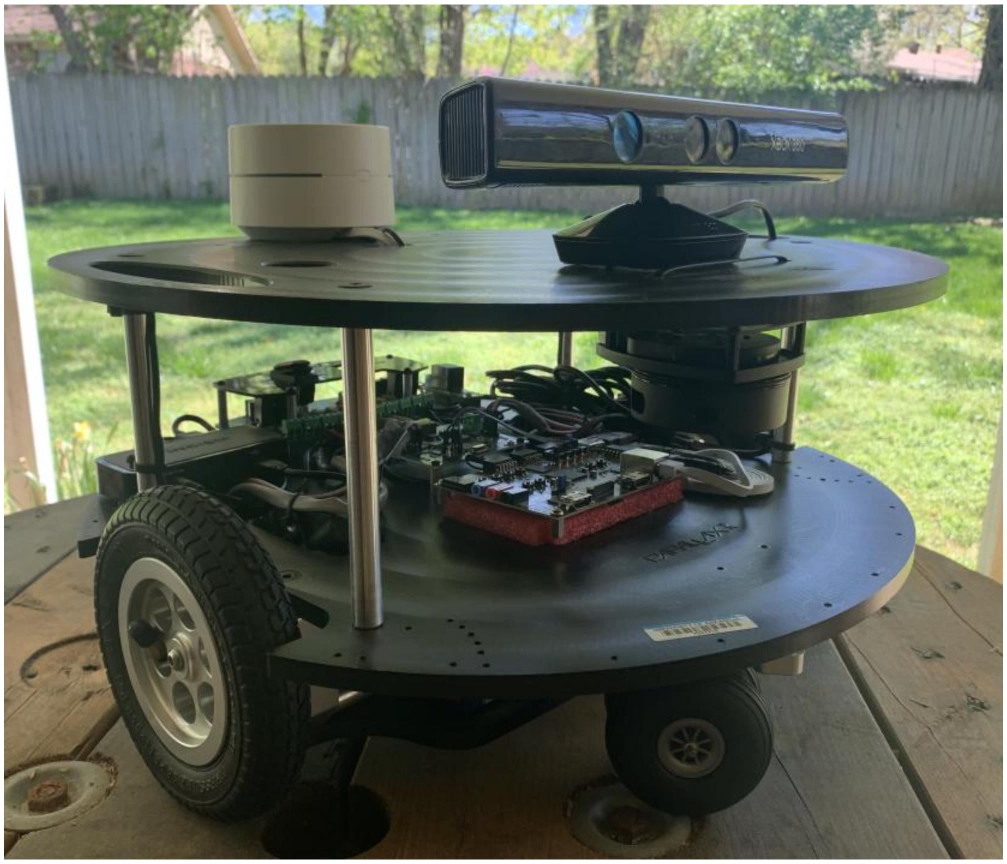


Figure 6.1: Arlo Demonstration Robot Final Implementation

mentary documentation for navigating the sources.

## 6.2 Platform Analysis

With all the capabilities added to the ADR platform, it is difficult to direct an evaluation of the robot. Focuses on odometry estimation accuracy, SLAM performance, or sensor and network bandwidth could all provide viable results for validating this platform, though no intentions to optimize these areas were set. For these reasons, this work delegates this in-depth analysis to future work. It is believed that this platform was in fact validated by the successful implementations of the reviewed commercial functionalities and a look at the developmental problems serves as an appropriate analysis.

### 6.2.1 Low-Level Components and Central Control Board

Though the Arlo Complete Robotic System, on which the ADR is based, provides all the necessary components for control of the motors, the DHB-10 motor controller presented a

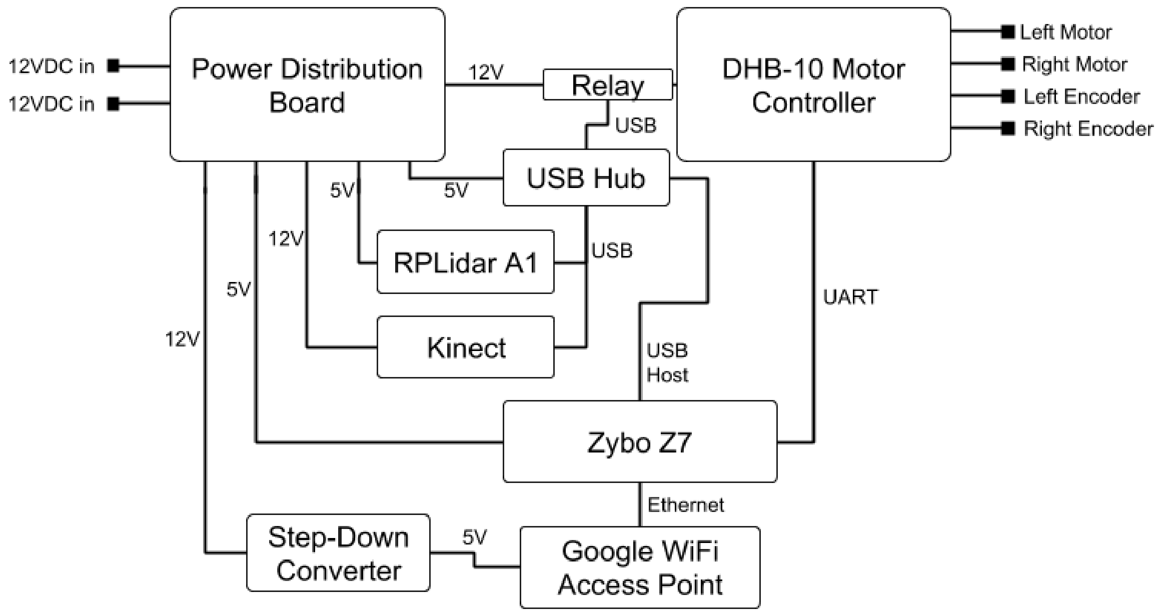


Figure 6.2: Schematic for Arlo Demonstration Robot

number of issues. Firstly, the motor controller firmware is quite particular about the format of commands and requests being issued to it. Attempts to interface with the motor controller were first done with the supplied Propellor ActivityBoard and proved to be successful. However, moving to a simple ESP8266 development board capable of UART communication initially failed. Attempts with a Linux serial program and USB to serial converter also initially failed, though, for both boards, the first command would always be accepted with proceeding inputs being ignored. This suggested a misformatted command. The issue was determined to be the requirement for a single carriage return character to be appended to the end of each command, a requirement not mentioned in the motor controller firmware documentation. Once the format was revised, no later communication issues arose.

The second issue faced with the motor controller was its tendency to enter an error state with no discernable cause. Upon entering this error state, the motor controller had to be manually reset with a power toggle. With the issue arising after the creation of the iOS teleoperation application, the presumed trigger was that the motor controller was now being sent a stream of movement commands rather than the sparsely issued commands of simpler teleoperation methods, like using the keyboard. Attempts at avoiding the error state all together failed, though a workaround was applied by adding a programmable relay to the motor controller power supply. This entailed the use of a simple ESP8266-enabled relay that accepted either an on or off command for closing or opening the switch. The relay enabled remote management of the motor controller and even enabled a motor stop function, though this fix should not have needed to be put in place.

The Zybo development board used as the central control board also added a few issues to be overcome. The first was the previously discussed issue of the UART1 serial controller being unable to receive commands of the attached PMOD pin. Utilizing a set of UART controllers was an adequate solution, though not ideal as it utilizes additional hardware space and requires two interfaces to be opened for software drivers. The Zybo has a fair amount of reference designs available for support, though they do not offer the customizations that the ADR required. As a result of having manually created a hardware design, some of the components available on the board were not enabled by default, such as the USB port. When building the boot images for both hardware and software with PetaLinux, the USB interface is not included in the hardware descriptor files used by PetaLinux. To enable USB, the necessary configuration file entries were borrowed from available Zybo reference designs and applied to the PetaLinux workspace. This enabled full access to the USB port from the processing system.

### 6.2.2 ROS Integrations

The extensive active development of ROS and the large variety of functional libraries contributed by the open-source community provides both benefits and downfalls. Many imaginable features have already been merged with ROS, though finding packages that support the latest releases can prove to be difficult. With ROS Melodic, the suggested navigation stack is not supported and thus alternatives must be chosen to fill the functional holes in the stack. For newcomers to ROS, this ecosystem can be daunting and give the impression of a much larger learning curve. The large availability of packages allowed for the development of this work, though the large number of ROS nodes exposed by even a simple system is quite large and can also be overwhelming to ROS amateurs. This work utilizes only a few ROS packages but the number of ROS processes can be extremely large when layering the platform's capabilities for some application. In retrospect, an additional objective of better organizing the exposed capabilities both in the developed ROS framework and in the discussion would have proven to be more conducive to those without prior ROS knowledge.

During the initial development of ROS, a direct ethernet connection facilitated the retrieval of ROS data of nodes running on the robot platform from a more powerful development machine. Two disrupting concerns developed at this stage. Firstly, the large amounts of data that were being issued at high-frequencies proved to put considerable traffic over the network connection. An ethernet connection could handle this traffic quite well allowing for minimal network overhead, however when considering remote operation, a wireless network setup could suffer from extreme congestion. The second concern arose from using the RViz visualization tool to display odometry, laser scans, maps, and images. With so much incoming data that must be processed and presented, development machines with minimal graphics performance could not keep up with load giving way to unresponsive systems that in some cases require a reboot.

In order to allow for remote management of the ADR platform, it was essential that it receive some sort of wireless networking component. Initial tests of the system utilized a simple WiFi USB dongle that was integrated into the Ubuntu system with the inclusion of additional kernel modules that enabled wireless networking and wireless extensions. With the development environment's network setup existing of a single central WiFi router, poor performance was immediately noticed as the development machine was wirelessly connected and positioned far from the router. With the mobile robot platform, areas with poor WiFi coverage were consistently affecting the network delay and thus affected the performance of ROS nodes requiring high bandwidth.

An obvious fix to the problem of dead-zones is the simple addition of more access points. However, this is typically not feasible in many scenarios and would only allow for better wireless reliability for the environment with the added access points. The root of the problem was the platform's initial dependency on a network setup being available. The solution this work determined was to make the mobile platform an access point itself. This allowed for a more powerful signal than was available with the WiFi dongle and removed any dependency on an external network. This was important for demonstrations, as it does not require any network modifications to be made in new environments. With the access point providing the wireless network on which other devices can interact with the robot, the Zybo no longer needed any additional network drivers beyond that required by ethernet. The chosen access point was the Google WiFi Wireless Mesh Router. It was powered via a 5V 3A step down voltage converter connected to an available auxillary 12V PDB terminal.

# Chapter 7
# Conclusion

This thesis intended to demonstrate the procedures of developing a prototype robotic platform, purposed for functional exploration and feature demonstration, and integrate a widely accepted solution for control. The resulting Arlo Demonstration Robot provides an extremely capable platform on which many areas of robotics could be explored and hosts a robust set of capabilities that can effectively demonstrate advanced features of mobile ground robots. In summary, the main contributions of this work are as follows, it:

- Established current state of the art in commercialized robotics
- Selected a subset of operation-critical capabilities seen in survey of state of the art
- Selected a robot platform suitable for demonstrating these capabilities
- Implemented a control system centering around ROS
- Effectuated the selected capability set
- Evaluated and demonstrated the robot platform
- Released an open-source ROS robot stack for the ADR platform

## 7.1 Contributions

Taking from the motivation of this work, the platform was intended to contribute to the effort of bridging robotics in industry and those in educational settings. Conducting a survey of the current state of the robotics industry allowed comparisons to be drawn from the various applications and features implemented in differing fields. From the brief review, the most utilized features involved the use of SLAM or equivalent methods for enabling autonomous navigation. With such varying degrees of complexity in the end applications, it was encouraging for this work to realize the underlying mechanisms were actually quite simple and mirrored efforts of the research community. A platform that could be built in a university or hobby setting that demonstrated the use of low-cost sensors and open-source libraries for standard commercial features would be a tremendously useful resource. The

Arlo Demonstration Platform does just that and establishes itself as an effective educational framework with its allowance for great flexibility in applications, components, and features.

This consideration for maximizing flexibility was present at all stages of development. Extending from software down to the hardware level, the ADR was outfitted with a control system that relied on the Zybo Z7 SoC board. A base hardware design was developed for the platform to serve as a reference design that enabled programmable logic and the Zynq powered processing system. This foundation of the control system enabled support for custom hardware implementations to complement software processes and ensure minimal restrictions on the applications the platform could host. Further, the adoption of the ROS communication framework opened up the possibility of easily integrating any of the community contributed functionality packages and enabled the extended resources and support of the ROS community.

Substantiating the versatility of the platform relied on the ability to easily demonstrate the features and capabilities available on the ADR. Fitting the ADR with behaviors such as greeting an audience or autonomous navigation of a presentation area used high-level functions working atop the underlying components of the system to perform meaningful demonstrations. Making available the source code and configurations for all of the components of the ADR also submits the platform to welcomed validation from the open-source robotics community.

## 7.2 Future Work

Hopes for future efforts towards the development of this platform were at the core of the motivation for this work. The ADR features a control system capable of much more than has been detailed in this thesis. ROS explorations alone could warrant many separate efforts, the available programmable logic has been made available though not utilized thoroughly, and the physical platform still has available space for more peripheral components. Additionally, ROS organizational philosophies influenced the development of the control system such that it could be easily integrated for other platforms outside of simple ground robots.

### 7.2.1 Hardware Utilizations

With the Zybo Z7 SoC at the core of the control system of the ADR, there is the possibility of utilizing the programmable logic for more than simple UART controllers as most of the PL is unused for the base ADR hardware design. Functionalities such as odometry could benefit from a hardware implementation. With a current software implementation, the serial interface to the motor controller must traverse the software stack and is susceptible to significant overhead. Reducing this overhead, and thus reducing the time between encoder readings and odometric calculations could reduce the error the accumulation rate of these calculations. Sensors such as the RPLidar used for this platform typically utilize a typical serial interface though provide a USB to UART converter to allow easy interfacing with many machines. However, utilizing a purely hardware interface to the scanner with a set of memory mapped registers would allow the overhead introduced by USB to be introduced. Further, with a single USB port on the Zybo, it would reduce the congestion on the USB bus for components with only a USB interface.

If any additions are made to the hardware design system, it would be extremely useful to allow for remote updates to the firmware as all systems will require maintenance with continued use. Not having to be physically connected to a platform for updates would be a nice demonstration for methods of reducing downtime. This would require some network interface that allowed the upload of new firmware. With the Zybo using a Xilinx Zynq-7000 series SoC, hardware, software and other data can be loaded from non-volatile memory, when connected to a network, using a Partition Loader or U-Boot.

Another option that is available with the use of the FPGA SoC is partial reconfiguration. As some of the functions implemented with this work are suitable for hardware acceleration, the PL space must be considered. If the hardware space were not sufficient for the intended custom IP, partial reconfiguration could be used to reconfigure the PL when needing a particular function. For example, custom drivers for peripherals could be enabled when using particular components and hardware repurposed for other uses when not in use. This capability would be significantly beneficial for a research platform as it can reduce the overall performance and overhead requirements of the control system.

### 7.2.2 Security

One majority concern for the commercial counterparts of this research that has not been discussed is the security of the system. Platforms for intended demonstration use in a variety of environments will be exposed to a number of scenarios that could expose vulnerabilities of the system. Additionally, security should be integrated as a demonstration itself as it is a prime concern in research and industrial efforts alike. For this system, the primary interface of concern is the wireless network. Using the onboard access point with support for the latest WPA2-PSK security, connections on the local area network (LAN) can be assumed to be safe. However, there are situations in which the ROS network of devices could extend beyond the LAN and require a traversal of the open internet. In these cases, the security of the connection can no longer be enforced by the access point's security measures and falls back to the security employed by the communicating processes themselves.

With the ROS communication framework based on plaintext messages, any that must be transmitted beyond the wireless LAN will be broadcast in plaintext. Making matters worse, ROS processes with internet exposed ports could be targeted by malicious parties and issued unwarranted control messages as there is no method of authentication of messages by default. Thus, one intended effort of future research is an in-depth analysis of further vulnerabilities of ROS and the implementation of defensive measures to allow for a truly secure ROS system that can exist on the open internet. With ROS packages such as ros_auth attempting to establish an authentication method for ROS nodes, it is apparent there is a need for efforts in this direction [69].

## 7.3 Conclusion

The Arlo Demonstration Robot prototype and corresponding support systems that were developed and discussed over the course of this work proved to be a worthwhile undertaking and in all resulted in an astounding success. With contributions to a personal foundation of understanding in robotics as well as to that of the open-source community, an additional educational resource has been made available to researchers and hobbyists alike. The success of this work provides a glimpse into the systems of robots in industry and enables such systems on platforms suitable for the classroom or workshop. It is the hopes of this work that the contributions made with the Arlo Demonstration Robot can encourage interest in and future development of the robots and systems that have become apart of daily life. After all, the versatility of educational tools available today will help to determine the robotic infrastructures of tomorrow.

## Bibliography

[1] International Federation of Robotics. *Executive Summary World Robotics 2018 Industrial Robots*. 2018. URL https://ifr.org/downloads/press2018/Executive_Summary_WR_2018_Industrial_Robots.pdf.

[2] International Federation of Robotics. *Executive Summary World Robotics 2018 Service Robots*. 2018. URL https://ifr.org/downloads/press2018/Executive_Summary_WR_Service_Robots_2018.pdf.

[3] Mordechai Ben-Ari and Francesco Mondada. *Robots and Their Applications*, pages 1–20. Springer International Publishing, Cham, 2018. ISBN 978-3-319-62533-1. doi: 10.

[4] Stanislav Ivanov, Craig Webster, and Katerina Berezina. Adoption of robots and service automation by tourism and hospitality companies. *Revista Turismo and Desenvolvimento*, 27/28, 2017.

[5] Amazon.com to acquire kiva systems, inc., Mar 2012. URL https://www.businesswire.com/news/home/20120319006569/en/Amazon.com-Acquire-Kiva-Systems.

[6] Nate Lanxon. Inside amazon's robotic fulfillment center, Nov 2018. URL https://www.bloomberg.com/news/articles/2018-11-06/inside-amazon-s-robotic-fulfillment-center.

[7] WIRED. High-speed robots part 1: Meet bettybot in "human exclusion zone" warehouses, Jul 2013. URL https://www.youtube.com/watch?v=8gy5tYVR-28.

[8] Jimmy Stamp. Fulfillment centers, robotic logistics, and the future of retail, Aug 2017. URL https://archpaper.com/2017/08/architecture-fulfillment-centers/.

[9] Brian Heater. Amazon built an electronic vest to improve worker/robot interactions, Jan 2019. URL https://techcrunch.com/2019/01/18/amazon-built-an-electronic-vest-to-improve-worker-robot-interactions/.

[10] OTTO Motors. Discover otto: The sdv designed for material handling, . URL https://ottomotors.com/resources/videos/video.

[11] Mobile Industrial Robots. Transport anything anywhere with mir robots, . URL https://www.mobile-industrial-robots.com/en/resources/mir-videos/transport-anything-anywhere-with-mir-robots/.

[12] OTTO Motors. Otto 1500, . URL https://ottomotors.com/resources/videos/the-otto-1500#.

[13] OTTO Motors. Otto 100, . URL https://ottomotors.com/otto100.

[14] Mobile Industrial Robots. What every business owner needs to know about the technology behind autonomous mobile robots, . URL https://www.mobile-industrial-

robots.com/en/resources/whitepapers/what-every-business-owner-needs-to-know-about-the-technology-behind-autonomous-mobile-robots/.

[15] Mobile Industrial Robots. Mir100, . URL https://www.mobile-industrial-robots.com/en/products/mir100/.

[16] Haydar Sahin and Levent Guvenc. Household robotics - autonomous devices for vacuuming and lawn mowing. *Control Systems, IEEE*, 27:20 – 96, 05 2007. doi: 10.1109/MCS.2007.338262.

[17] iRobot. Change the way you clean forever., . URL https://www.irobot.com/for-the-home/vacuuming/roomba.

[18] iRobot. irobot roomba 675, . URL https://store.irobot.com/default/roomba-vacuuming-robot-vacuum-irobot-roomba-675/R675020.html.

[19] Steve Crowe. irobot terra robot lawn mower hopes to jumpstart u.s. market, Jan 2019. URL https://www.therobotreport.com/irobot-terra-robot-lawn-mower/.

[20] Husqvarna. Robotic lawn mowers, . URL https://www.husqvarna.com/us/products/robotic-lawn-mowers/.

[21] Husqvarna. Husqvarna automower 310, . URL https://www.husqvarna.com/us/products/robotic-lawn-mowers/automower-310/967672966/.

[22] Jonathon Vanian. Five sci-fi robots that could revolutionize business, Sep 2015. URL http://fortune.com/2015/09/24/robots-business-retail-military-manufacturing/.

[23] KGO-TV. Sj hardware store uses robot to help customers, Dec 2014. URL https://abc7news.com/technology/sj-hardware-store-uses-robot-to-help-customers/419533/.

[24] Clint DeBoer. Lowes robot store assistant - is lowebot the future, or a bad idea?, Dec 2016. URL https://www.protoolreviews.com/news/lowes-robot-lowebot-store-assistant/27164/.

[25] Lowe's Innovation Labs. Robotics. URL http://www.lowesinnovationlabs.com/robotics.

[26] Lowe's Innovation Labs. Lowe's introduces lowebot, Aug 2016. URL https://www.youtube.com/watch?v=hP3yfGHTXFo&feature=youtu.be.

[27] Jonathon Vanian. Why walmart is testing robots in stores-and here's what it learned, Mar 2018. URL http://fortune.com/2018/03/26/walmart-robot-bossa-nova/.

[28] Savioke. Hospitality, . URL http://www.savioke.com/hospitality-1.

[29] Savioke. Logistics, . URL http://www.savioke.com/logistics.

[30] Savioke. The savioke approach to safety, Jul 2016. URL http://www.savioke.com/blog/2016/7/21/the-savioke-approach-to-safety.

[31] Douglas Gage. Ugv history 101: A brief history of unmanned ground vehicle (ugv) development efforts. *Unmanned Systems Magazine*, 13, 02 1970.

[32] G. N. Desouza and A. C. Kak. Vision for mobile robot navigation: a survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(2):237–267, Feb 2002. ISSN 0162-8828. doi: 10.1109/34.982903.

[33] Nicolas Varas. Wiki - move_base. URL http://wiki.ros.org/move_base.

[34] Josep Aulinas, Yvan R. Petillot, Joaquim Salvi, and Xavier Lladó. The slam problem: a survey. In *CCIA*, 2008.

[35] Rajesh Rao. Stereo and 3d vision, Mar 2009. URL https://courses.cs.washington.edu/courses/cse455/09wi/Lects/lect16.pdf.

[36] Richard I. Hartley and Peter Sturm. Triangulation. *Computer Vision and Image Understanding*, 68(2):146 – 157, 1997. ISSN 1077-3142. doi: https://doi.org/10.1006/cviu.1997.0547. URL http://www.sciencedirect.com/science/article/pii/S1077314297905476.

[37] Microsoft. Project natal 101, Jun 2009. URL https://web.archive.org/web/20120121223600/http://download.microsoft.com/download/A/4/A/A4A457B3-DF5D-4BF2-AD4E-963454BA0BCC/ProjectNatalFactSheetMay09.zip.

[38] Albert S. Huang, Abraham Bachrach, Peter Henry, Michael Krainin, Daniel Maturana, Dieter Fox, and Nicholas Roy. *Visual Odometry and Mapping for Autonomous Flight Using an RGB-D Camera*, pages 235–252. Springer International Publishing, Cham, 2017. ISBN 978-3-319-29363-9. doi: 10.1007/978-3-319-29363-9_14. URL https://doi.org/10.1007/978-3-319-29363-9_14.

[39] F. Endres, J. Hess, N. Engelhard, J. Sturm, D. Cremers, and W. Burgard. An evaluation of the rgb-d slam system. In *2012 IEEE International Conference on Robotics and Automation*, pages 1691–1696, May 2012. doi: 10.1109/ICRA.2012.6225199.

[40] M. Labbe and F. Michaud. Appearance-based loop closure detection for online large-scale and long-term operation. *IEEE Transactions on Robotics*, 29(3):734–745, June 2013. ISSN 1552-3098. doi: 10.1109/TRO.2013.2242375.

[41] M. Labbe and F. Michaud. Rtab-map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation. *Journal of Field Robotics*, 36(2):416–446. doi: 10.1002/rob.21831. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.21831.

[42] National Oceanic and Atmospheric Administration (NOAA) Coastal Services Center.

Lidar 101: An introduction to lidar technology, data, and applications, Nov 2012. URL https://coast.noaa.gov/data/digitalcoast/pdf/lidar-101.pdf.

[43] G. Grisetti, C. Stachniss, and W. Burgard. Improved techniques for grid mapping with rao-blackwellized particle filters. *IEEE Transactions on Robotics*, 23(1):34–46, Feb 2007. ISSN 1552-3098. doi: 10.1109/TRO.2006.889486.

[44] Matthias Gruhler. Wiki - navigation. URL http://wiki.ros.org/navigation.

[45] Jasprit Gill. Wiki - navigation robot setup. URL http://wiki.ros.org/navigation/Tutorials/RobotSetup.

[46] W. Hess, D. Kohler, H. Rapp, and D. Andor. Real-time loop closure in 2d lidar slam. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1271–1278, May 2016. doi: 10.1109/ICRA.2016.7487258.

[47] A. Nüchter, M. Bleier, J. Schauer, and P. Janotta. Continuous-time slam-improving google's cartographer 3d mapping. In F. Remondino, A. Georgopoulos, D. Gonzalez-Aguilera, and P. Agrafiotis, editors, *Latest Developments in Reality-Based 3D Surveying and Modelling*, pages 53–73. MDPI, Basel, Switzerland, January 2018. doi: 10.3390/books978-3-03842-685-1/4. URL http://www.mdpi.com/books/pdfdownload/article/515/1.

[48] ROBOTIS. Turtlebot3 manual. URL http://emanual.robotis.com/docs/en/platform/turtlebot3/overview/.

[49] iRobot. irobot create 2 open interface (oi) specification, Jul 2018. URL https://www.irobotweb.com/-/media/MainSite/Files/About/STEM/Create/2018-07-19_iRobot_Roomba_600_Open_Interface_Spec.pdf.

[50] Parallax. Arlo complete robot system. URL https://www.parallax.com/product/28966.

[51] Parallax. Dhb-10 arlo firmware guide v1.0, Apr 2016. URL https://www.parallax.com/downloads/dhb-10-motor-controller-firmware-guide.

[52] The MagPi Magazine. Sales soar: Raspberry pi british board beats commodore 64, world's third best-selling computer, Mar 2017. URL https://www.raspberrypi.org/magpi/raspberry-pi-sales/.

[53] Raspberry Pi. Raspberry pi 3 model b. URL https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/#buy-now-modal.

[54] Terasic Technologies. De main boards - de0-nano-soc kit/atlas-soc kit. URL https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&No=941.

[55] Digilent. Zybo z7 reference manual. URL https://reference.digilentinc.com/reference/programmable-logic/zybo-z7/reference-manual.

[56] Xilinx. Petalinux tools, . URL https://www.xilinx.com/products/design-tools/embedded-software/petalinux-sdk.html.

[57] Xilinx. Petalinux tools documentation - reference guide, . URL https://www.xilinx.com/support/documentation/sw_manuals/xilinx2018_3/ug1144-petalinux-tools-reference-guide.pdf.

[58] Morgan Quigley, Brian P. Gerkey, Ken Conley, Josh Faust, Tully Foote, Jeremy Leibs, Eric Berger, Rob Wheeler, and Andrew Y. Ng. Ros: an open-source robot operating system. 2009.

[59] chrisl8. chrisl8/arlobot, Mar 2018. URL https://github.com/chrisl8/ArloBot.

[60] Mike Purvis. Wiki - teleop_twist_keyboard, Jan 2015. URL http://wiki.ros.org/teleop_twist_keyboard.

[61] OMZ Software. Pythonista 3. URL http://omz-software.com/pythonista/index.html.

[62] Apple. Use siri shortcuts, Feb 2019. URL https://support.apple.com/en-us/HT209055.

[63] Apple. Use the shortcuts app on your iphone or ipad, Jan 2019. URL https://support.apple.com/en-us/HT208309.

[64] Adafruit. Slamtec rplidar a1 - 360 laser range scanner. URL https://www.adafruit.com/product/4010.

[65] Tony Huang. Rplidar-a1 360 degree laser range scanner - domestic laser range scanner. URL http://www.slamtec.com/en/lidar/a1.

[66] Steve Dent. Robotics, Dec 2014. URL https://www.engadget.com/2014/12/31/oroginal-kinect-discontinued/.

[67] Piyush Khandelwal. Wiki - openni_launch, Nov 2013. URL http://wiki.ros.org/openni_launch.

[68] Piyush Khandelwal. Wiki - rgbd_launch, Nov 2013. URL http://wiki.ros.org/rgbd_launch.

[69] Catharine McGhan. Wiki - rosauth, Nov 2014. URL http://wiki.ros.org/rosauth.