Design of Mobile Robot for use as a Teaching Platform for Autonomous Navigation and Obstacle Avoidance

By

Kyle Thompson

Submitted to the
Department of Mechanical Engineering
In Partial Fulfillment of the Requirements for the Degree of

BACHELOR OF SCIENCE IN MECHANICAL ENGINEERING

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2023

Authored by: Kyle Thompson
Department of Mechanical Engineering
May 15, 2023

Certified by: Harrison Chin
Instructor/Lecturer of Mechanical Engineering
Thesis Supervisor

Accepted by: Kenneth Kamrin
Department of Mechanical Engineering
Undergraduate Officer

Design of Mobile Robot for use as a Teaching Platform and for Autonomous Navigation and Obstacle Avoidance

by

Kyle Thompson

Submitted to the Department of Mechanical Engineering on May 18, 2023 in Partial fulfillment of the requirements for the Degree of Bachelor of Science in Mechanical Engineering

ABSTRACT

There is no end to the depth of learning in the field of robotics. However, we are many times limited by the methods we choose to teach ourselves and others. As a result, I studied the current teaching platforms used, along with several existing mobile robots available in the market, and found there was room for something more suitable for my coursework taken so far.

This thesis covers the design and implementation of a new autonomous mobile robot teaching platform, which has now been adopted by MIT's 2.12 Introduction to Robotics class. This involves the design requirements gathered from looking at what both students and instructors needed to improve their current platform for teaching many of the principles of robotics. The end result has been a successful, open-source mobile robot that is capable of a plethora of autonomous tasks and a high level of modularity—useful for applications in whatever designs are produced by students for the ever-changing term projects.

Thesis Supervisor: Dr. Harrison H. Chin

Title: Instructor/Lecturer of Mechanical Engineering

Acknowledgements

The author would like to acknowledge his supervisor Dr. Harrison Chin for all of his support and guidance, both through this research and the impact he has had on my education and growth as a student. The author would also like to acknowledge Joseph Ntaimo for his companionship during all of the late-night endeavors and his contributions to making this work possible. Finally, the author would like to thank his family for their never-ending support through his academic career.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Autonomous robots have been a subject of growing interest over the past few decades, specifically mobile robotics. With the emergence of countless applications for autonomous robot platforms, there have also been a plethora of interesting and challenging problems. These robotic platforms have provided solutions to many unique challenges and as such have made for valuable problems to solve across many fields of research. For instance, the recent discussion of lunar–and other extraterrestrial–development has again brought autonomous mobile robots to the forefronts of research for moving forward in the advancement of development beyond our planet. This comes in both the form of exploration/mapping of new environments and in physically assisting in shaping that environment autonomously in preparation for the arrival of later manned missions. For example, NASA is working on a new lunar rover for exploration on the Moon and other celestial bodies (see Fig. 1-1).[1]

Figure 1-1: NASA's Volatiles Investigating Polar Exploration Rover, or VIPER, is a mobile robot that will soon be used for resource mapping of the moon, mars, and beyond in preparation for human exploration.

Another area with growing interest in mobile robotics and autonomous navigation is the auto-mobile industry. In just the past few years, there has been immense growth of autonomous navigation. This motivation may come from the fact that more than 1.35

[1] (Chen, 2020)

million people die on roadways around the world each year, leading to the push to adopt autonomy into the auto-mobile industry in a way that can help increase the safety of those on the road[2]. This increase in autonomy enhanced driving can be seen by the myriad of companies and start-ups focusing around a host of different autonomous tasks and solutions for vehicles. Companies like Tesla have already augmented their cars with various self-driving features. For example, Tesla recently released a beta version of their enhanced autopilot that comes with a collection of new features. The autopilot-equipped vehicle can automatically stay in lane, match the speed of surrounding traffic, navigate lane changes and interchanges, automatically park with a single touch, or even autonomously navigate its way to find you in a parking lot[3].

Figure 1-2: Tesla's Model S which can include enhanced autopilot capable of auto-lane change, navigation of interchanges, and identifies and slows at stop signs and traffic lights

Behind all of these amazing applications of autonomous mobile robots is an array of sensors, as well as an abundance of algorithms to enable autonomous navigation. This can take the form of radar sensors, cameras, ultrasonics sensors, LIDAR, GPS, localization, odometry, or SLAM just to name a few.

This thesis focuses on the development of a mobile robot platform that is capable of undertaking a variety of autonomous tasks and more importantly, serve as a teaching platform to test our understanding of autonomy algorithms and other principles of robotics. Many of the complex problems seen in the field of robotics and autonomy, including space rovers and self-driving cars, can be boiled down to its principles which at a basic level can

[2] World Health Organization. (2018)
[3] *Autopilot and full self-driving capability: Tesla Support*

look just like a simple mobile robot. As someone who is always looking to learn in life, I hope that the work in this thesis can help others to keep learning. Moreover, I hope that the work outlined in this thesis can help to better teach the next generation of engineers who will continue to solve the challenges we will face in this world and in the field of robotics.

Chapter 2

Market Comparison

When this project started, one of the first things asked was: What's out there? What are some other people doing? Due to the popularity of robotics and autonomy, there are a multitude of mobile robot platforms and kits that can be purchased, each unique in some aspect. Similarly, due to the interesting challenges that autonomy poses and the degree to which a mobile platform allows for testing and teaching of such problems, there exist mobile robot platforms in academia. As a result, I began this project by looking at what was available, both commercially and academically, and it was during this search that I realized we needed something different. While there are many options, they all have their own drawbacks.

Figure 2-1: ROSbot 2 Pro, an autonomous robot platform with a sensor package and CPU conducive to a variety of jobs

There are many commercially available mobile robot platforms, such as the Robot 2 Pro[4] or Robotpark's Eva Robot[5]. These options all have their own CPU and GPU specifications as well as sensor package capable of many autonomous tasks. They also provide a quick and easy way to get started on a project involving autonomy; however, all of these options were either very expensive (like Clearpath Robotics Jackal[6]) or lack of

[4] *ROSbot 2 Pro*
[5] *Educational Mobile Robot Platform EVA - Basic Kit with sensors*
[6] *Jackal UGV - small weatherproof robot - clearpath 2022*

some desired features for teaching purposes. For this reason, it is cost-prohibited to acquire a sufficient number of robots to teach a large group of individuals, since we wish to promote a more hands on experience. Additionally, this large cost makes commercially available solutions inaccessible to institutions or programs with more limited funds. Furthermore, many commercially available solutions are difficult to scale down or are without many avenues to reduces costs. Another drawback when looking at commercial mobile robots, is in the many "blackbody" components. This means many parts of the hardware, drivetrain, electronics, software, or sensors are closed off to the user or packaged up in a way that makes it difficult to dig in and analyze, problem solve, or gain a complete understanding of what is going on. This can again come back around as another financial burden when components fail and result in purchasing entire new packages or collections of parts to fix even a small failure. Another common problem with commercially available products is the modularity of their platform, or lacking substance in their sensor package.

In addition to looking commercially, I also looked at other mobile robot platforms being used in academia. I looked at the robot used in *Robotics: Science and Systems* course at MIT, also known as RACECAR. This class uses a mobile robot as its teaching platform for fundamental ideas of robotics and autonomy[7].

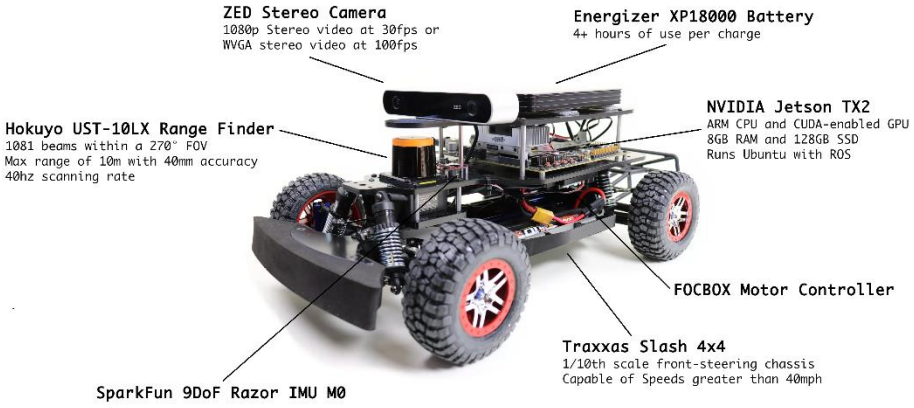

Figure 2 2: Current mobile robot platform used to teach and as a final project platform for students at MIT in Robotics: Science and Systems

[7] *Racecar*

While this design is great for testing autonomy algorithms in a project setting, it lacks in its ability for modularity and freedom of design. The tightly packaged system contains a variety of sensors useful in tasks such as SLAM, motion planning, and autonomous navigation. It also does so while remaining compact and neatly contained. However, this platform lacks much room for modularity or addition of student designs or improvements. Furthermore, its closed off style makes it difficult for students to entirely understand the system and its inner workings. This can prove to be difficult for students to solve problems that arise in hardware or electronics as well as lead to a lacking in their understanding of the physical components of their project.

Lastly, I looked at another mobile robot platform used in academia: the mobile robot platform from MIT's 2.12 *Introduction to Robotics* class.

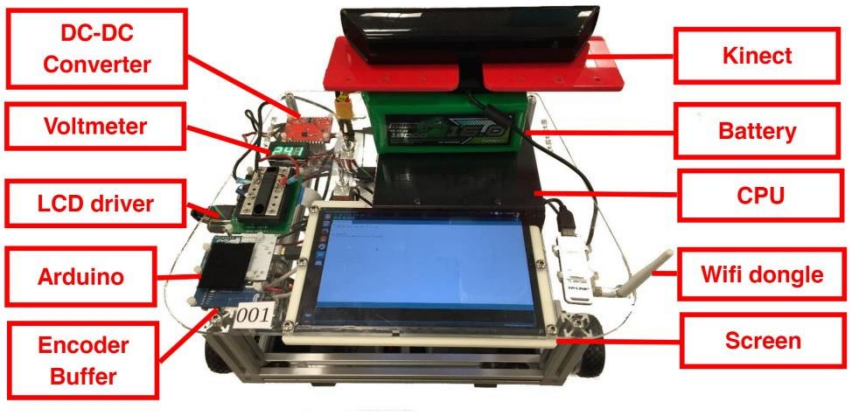

Figure 2-3: Previous mobile robot used in MIT's Introduction to Robotics class

This platform was developed in 2014 and used to teach fundamentals of robotics while giving a platform to be used by students in a final project. While this design allowed for the addition of student designs and modifications of the robot, it was outdated, heavy, difficult to troubleshoot, and—similar to the platform used in RSS—was too closed off to really give students a deep understanding of what makes up the robot they would work with. This again leads to difficulty in problem diagnostics/solving and a weakness in their understanding of the physical portion of the project.

Below is a table summarizing some of the key features of the aforementioned mobile robot platforms.

Platform	Key Features	Modularity	Price
ROSbot 2 Pro	• RGBD Camera • LIDAR • IMU • Distance Sensor • CPU+GPU	Very Low	~3,300
RACECAR	• RGBD Camera • LIDAR • IMU • WIFI • CPU/GPU (Jetson TX2)	Very Low	~3,500
2.12 Mobile Robot V1	• 3D Camera • CPU	Medium	~2,500
2.12 Mobile Robot V2	• RGBD Camera • LIDAR • IMU • Distance Sensor • CPU/GPU (Jetson Nano)	Very Modular	~1,800

Table 2-1: Table featuring some key components/features of the models compared

Chapter 3

Design Requirements

Before designing a solution to the problem at hand, it is useful to first lay out an intent of the proposed design along with requirements and constraints. This can help to streamline the process and ensure that an adequate solution is found. It is this method that has helped me to solve many problems I have encountered and I continue to use it as I come across new problems.

The design of this new mobile robot was motivated and influenced by a series of design requirements and intent that resulted in a design that falls within the requirements while trying to maximize the intent. These requirements and intent were motivated by the interests of MIT faculty involved in the teaching of the fundamentals of robotics, feedback from students, and by the comparison and analysis of other platforms currently available—see the previous section for comparison of other platforms.

The intent for this new mobile robot platform is multifaceted. The previous version for the mobile robot this project was constructed to replace, had reached the end of its useful life. Many of the components were obsolete: either unable to provide the performance required for the projects or no longer supported by newer software and components. And the components that weren't obsolete were wearing down and required extensive repair or replacement. This leads directly into the next two intentions for this project. This new mobile robot platform is intended to be a more modern solution, with more advanced mechanical and electrical components. Hence, the platform is intended to be capable for solving the robotic problems and projects of today, but more importantly, capable of meeting future demands. Furthermore, since a redesign was a topic of discussion, this brought about the perfect opportunity to incorporate feedback from the time spent using the previous version of the mobile robot platform. From this analysis, it was realized that this new version should be more modular and more open to the users. In other words, it was the intent of this new version to be capable of handling various tasks and projects. This means that the platform needs to be highly modular for additional

components, sensor packages, end effectors, or whatever mechanism is needed for the undertaking. Additionally, it was the hope of the design that it be constructed in a way that makes for an easier understanding of the various subsystems that make up the platform. This helps in both teaching the students, and instructing them in an easier debug process that hopefully can be carried out for other systems.

Motivated by these design intents for the new mobile robot platform and other outside factors, there was also a series of design requirements developed. In designing this platform, it was required to still be capable of performing the tasks of the previous version. This meant the new version must also be capable of carrying out the following: differential motion, vehicle kinematics, odometry, dead reckoning, visual navigation, and running ROS. Some of the new design requirements included: Simultaneous Localization and Mapping (SLAM) capability, self-driving, fully wireless, modularity, accommodation of larger sensor packages, payload capacity for larger mechanisms, and running the Robot Operating System (ROS) as well as ROS2.

On top of all of these design requirements for components and layout, there was a motivation to keep the cost low enough to still accomplish one of the main goals for this mobile robot. This mobile robot is intended to be an enhanced teaching too for the problems in robotics and autonomy. As a result, it is important that it is affordable enough to accommodate as many institutions as possible, as well as allow for enough platforms for students to get as much of an individual learning experience as possible within time and cost constraints. Keeping the price lower can allow for less students being required to share the platform, resulting in a more hands on experience. The motivation for modularity also comes from the ability for faster repairs and quick swap of components. This means less time spend decommissioned for the mobile robot, as well as giving institutions or programs the ability to add additional, or more expensive, components if it falls within their cost range. Overall, these new design requirements came out of a vision for the advancement in the future projects to be completed by the mobile robot and the broadening of the audience it can reach.

The hope behind his mobile robot is that by following the previously mentioned design intent and fulfilling the design requirements, this platform will be capable of

providing a better teaching experience. Moreover, the platform will be capable of testing more advanced autonomy algorithms and complete more complex tasks—both mechanically and algorithmically. These design considerations give the platform the ability to stay relevant for the ever-growing scope of problems in robotics.

Tables 3-1 and 3-2 summarize the design intents and requirements, based on market surveys and curriculum reviews mentioned before.

Design Intent
More modern platform
Increased modularity
More open layout for subsystems
Competitive pricing compared to market
Improve user interaction

Table 3-1: Table depicting a short summary of the design intent for the platform.

Design Requirements
ROS & ROS2
Differential Motion, Odometry, Dead Reckoning
Visual Navigation, Computer Vision
SLAM, self-driving
Fully Wireless
Large payload and sensor package accommodation

Table 3-2: Table depicting a short summary of design requirements.

Chapter 4

Hardware

Overview

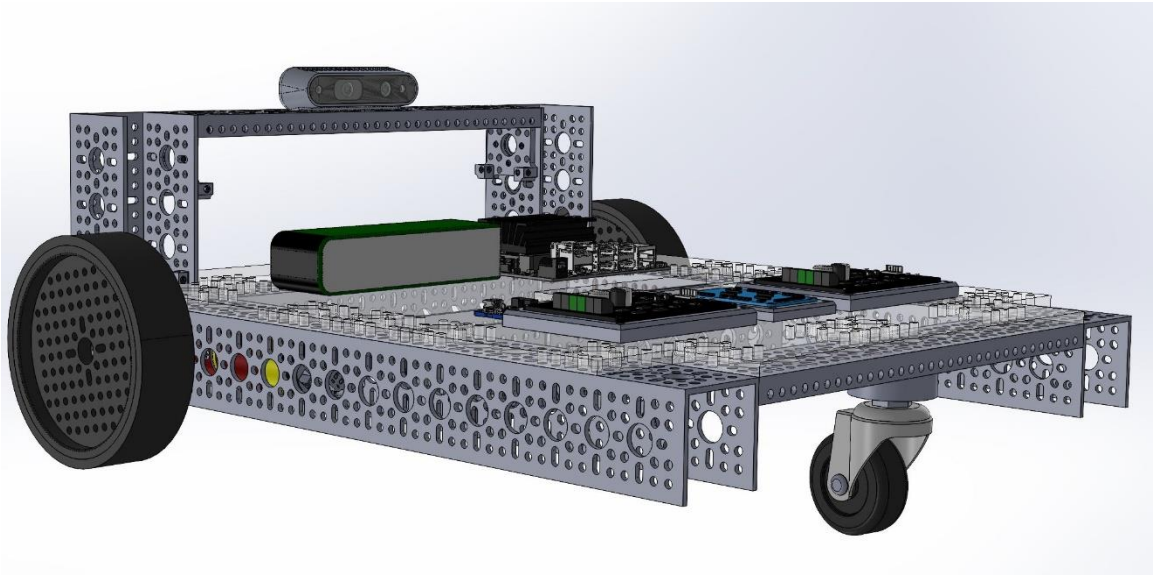

Figure 4-1: CAD of the core mobile robot platform. This contains the core components before added modules or more advanced sensor packages.

The new mobile robot platform is approximately 17”L x 14”W x 7”H. The mobile robot is split into effectively 3 layers. The first (bottom or underside) layer contains the drivetrain. The second layer contains the electrical components and brains of the vehicle. And lastly, the third layer is the layer ready to be equipped with additional modules, mechanism, sensor packages, as well as even room for additional layers. The previously mentioned measurement describes the height with one top layer, completely open for additional modules and mechanisms.

This division into three layers allowed for splitting the subsystems in a way that made it easy to isolate problems between subsystems, as well as more easily teach students the role of each individual system. It is also the intention that this will make any future redesigns or adjustments possible while minimizing the effect on other subsystems. Additionally, each layer of the platform was designed for as much modularity and open

source as possible. This means that each layer is capable of a wide variety of components and able to be adjusted or changed as the project sees fit. Whether it comes from the wealth of 3D printed mounts, the adaptation of the configurable goBILDA[8] pattern on the frame, or the highly customizable acrylic base layer—a component that also allows for a clear view of all the components.

Overall, this design originated from the previously mention design intent and requirements described in the previous section—Chapter 3. It is my hope that this design will serve as a great open-source educational robot platform for institutions to provide students with an excellent hands-on opportunity to solve and test the difficult problems in robotics, as well as conduct varying term projects.

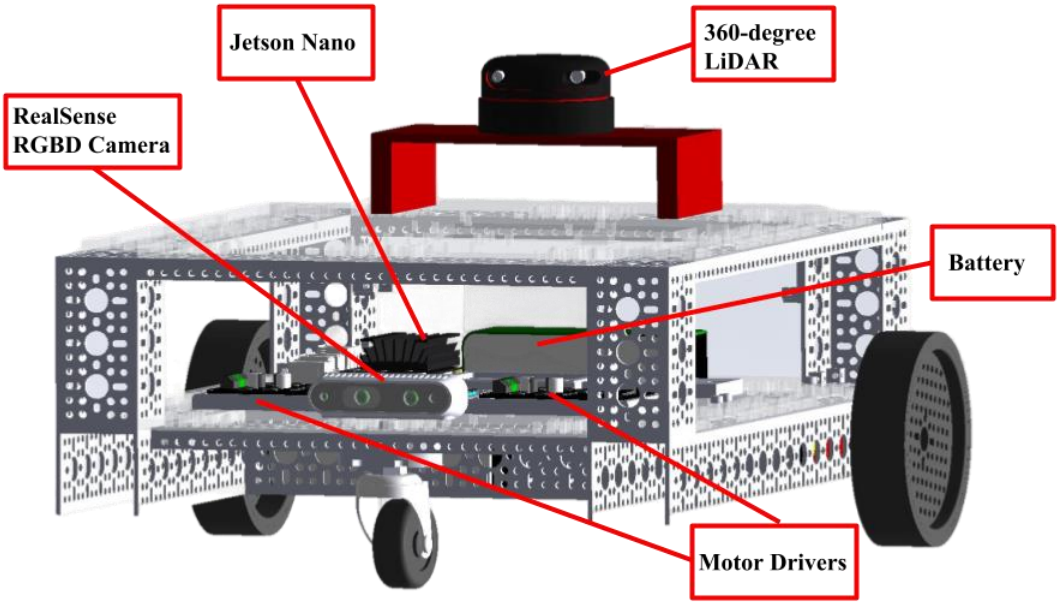

Figure 4-2: View of the intended standard mobile robot platform.

[8] goBILDA® is a registered trademark of https://www.gobilda.com/

Mechanical Chassis

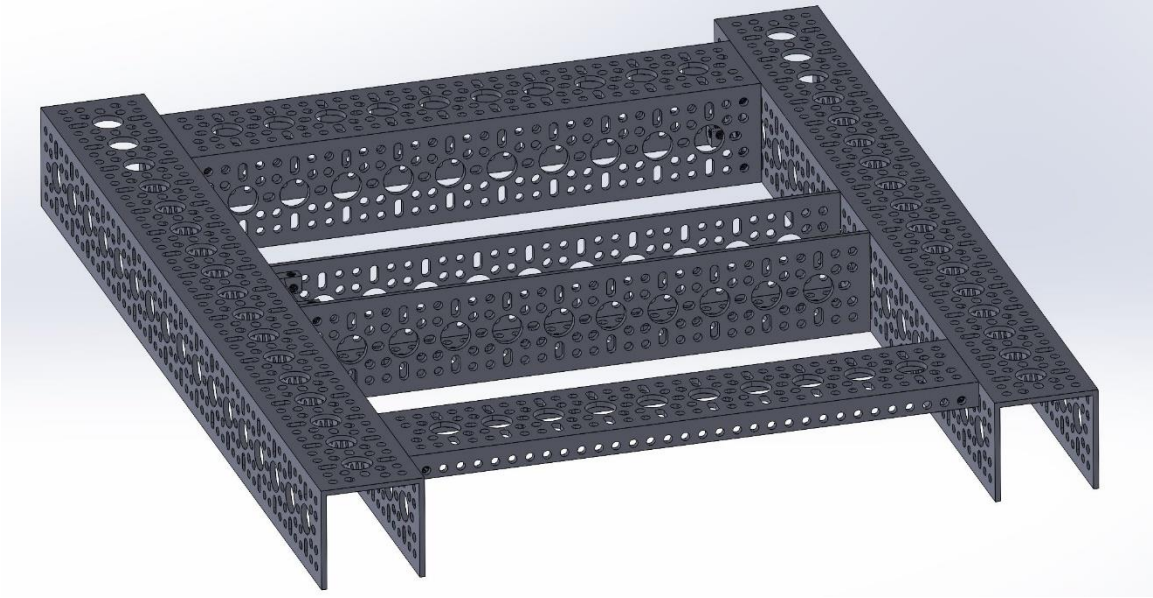

Figure 4-3: CAD of the mechanical chassis that makes up the mobile robot platform.

The mechanical chassis of the mobile platform is derived from the goBILDA Strafer[9] kit, with an inverting of the middle U-channel and the addition of one of their low-side U-channels added for additional mounting and stability. The frame was chosen from goBILDA components due to their unique mounting pattern—great for ensuring modularity—and for the ease of access in acquiring parts. The pattern's grid style, with cut-outs for bearings made for an excellent choice when going for a frame that allows for as much freedom in design and additions as possible. The pattern provides a consistent basis that allows for easy mounting when designing additional mechanisms or modules. Additionally, choosing a frame from a provider like goBILDA allows access to order parts in large quantities, with quick turnarounds, and makes acquiring spare and repair parts all the easier.

The U-channels that make up the frame make for easy compartmenting of components like the drive train, while keeping them protected. It should be noted though that the middle U-channel was inverted in order to allow for the safe storage of more sensitive components that could not risk the potential of collisions with any extrusions or

[9] Strafer™ is a registered trademark of https://www.gobilda.com/

obstacles from the ground while the mobile robot traverses. Additionally, a low-side U-channel was added to the front to allow for addition of low-profile components in the front. For example, in this first version specifically, it was added for the addition of a caster wheel mount, which could be used when teaching differential motion. However, the frame still allows for the use of a four-wheel configuration (driven or passive), again in hopes to allow for as many configurations as possible.

Drive Train

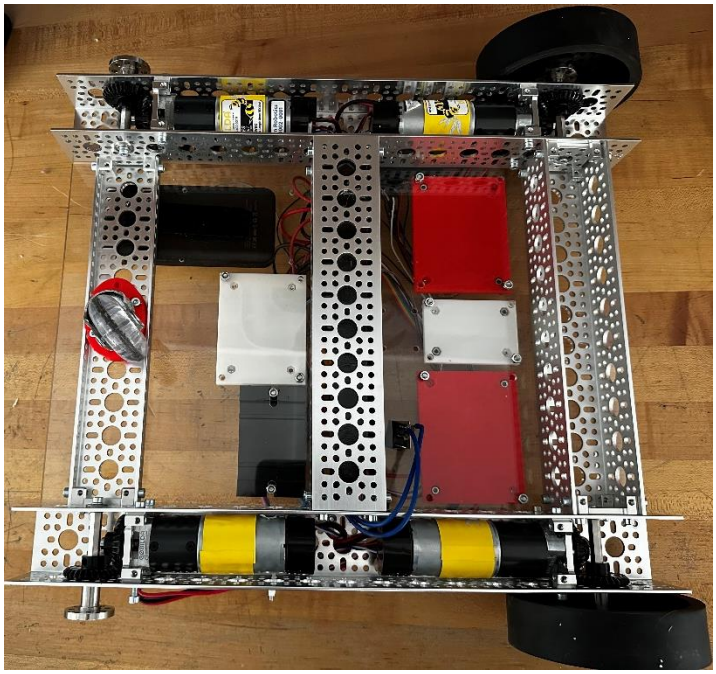

Figure 4-4: Image of the chassis and drive train for the mobile robot platform

The drive train for the mobile robot consists of anywhere from 2-4 planetary gear motors and a 12V battery, seated within the U-channels of the frame. These motors are powered off of the 12V battery which is seated between foam padding in the inverted middle channel. The motors convert the electrical energy from the battery into mechanical energy which is used to drive steel miter gears and 8mm hub-shafts, which are on ball bearings, that deliver the mechanical power from the motors to the wheels. This configuration allows for design freedom in the number of driven wheels used for the platform. The 90-degree position of the motors allows them to be nestled within the U-

channels, which provides a protected and compact design. Furthermore, it allows for a large amount of freedom in the choice of motors used in the drive train. For example, utilizing motors provided by goBILDA motors of the proper form factor can be chosen for high speed (lower torque) up to 6,000 RPM or high torque (lower speeds) up to 250 kg•cm. This again helps in providing a platform that is capable of being outfitted and adapted for a host of tasks, without need for any large redesign.

Chapter 5

Electrical Components

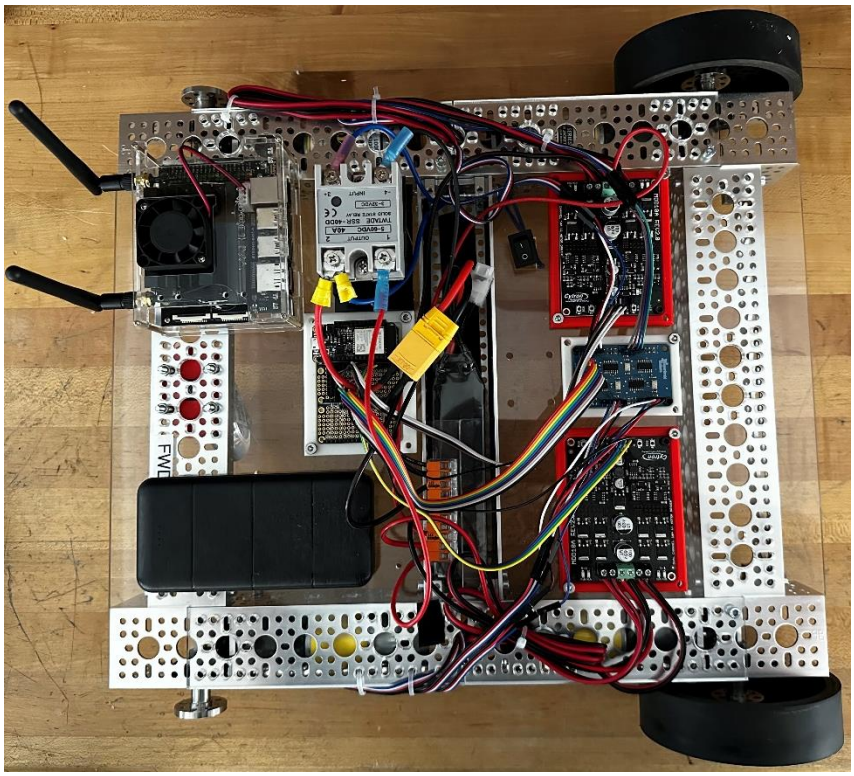

Figure 5-1: Image of a top down view looking at the second layer of the mobile robot.

The second layer of the mobile robot consists of the electrical components that make up this particular configuration of the platform. The electrical components lie on 3D printed mounts that attach to a custom cut acrylic plate. This acrylic plate is labeled where the components go, which again helps with creating a clear platform for students and individuals using the mobile robot to understand. The 3D mounts are also labeled to help clarify their use. Another key feature of this layer was the choice to keep as many subsystems together as possible. For example, the components involved with differential motion and dead reckoning are in the same area of acrylic to lead to a more intuitive layout. This helps with students understanding all the subsystems, as well as help in debugging or repairing. The 3D mounts on the custom acrylic also allow for any configuration of components, as well as an easy swap out of broken or outdated components. This again

helps increase modularity and the overall lifetime of the platform. The specific electrical components used in this build of the mobile robot are outline in the following sections.

Motor Driver

Figure 5-2: Shown is the motor driver used for driving the motors used on the platform.

The motors on the mobile platform are driven by Cytron's MMDD10A motor drivers[10]. The platform is equipped with two motor drivers, each capable of driving two motors. The drivers have a max continuous current of 10A and a peak of 30A—for 10 seconds—for both channels. This allows for great flexibility in completing higher torque tasks, including higher payload projects with the mobile platform. The driver supports voltages from 5V to 30VDC and is compatible with 3.3V and 5V logic input. This makes the component compatible with a host of microcontrollers. Another few useful features are the regenerative braking and the two activation buttons that allow for manual activation on each channel. These activation buttons allow for a much quicker check to test and confirm the working of motors, which can help to save debugging time on the software and hardware side of motor control. Overall, using two of these motor drivers allows for great flexibility in the range of tasks possible, as well as number of motors used (whether it be for four-wheel drive, or the addition of a motor in an added mechanism).

[10] *10Amp 5V-30V DC Motor Driver (2 channels)*

Encoder Breakout

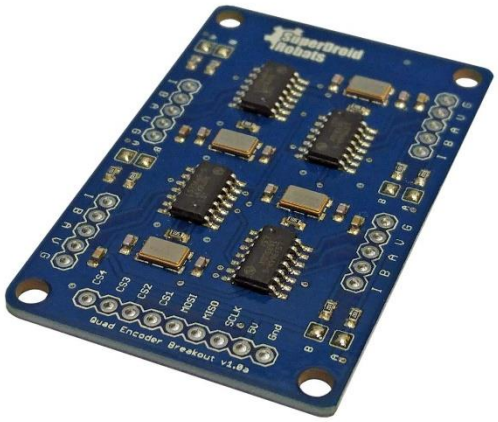

Figure 5-3: This image shows the quad encoder breakout board used for the mobile robot.

Another important electrical component on the new mobile robot is the Quadruple LS7366R Quadrature Encoder Buffer by SuperDroid Robots[11]. This breakout board interfaces with up to four encoder outputs, making it simple to perform operations that involve keeping track of encoder counts. For example, it makes for easy tracking of the angular position, speed, and total distance traveled by any of the four encoders attached. This makes the encoder breakout board a great asset for a range of tasks that could be expected of the new mobile robot, useful for the motors used in the drive train as well as on an additional mechanism attached to the platform. For example, this breakout board can be useful for providing fast and accurate encoder readings crucial for tasks involving some use of dead reckoning. Furthermore, this component helps to offload the work required of the onboard microcontroller, freeing it up for its other tasks. This is again helpful in keeping with the design intent of a platform capable of a wide and growing range of tasks, as well as remaining organized and easy to understand when used as a teaching platform.

[11] *Quadruple LS7366R quadrature encoder buffer*

Microcontroller

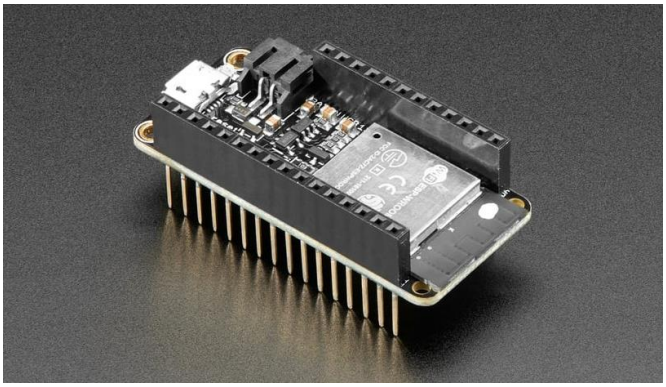

Figure 5-4: Shown is an Adafruit HUZZAH32, and ESP32 Feather Board, the microcontroller chosen for this robot platform.

The microcontroller used on this mobile robot is an ESP32-based Feather[12], by Adafruit. This is a dual-core ESP32 chip microcontroller fitted onto Adafruit's Feather eco-system. This makes for a powerful processor with 4 MB of SPI Flash, tuned antenna, and a layout that makes for easy quick swapping and testing while prototyping or designing for whatever project on hand. The ESP32 also has built in WiFi and Bluetooth, making wireless and/or internet-connected projects a possibility as well.

The Feather eco-system that this component makes use of allows for compatibility with any of Adafruit's 50+ Wings that allow for the addition of a host of other modules. This allows for added functionality that increases the flexibility in the projects possible. This form factor is compact, flexible, and can be used as a standalone module or the main board in a chain of additional components. These features made the HUZZAH32 an excellent choice that aligned well with design intent for a modular platform that is able to tackle an every growing set of projects and tasks.

The ESP32 is a change from the previous Arduino, specifically Uno, that was used in the mobile robot that came before this redesign. There are plenty of differences between the two; however, keeping with the intention to make a widely accessible teaching platform, the ESP32 can install an "ESP32-Arduino Core" which makes it possible to use the ESP32 as if it were an Arduino. This means it is also compatible with an Arduino IDE

[12] *Assembled Adafruit huzzah32 – esp32 feather board*

and can reuse most Arduino libraries in software written for the ESP32. While the ESP32 can practically be used as an Arduino, it allows for so much more. The ESP32 is cheaper than an Arduino Uno while being more powerful and capable. The ESP32 has built in Wi-Fi (acting as either an access point or Wi-Fi station) and Bluetooth. The ESP32 has a clock frequency between 160-240MHz, far greater than an Arduino Uno or Mega (around 16MHz). The ESP32 has more digital pins than an Uno and more analog pins than an Uno or Mega. The ESP32 also has 4MB of memory (15 times that of an Arduino Mega) and has lower power consumption. These are just some of the main differences between the commonly used Arduino line and the ESP32. Furthermore, the HUZZAH32 still comes in a smaller, lighter form while being cheaper than even an Arduino Uno.

Overall, the choice of the HUZZAH32 as this mobile robot's microcontroller was motivated by its ability to fall in line with keeping the platform highly modular and capable of taking on more powerful projects than what was used previously, all in a smaller and cheaper package.

Switch for Drive Train

Figure 5-5: Shown in this figure is a 5-60V Solid-State Relay and Heat Sink.

A solid-state relay by TWTADE[13] was chosen to be used to switch off power from the battery to the motors. One reason this was done was due to the availability and price at

[13] *TWTADE SSR-40 solid state relay + heat sink*

which the SSR could be acquired for. In addition, an SSR does provide some advantages over a mechanical switch. A solid-state relay has no moving parts and therefore can act quicker and last longer than a traditional manual switch. As a result, the SSR could be used as a cheaper, quicker, and longer lasting control over the potentially high amperage or voltage needed in the drive train or added mechanisms. This ability to handle a wide range of voltages and amperages at this low cost makes for increased flexibility when adding modules that require more power—without needing to redesign for a larger manual switch. Overall, the SSR allowed for a cheap solution that also aligns with the intent of an ever growing platform with increased modularity.

On board Computer

Figure 5-6: This image depicts the Jetson Nano Developer Kit used on the mobile robot

The onboard computer for this standard configuration is NVIDIA's Jetson Nano Developer Kit[14]. This kit has a 128-core NVIDIA Maxwell™ architecture GPU, Quad-core ARM® Cortex®-A57 MPCore processor and 4GB of memory. The kit can also utilize a microSD card or a USB flash drive for storage. The Jetson Nano provides a powerful on-board computer while still maintaining a small form factor. This allows it fit in well on the mobile robot platform, alongside all of the other electrical components. Additionally, the

[14] *Jetson Nano Developer Kit*

Jetson Nano is able to provide this performance with low power consumption—an important constraint when designing an autonomous mobile robot.

The Jetson Nano runs Ubuntu, a Linux-based operating system. This made it a great choice for using in robotics and autonomy projects. The kit is both ROS and ROS2 compatible which made it a great choice for completing tasks involving ROS, while still being open to future projects with ROS2. The Jetson Nano is one of the most popular devices for embedded AI tasks, computer vision applications, and other tasks in robotic projects. As a result, there is a lot of open-source resources and information that make this device a great choice for use on a teaching platform. This is useful for a teaching platform since students can expect to encounter similar on-board computers in other areas of the robotics field, making it a valuable kit to begin learning and testing their solutions. The powerful GPU also makes for a great asset when considering tasks that involve machine learning or AI-based computer vision tasks (like image classification or object detection). This was especially valued since many of the intended uses for this mobile robot include tasks involving computer vision software and machine learning libraries. This again ensures that the platform is able to complete a wide range of tasks and remain capable for tasks of growing complexity or growing reliance on machine learning and AI.

Overall, the Jetson Nano provides a powerful on-board computer for this mobile robot—that is capable of handling a wide range of autonomous tasks—all while remaining in a small form factor, low power consumption, and affordable price.

Chapter 6

Modular Mounts and Acrylic

One of the main design intents for this project was a platform that would emphasize modularity and flexibility, in order to adapt to whatever task at hand. One of the ways this was accomplished was through the use of modular 3D printed mounts, as well as a laser cut acrylic layer. These allowed for adaptability to a wide range of solutions, while minimizing difficulty.

The second layer of the mobile robot is home to the core electrical components that make up the robot. The backbone of this layer is a laser cut piece of clear acrylic, which allows for flexibility in the arrangement of components while still giving a feel of an open platform.

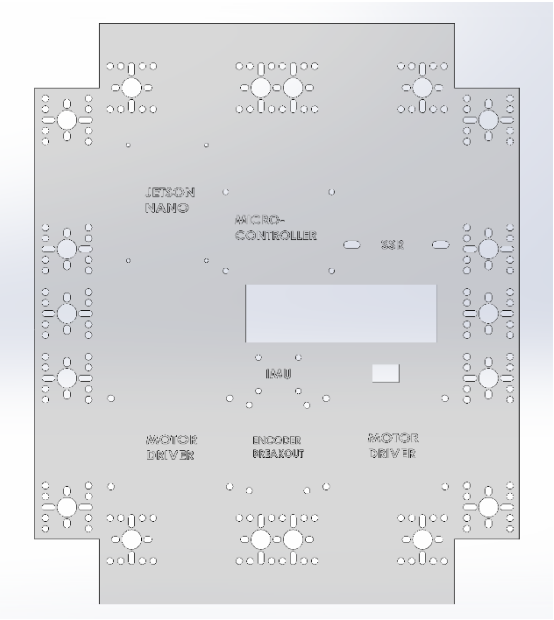

Figure 6-1: CAD of the laser cut acrylic layer with labeled engravings for the electrical components.

The acrylic layer allows for rapid prototyping of configurations, as well as quick additions or modifications. As a result, it makes for a great base that allows for adaptation in layout, components, or addition of future modules. It also allows for custom engraving, which can prove useful for labeling components and helping to keep subsystems organized.

This is especially useful as a teaching platform when students are trying to familiarize themselves with the layout and the various subsystems, for example, allowing for a clear labeling and sectioning of the components responsible for control of the motors and capable of dead reckoning. It also allows for students to use the layer as a template for their own custom acrylic layer, compatible with whatever mechanism they plan on adding to best fit their project. The acrylic layer follows the unique goBILDA pattern with a dilation for greater tolerance. This means the platform is still compatible with goBILDA's wide selection of components and structures. Another benefit from this use of an acrylic layer, is the ability to completely swap out an entire layer of the mobile robot if need be by simply having pre-configured acrylic layers available for troubleshooting or quick repair. Moreover, the open-source acrylic layer allows for the add-on of components with minimal redesign, the user simply needs to add mounting holes to the layout. It is also possible to have upgraded models of the mobile platform that simply need a different mounting pattern on the acrylic to facilitate its different layout of components. This again adds another layer of flexibility for the robot and its use as a teaching platform or project component at a variety of institutions.

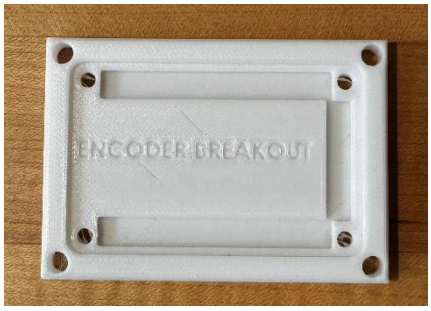

Figure 6-2: Picture of the 3D mount for the encoder breakout board.

Attached to the acrylic layer is a variety of custom 3D mounts for the electrical components that make up the system. This adds another layer of modularity by giving the ability to swap out single components through just changing one 3D mount. This means faster repairs and faster redesigns as updated components are needed in the future. This is a great asset during rapid prototyping and debugging. It allows for a quicker exchange of parts without needing to disable the entire platform. Furthermore, when components become outdated or the project at hand requires something more powerful, parts can be

quickly swapped out by just exchanging 3D mounts (meaning less overhauling of the design and platform).

The use of mounts also helps in organizing subsystems and components, especially with the ability to label the mounts. Components can be added with their 3D mounts as they become relevant to the coursework or project at hand. Additionally, the ability for clear labeling and different filament colors allow for an easier segmenting of subsystems—which can be very helpful when used as a teaching platform as students familiarize themselves.

Overall, the second layer of the mobile robot holds true to its intention of an adaptable and modular design that also allows for both flexibility in projects and clarity in its use as a teaching platform.

Chapter 7

Sensor Package

This standard design of the mobile robot includes sensors useful for completing a variety of autonomous tasks, as well as the ability to carry larger or more expansive sensor packages. This standard design in particular makes use of an Intel RealSense depth camera, rotary encoders, a time-of-flight sensor, and a basic LiDAR sensor.

Figure 7-1: Image of an Intel RealSense D435 Depth Camera.

One of the main sensors on the platform used for autonomy tasks is the Intel RealSense RGBD camera[15]. This sensor is great for a host of vision-oriented tasks or navigation. The D435 has the widest field of view of all Intel RealSense cameras, which makes it great for capturing as much of the mobile robot's environment as possible. The sensor is a stereo camera that provides depth data up to 10m, as well as color information. This makes it great for robotic navigation and object detection. The camera has an on-chip self-calibration that makes for easy calibration in les than 15 seconds without the need of specialized targets. This makes it a great choice for beginners in the stereo camera and vision task space.

Each of the four standard motors on the mobile robot platform have relative, quadrature encoders (Hall Effect sensors). For the 312 RPM motors used on the standard configuration, this makes for a 537.7 PPR at the output shaft, or 0.67° of resolution[16]. These sensors are useful in tasks that involve any nature of dead reckoning, or use of a mechanism

[15] *Depth camera D435*
[16] *5203 series Yellow jacket planetary gear motor 312RPM*

that must keep track of its distance traveled, speed, position, or any combination of this information.

Figure 7-2: Shown in this image is Adafruit's VL53L0X time of flight sensor.

A time-of-flight sensor is another sensor utilized on the standard configuration of the mobile robot; specifically, this design uses an Adafruit VL53L0X time of flight sensor[17]. This sensor utilizes a small laser and matching sensor to detect how long the light has taken to bounce back to the sensor, therefore giving the effective distance. This sensor uses a very narrow light source and as a result is very precise in determining the distance of the surface in front of it. This makes for a useful sensor in object detection and avoidance. The time-of-flight sensor doesn't suffer from linearity problems or "double-imaging" which make it difficult to tell if an object is far or close. This small, precise sensor is useful in a wide variety of autonomous projects or tasks.

Figure 7-3: Shown in the figure is SLAMTEC's RPlidar A2, a 360° laser range scanner.

[17] *Adafruit VL53L0X time of flight distance sensor*

The last sensor include in the standard design of the mobile robot is a 360° laser range scanner, specifically the SLAMTEC RPlidar A2[18], shown in figure 7-3. This sensor allows for accurate 360° mapping of the mobile robot's environment from 0.2-16M with a measuring accuracy of 2.5%. This is all possible in a small form factor and at a comparatively affordable price. This sensor is useful as it can act as the "eyes" for the autonomous robot, identifying the surrounding environment and its objects. This makes it useful for tasks such as SLAM, which is seeing a rapid growth in its use in the robotics field. As a result, it is useful to have this sensor on the mobile robot in order to be capable of performing a task like LiDAR based SLAM—generating a 3D map of the platform's environment. This again helps in making a platform that is adaptable and suitable for a wide range of autonomy tasks and teaching applications.

[18] *RPLiDAR A2*

Chapter 8

Lab Platform and Uses

One of the major motivations for this project, and initial use case, was as a lab platform for MIT's 2.12 Introduction to Robotics class. This was to replace the previous model which had reached its end of life—as described in Chapter 3—and serve as the teaching platform for the class. The new mobile robot was used as a platform for several labs that involve some of the core theories and techniques encountered in the realm of autonomy and robotics. Additionally, the mobile platform served as the basis for the final term project in the class. Both of these applications served as a great initial testing of the new design and its capability as an adaptive teaching platform, able to handle a variety of solutions.

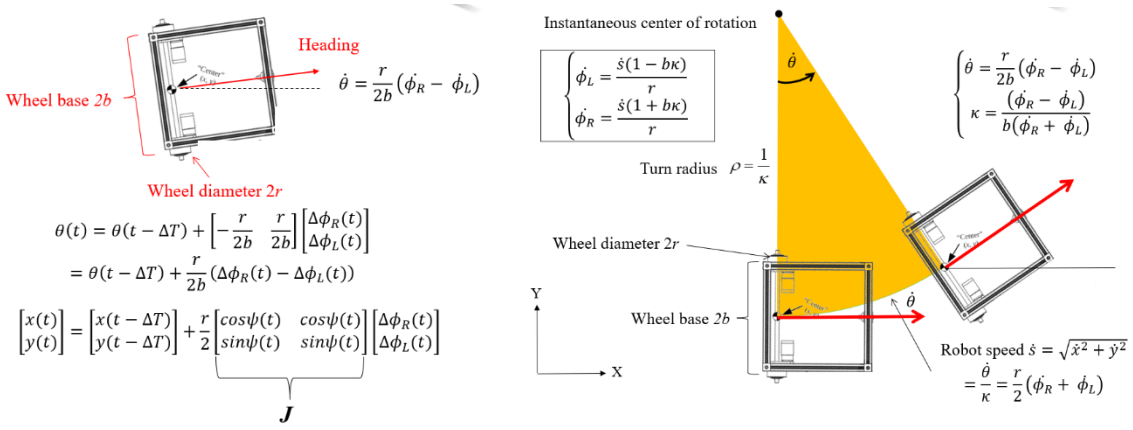

Figure 8-1: Example image of derived equations depicting the differential motion of the mobile robot as well as the vehicle kinematics following some arc.

One such lab in which the mobile robot proved useful was the lab teaching students how to implement differential motion and vehicle kinematics. Figure 8-1 shows the basic equations used in differential motion and vehicles kinematics. Students used a combination of differential motions and vehicle kinematics to generate a desired trajectory along with wheel velocities for the mobile robot. The trajectory was then followed using a dead reckoning estimate. This lab concluded with students commanding the mobile robot to

follow a U-shaped trajectory using a feed forward trajectory control based on the vehicle kinematics and generated wheel velocities.[19] This served as an example for students to draw upon when completing future autonomy stacks. Trajectory following, using odometry, can serve as one of the parts that make up a solution to an arbitrary autonomy task. For example, in the term project this platform was tested on, some students chose to use trajectories generated based on vehicle kinematics and dead reckoning as a core component for their autonomous solution to the project.

Another useful lab performed on the mobile robot is vehicle navigation using visual feedback. Specifically, the lab utilized AprilTags to estimate the robot pose in the world frame as well as communication on Robot Operating System (ROS). This can prove immensely helpful as both AprilTags and ROS are commonly used in the field of robotics for a range of autonomous tasks. As a result, this lab both teaches students key skills that can be adapted to their term project, as well as future projects, and shows that this mobile robot is capable of a wide, and popular, variety of objectives. AprilTags are 2D barcodes that can robustly detect 3D information of the tag with respect to the camera used for detection. There are many libraries that make it easy to implement AprilTag detection. Overall, AprilTag detection can be a quick, robust solution to many problems involving autonomous navigation or detection. For the lab involving vehicle navigation with AprilTags, the students use the on-board camera to scan for AprilTags. If the desired tag is detected, the vehicle effectively turns towards the tag and moves forward until it is the desired distance from the tag and is within some target region with respect to the robot's pose[20]. This lab gives the students some of the fundamental components for implementing an assortment of visual navigation tasks on the mobile robot.

Another tested use case for the new platform was in the final term project for MIT's 2.12 Introduction to Robotics. Students were given a project that involved the autonomous retrieval and drop-off of an item with unknown obstacles scattered around the environment. The mobile robot platform proved successful in completing the autonomous task and more

[19] Appendix A*: 2.12 Introduction to Robotics Lab 7*
[20] Appendix A: *2.12 Introduction to Robotics Lab 9*

importantly allowed for a myriad of unique solutions. Figure 8-2 below shows one group's mobile robot with added mechanisms.

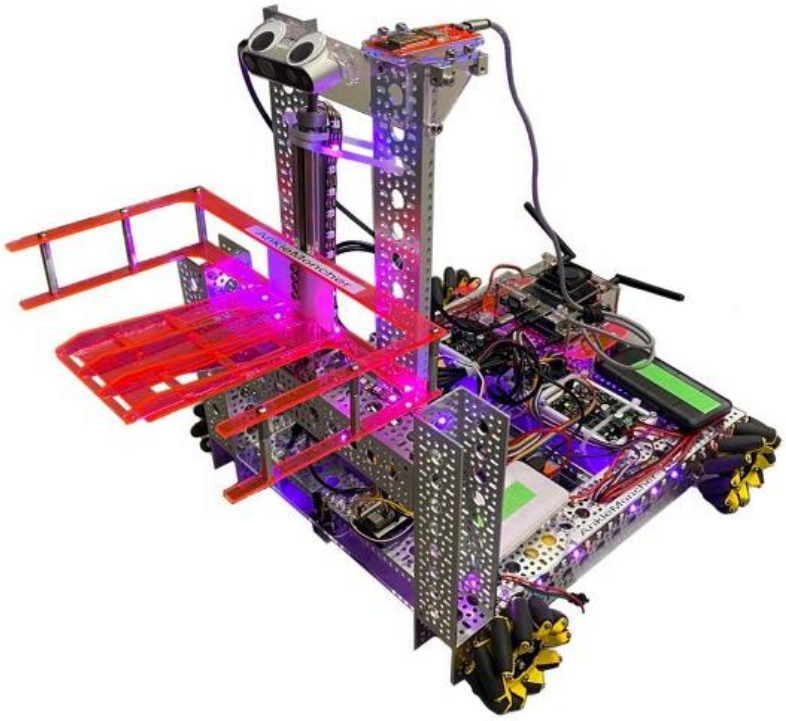

Figure 8-2: Image of one lab team's outfitted mobile robot for their term project.

The mobile robot was successful in providing a modular platform capable of facilitating a wide array of solutions to this terms project. For this one project, each team had its own unique addition of mechanisms or utilization of provided sensors. All of the added mechanisms were easily integrated on the platform due to its modular design. Moreover, teams were able to rapidly prototype a variety of mechanisms they designed on the actual platform before settling on their final design. This was again possible due to the adaptability designed into this platform. Many of the teams utilized additional goBILDA components as they promote modularity in their own right and interface well with the frame of the platform (which is also made of goBILDA parts as described in Chapter 4). The diverse solutions used on the mobile robot helps to increase confidence in the ability for this platform to complete a wide range of objectives.

This platform was tested as a teaching platform for some of the core mobile robotics labs in MIT's 2.12 Introduction to Robotics. This platform was able to fully demonstrate the labs and show students the possibilities of using those "building blocks" of autonomy

principles into a final term project. In the final term project, it was shown that this mobile robot is entirely capable of providing a platform that can be used to accommodate a large array of solutions. The designed flexibility and increased potential of this mobile robot platform—as well as its modularity—allowed for the addition of variously unique mechanisms to complete the project's objective. Overall, this new platform proved to be a great asset to the class and the students' learning/testing of autonomy in mobile robotics. Moreover, this design showed that it is capable of handling an every growing set of potential tasks demanded in future term projects.

Chapter 9

Conclusion

The mobile robot design described in this thesis proved to be a successful teaching platform and capable of adapting to a wide array of applications. This design incorporated a higher degree of modularity than other available platforms, as well as the ability to continuously upgrade its components without major redesign. Moreover, the design proved to work well as an instructional tool, giving students an understanding of the different subsystems that make up the entire robot, as well as working as a test bed for learning key autonomy solutions. All of this was provided in a design that is open-source and cost-effective, making it a viable option for other institutions—or even hobbyists wishing to learn some of the fundamentals of robotics through their very own projects. The affordability of the design also leaves room for the upgrade of components and for adding more expensive sensor packages. This design is capable of completing a host of different projects involving autonomy or teleoperated solutions, and is able to stand the test of time.

Future Work

Although this design effectively accomplished the design intent while completing the design requirements, there are still areas for improvement. For example, using a more powerful on-board computer like a Jetson TX2[21] would help to make this platform a viable solution for even more computationally intensive projects. This was the initial plan for the design, however the Jetson TX2 was unavailable at the time of purchasing. It is the hope that a future iteration could swap out the Jetson Nano for the Jetson TX2, which would incur a price increase. However, the price would still remain below the other commercially available platforms and would give a beneficial increase in computation power. A similar

[21] *Jetson TX2*

avenue to investigate would be an on-board computer with an Intel processor since there were issues with compatibility for certain libraries that relied on an Intel processor. Another potential alteration could be changing to a manual switch for switching the drive train on and off. This could be done to free up more space on the second layer (the layer with the other electrical components), although it could have its drawbacks in decreased lifetime due to moving parts. Furthermore, I am looking into adding a snap-in feature for the 3D printed mounts which would be an interesting addition as it would allow for the choice between a more stable bolted option, or a quick connect and disconnect mode—for say more rapid prototyping and debugging versus a more locked in configuration before a final design. On a similar note, I think utilizing heat insert threads would greatly benefit the 3D printed mounts as opposed to the captive nuts currently used, allowing for a smoother removal and attaching of components. Another useful change to investigate would be a more concise and organized wire management system. This would help prevent common confusions with wiring and decrease debugging time from miswiring. Finally, it could prove beneficial to look into creating a custom PCB board for the platform. This could result in a more concise design, as well as give more freedom in the layout and organization as well as customization as a whole.

References

10Amp 5V-30V DC Motor Driver (2 channels). Cytron Technologies. (n.d.). https://www.cytron.io/p-10amp-5v-30v-dc-motor-driver-2-channels

5203 series Yellow jacket planetary gear motor (19.2:1 ratio, 24mm length 8mm Rex Shaft, 312 rpm, 3.3 - 5V encoder). goBILDA. (n.d.). https://www.gobilda.com/5203-series-yellow-jacket-planetary-gear-motor-19-2-1-ratio-24mm-length-8mm-rex-shaft-312-rpm-3-3-5v-encoder/

Autopilot and full self-driving capability: Tesla Support. Tesla. (n.d.). https://www.tesla.com/support/autopilot

Chen, R. (2020, February 5). *Viper mission overview*. NASA. https://www.nasa.gov/viper/overview

Depth camera D435. Intel® RealSense^{TM} Depth and Tracking Cameras. (2022, December 5). https://www.intelrealsense.com/depth-camera-d435/

Educational Mobile Robot Platform EVA - Basic Kit with sensors. Robotpark Global. (n.d.). https://www.robotpark.com/Educational-Mobile-Robot-Platform-Eva-BASIC-KIT-With-SENSORS

Industries, A. (n.d.-a). *Adafruit VL53L0X time of flight distance sensor - ~30 to 1000mm*. adafruit industries blog RSS. https://www.adafruit.com/product/3317?gclid=CjwKCAjw04yjBhApEiwAJcvNoQMLxyy3kqcnPxJ9mWWoul39g-59C4Rr9FRBjtNNRifNoajABT28_hoCprcQAvD_BwE

Industries, A. (n.d.-b). *Assembled Adafruit huzzah32 – esp32 feather board*. adafruit industries blog RSS. https://www.adafruit.com/product/3619

Jackal UGV - small weatherproof robot - clearpath. Clearpath Robotics. (2022, December 20). https://clearpathrobotics.com/jackal-small-unmanned-ground-vehicle/

Jetson Nano Developer Kit. NVIDIA Developer. (2022, September 28). https://developer.nvidia.com/embedded/jetson-nano-developer-kit

Nvidia Jetson TX2: High performance AI at the edge. NVIDIA. (n.d.). https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-tx2/

Peng, M. (n.d.). *RPLIDAR A2*. SLAMTEC Global Network. https://www.slamtec.ai/home/rplidar_a2/?gclid=CjwKCAjw04yjBhApEiwAJcvNoaq-iJ4SNWLrW9p5FNknOsyYZIKCgBN7Q8Zcaukk7qBlMhd9znolcxoCZv0QAvD_BwE

Quadruple LS7366R quadrature encoder buffer. (n.d.). https://www.superdroidrobots.com/store/robot-parts/electrical-parts/encoders-accessories/buffer-pull-up-boards/product=2418

Racecar. racecar. (n.d.). https://racecar.mit.edu/

ROSbot 2 Pro. Husarion Store. (n.d.). https://store.husarion.com/products/rosbot-pro

Twtade SSR-40 DD 40A DC 3-32V to DC 5-60V SSR solid state relay + heat ... (n.d.). https://www.amazon.com/TWTADE-SSR-40-3-32V-SSR-25VA-Potentiometer/dp/B09VYSNMGC

World Health Organization. (n.d.). *Global status report on road safety 2018*. World Health Organization. https://www.who.int/publications-detail-redirect/9789241565684

Appendix A

Lab Handouts

Lab 7

2.12: Introduction to Robotics
Lab 7: Mobile Robot Path Tracking using Odometry *

Spring 2023

Instructions:

1. **When your team is done with each task, call a TA to do your check-off.**
2. **No need to turn in your code or answers to the questions in the handout.**

1 Introduction

Today's goals are to:

1. Estimate robot's change in pose $(\Delta x, \Delta y, \Delta\theta)$ using the encoders in the motors of the wheels.
2. Move the robot using delta pose $(\Delta x, \Delta y, \Delta\theta)$ commands.

To do so, we will apply the wheeled-robot kinematics from lecture. In this handout, we will denote the mobile robot platform as robot.

2 Setting up the code

Open a terminal window (Ctrl+Alt+T for Windows/Linux users and Ctrl+Opt+T for Mac users) and enter the following commands.

```
cd ~                              # note: make sure we are at home folder
git clone https://github.com/mit212/mobile_robot_2023.git
```

Alternatively you can go to the github site and download all the necessary files.

3 Instructions

1. Open the project with PlatformIO IDE in VScode.

*

1. Version 1 - 2016: Peter Yu, Ryan Fish and Kamal Youcef-Toumi
2. Version 2 - 2017: Yingnan Cui and Kamal Youcef-Toumi
3. Version 3 - 2019: Jerry Ng
4. Version 4 - 2023: Joseph Ntaimo, Kentaro Barhydt, Ravi Tejwani, Kamal Youcef-Toumi and Harrison Chin

2. Complete the `TO DO` lines in `pathPlanner.cpp`.

3. Select `env:robot` before uploading the code.

4. To run the code on the joystick, joystick.cpp needs to be in the joystick folder, and make sure to select `env:joystick` before uploading the code.

5. For wireless communication:

   (a) Get the MAC address of the robot and put it in the joystick
   - Drag all the files in the `src/robot` folder out into `src/`
   - Drag `get_mac.cpp` into the robot folder
   - Set the environment to robot
   - Upload
   - Open Serial monitor
   - Use the address printed in the serial monitor to fill in the broadcast address in line 22 of the `src/joystick/joystick.cpp` file:

   (b) Get the MAC address of the joystick and put it in the robot (similar process to that above)
   - Drag all 5 robot files back into the robot folder
   - Drag all the files in the `src/joystick` folder out into `src/`
   - Drag `get_mac.cpp` into the joystick folder
   - Set the environment to joystick
   - Upload
   - Open Serial monitor
   - Use the address printed in the serial monitor to fill in the broadcast address in line 16 of the `src/robot/wireless.cpp` file:

   Once it is done you can open the Serial Monitor and observe the data being sent over from the robot.

4 Code modifications for the tasks

In this lab, you will need to complete five tasks (described in the section below). These are the functions that you need to modify following functions in `src/robot/pathPlanner.cpp`

1. setDesiredVel() : Convert the desired velocity in linear velocity and curvature, κ, to angular velocities to send to the motor.

2. getSetPointTrajectory() : Change the velocity and κ curvature setpoints to make the robot follow a trajectory.

3. updateRobotPose() : Use the odometry readings to update the current position of the robot relative to the world frame.

Few things to consider:

- Task 1 and 2: Once this is done correctly, you should be able to leave the drivetrain off and push the robot around and get odometry readings that make sense. If you want to update the way the data is printed, change printOdometry() on line 162.
- Task 3 and 5: Robot trajectory tracking : This will be filling out getSetPointTrajectory(). You will need to actually set the variables `vel` and `k` based on how much of the path they have traversed and/or how much theta has changed. Make sure to run `setDesiredVel(vel,k)` at the end of the function. Otherwise the velocity won't actually change.
- When running the robot, all of the data will be sent over to the remote PC and printed over the serial port if the remote's broadcast address was correctly put into the robot's `wireless.cpp` file. This should make it much easier to debug.
- If you wish to control the robot with the joystick again, comment out `getSetPointTrajectory()` in the main loop() and switch to `getSetPointJoystick()`, where they have to figure out how to maps the `joyData.joyX` and `joyData.joyY` variables from their initial range of 0-1024 to the robot's desiredVelocity range of -30 to +30 rad/s.
- When running the robot, place it on the ground, turn on the drivetrain switch, then press reset on the microcontroller to make it move.

5 Task 1 & 2: Mobile Robot Odometry

Odometry is to estimate the change in robot pose over time using changes in encoder values. First, let's define some variables and constants:

- (x, y, θ): estimated robot pose using odometry relative to the world frame (starting frame),
- ϕ_R, ϕ_L: the right and left wheel net rotation (positive sign is in the robot forward direction),
- $b = 0.2$m, $r = 0.06$m, and $a = 0.3$m: robot dimensions as shown in Figure 1.

Motor 1 drives right wheel, and motor 2 drives left wheel.

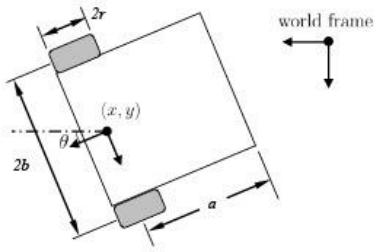

Figure 1: Definition of the robot dimensions and robot frame. Robot forward direction matches the x axis of the robot frame. Note that since we use the caster wheel as forward direction, the x axis is reversed in this figure.

To find the odometry, encoder values need to go through 2 conversions:

1. encoder count change to wheel position change;

2. wheel position change to robot pose change.

Recall from the lectures that we can first compute $\dot{\theta}$ using the following expression.

$$\dot{\theta} = \frac{r}{2b}\left(\dot{\phi}_R - \dot{\phi}_L\right). \tag{1}$$

However, in our robot platform, we can only measure ϕ's at discrete timestamps. Thus we rewrite (1) as (2):

$$\Delta\theta = \frac{r}{2b}\left(\Delta\phi_R - \Delta\phi_L\right). \tag{2}$$

To estimate current robot orientation $\theta(t)$ we use:

$$\theta(t) = \theta(t - \mathrm{d}t) + \Delta\theta. \tag{3}$$

Now we can use $\theta(t)$ to compute the position of the robot.

$$\begin{bmatrix} x(t) \\ y(t) \end{bmatrix} = \begin{bmatrix} x(t - \mathrm{d}t) \\ y(t - \mathrm{d}t) \end{bmatrix} + \frac{r}{2}\begin{bmatrix} \cos\theta(t) & \cos\theta(t) \\ \sin\theta(t) & \sin\theta(t) \end{bmatrix}\begin{bmatrix} \Delta\phi_R(t) \\ \Delta\phi_L(t) \end{bmatrix} \tag{4}$$

All relevant variables have been defined in `updateRobotPose(dPhiL, dPhiR))` and are listed below. You do not need to define your own variable. The following variables are of `float` type.

Question 1 *Is this approximation accurate? What is a way to improve this approximation?*

- `X`, `Y`, `Th`: $x(t)$, $y(t)$, $\theta(t)$,
- `dX`, `dY`, `dTh`: $\dot{x}(t)$, $\dot{y}(t)$, $\dot{\theta}(t)$,
- `pathDistance`: keeps track the total distance traveled by your robot.

6 Task 3 & 4: Robot Trajectory Tracking

The task is to program a navigation policy to follow a U-shaped trajectory and stop at the goal position. An illustration of the track is shown in Figure 2.

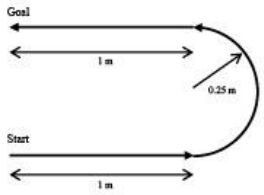

Figure 2: The U Trajectory.

To do so, we'll implement a policy in `callback(self, msg)` and a function `PathPlanner(self, robotVel, K)` in `planner.py`.

Here are three steps to complete this task:

1. Determine what stage of the track your robot is in. We suggest to write an `if/else` statement in `getSetPointTrajectory()` function. The U trajectory can be divided into 3 stages: 1) straight line, 2) semi-circle, and 3) straight line again.
2. Specify a robot velocity and motion curvature for each stage.
3. Compute the desired R/L wheel velocities from a specific motion velocity and curvature. A function `setDesiredVel(float vel, float k)` for this purpose has been declared for you in `pathPlanner.cpp`.

You need to complete this function using the following two equations.

$$\kappa = \frac{\dot{\phi}_R - \dot{\phi}_L}{b(\dot{\phi}_R + \dot{\phi}_L)}, \tag{5}$$

where κ is the curvature, the inverse of the radius of a circle, as shown in Figure 3.

$$\texttt{robotVel} = |(\dot{x}, \dot{y})| = r\frac{(\dot{\phi}_R + \dot{\phi}_L)}{2}. \tag{6}$$

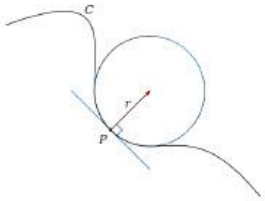

Figure 3: Curvature of a circle is defined to be the reciprocal of the radius. Image source: Wikipedia.

Question 2 *Given the curvature κ and the maximum wheel speed $\dot{\phi}_{max}$, what is the maximum* `robotVel` *that you can drive your robot at?*

7 Task 5: The effect of slippage

Does the robot accurately follow the desired trajectory? Can you think of any effects that influence tracking?

Lab 9

2.12: Introduction to Robotics
Lab 9: Visual Navigation with April Tags using ROS *

Spring 2022

Instructions:

1. **When your team is done with each task, call a TA to do your check-off.**
2. **After the lab, zip your files and make a backup using online storage/flash drive.**
3. **No need to turn in your code or answers to the questions in the handout.**

1 Introduction

An AprilTag is a 2D fiducial marker designed for easy estimation of the 6 DOF pose of the tag with only the use of a monocular camera in real time [1]. Every tag has an ID which can be identified. AprilTags are wildly used in robotic experiments to simplify and bypass object detection, identification, and pose estimation problems since these problems are still challenging. In this lab we will use AprilTags to estimate robot pose in the world frame. See Figure 1 for an example.

Figure 1: An AprilTag detection example. The detection program overlay highlighted tag image on the whole image after the tag is detected.

*

1. Version 1 - 2016: Peter Yu, Ryan Fish and Kamal Youcef-Toumi
2. Version 2 - 2017: Yingnan Cui, Luke Roberto and Abbas Shikari
3. Version 3 - 2019: Jerry Ng
4. Version 4 - 2022: Emily Kamienski

2 Setting up the code

Open a terminal (Ctrl+Alt+T), and enter the following commands without the leading $.

```
$ cd ~                              # make sure we are at home folder
$ git clone https://github.com/mit212/lab9_2022.git
$ source /opt/ros/kinetic/setup.bash
$ cd lab9_2022/catkin_ws
$ catkin_make                       # compile our packages
$ source devel/setup.sh            #setup the environment
```

You may need to make `me212bot_node.py` and `apriltag_navi.py` executable by navigating to the folder these files are located in and running `chmod +x filename.py`.

2.1 Folders and files

The `lab9_2022` folder contains all the files required for this lab. The overall structure follows the ROS catkin build system and is shown below. Unimportant files and folders are ignored for brevity.

- `catkin_ws` : ROS catkin Workspace
  - `src` : source space, where ROS packages are located
    - `me212bot` : a folder containing a ROS package for the 2.12 moving platform.
    - `apriltag` : a folder containing a ROS package for detecting AprilTags [2].

The folder hierarchy helps organize a large project that contains several *packages*. Package is a term for units of files related to one idea. The files can be code for libraries and programs, as well as config files. Now let's focus on the content inside package `me212bot`, which is for our robot platform. We shorten the name from `me212_robot` from Lab 2 for brevity. Important files in the package are listed below.

- `src/controller` :
  - `controller.ino` : a wheel velocity controller in Arduino. (Your old friend.)
  - `helper.h`, `helper.cpp` : helper classes for the controller.
- `scripts/me212bot_node.py` : a ROS node for the moving platform: read odometry from and send velocity commands to Arduino. (Your new friend.)
- `scripts/apriltag_navi.py` : a ROS node for implementing navigation policy using AprilTag. (Your new friend.)
- `scripts/helper.py`, `tf_helper_example.py` : helper functions for doing transform and their example usage.
- `launch/frames.launch` : a launch file for publishing two static coordinate transforms.

In this lab, `me212bot_node.py` and `apriltag_navi.py` have been modified for you. **Note about indents:** We recommend you use the *Geany* text editor because Python uses indents to indicate blocks. It is important to be consistent of the indent characters. Using a mix of tabs and spaces will cause tremendous issues when attempting to debug. The convention in this lab is 4 spaces for one level. To help you be consistent, Geany is configured to show spaces and tabs explicitly.

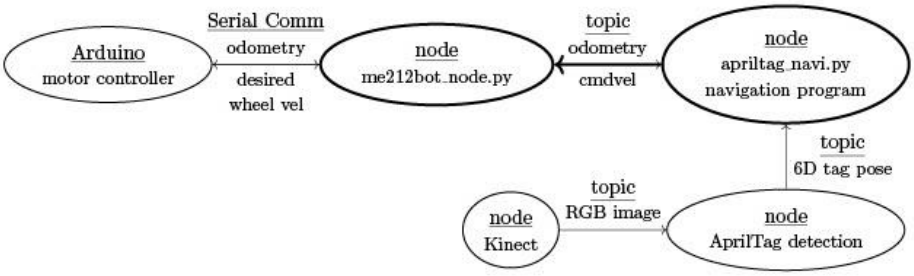

Figure 2: System architecture. The /cmdvel topic, shown with a thick arrow, is the one that we will walk through. This involves subscribing in me212bot_node and publishing in apriltag_navi node.

3 Publish/subscribe cmdvel through ROS topics

In this section, we use a ROS node for our moving platform. Here we name the package as `me212bot`. The main idea here is to use the Arduino microcontroller as a single node. See Figure 2 for the system architecture. Conventionally, Arduinos should focus on low level computation, like motor velocity control, which won't be changed frequently. Anything higher than that should be moved to a high-level program on the PC side with ROS interface. We remove the `PathPlanner` class from Arduino because this is a high level component and likely to be changed more frequently. The ROS node `me212bot_node` will act as a translator between ROS topic and serial communication.

The 4 major steps:

Step 1 Upload the provided `controller.ino` to Arduino. Remember to change the motor type in the code according to your robot. The type is written on the robot: either 26 or 53.

Steps 2-4 have been completed for you

Step 2 We have defined the message type for `cmdvel`.

Step 3 A subscriber in `me212bot_node.py` subscribes to the `cmdvel` and sends the commands to the Arduino.

Step 4 A publisher in `apriltag_navi.py` publishes wheel velocities.

After uploading the program, you will not see the wheels running because it waits for commands in serial port. Look at the `apriltag_callback` function in `apriltag_navi.py` and notice where the April tag subscriber is defined in `main()`.

3.1 Coordinate Transformations

To estimate robot pose in world frame, we assume we know the rigid transforms 1) between Kinect camera frame and robot frame; and 2) between AprilTag frame and world frame. And the AprilTag detection package will estimate the transform 3) between the AprilTag and the camera. By chaining these 3 transforms *correctly*, we can estimate the robot pose. See the appendix for more details.

3.2 Modifying April Tag ID and Size

Look at the ID of the April tag you were given. The April tag ID can be modified in `apriltag_navi.py` in the `apriltag_callback` function.

Additionally for the term project you may need to modify the April tag size, which can be done in the `~/catkin_ws/src/apriltags/launch/apriltags.launch` file. For example, the side length of the black square portion of the April tags used in this lab measures 0.083 meters.

4 Testing April Tag Detection (don't type the 6 commands, read next code block below)

We have several nodes and launch files to run in order to test the AprilTag localization. They are as follows:

```
$ roscore
$ roslaunch me212bot frames.launch    # launch static transform publishers
$ roslaunch me212bot viz.launch             # launch rviz
$ roslaunch freenect_launch freenect.launch # launch Kinect node
$ rosrun me212bot me212bot_node.py          # run the robot node
$ rosrun me212bot apriltag_navi.py          # run the navigation node
```

Here we put the static transform publisher of `apriltag` w.r.t. `map` and `camera` w.r.t. `robot_base` into a launch file `frames.launch`. The basic function of a launch file is to run several nodes in one command. You can open `frames.launch` to understand how to do that. It is just reformatting `rosrun` commands into XML [3].

To run the 6 commands above we need 6 terminals, which would be chaotic. Also, remembering and typing all the commands are error-prone. The tool we want to introduce to help us is `pman` (see the Appendix for more details). Instead of typing all the commands separately in different terminals, just go to the folder `~/lab9_2022/` in the terminal and type:

```
$ source software/config/environment.sh     #sets environment for the lab
$ pman
```

Then you will see a GUI with the 6 commands already inside (Figure 3). You can right click on each one to start. Note that you should start `roscore` before others. Or you can go to the menu bar on the top of the screen, and click on `Scripts/run`. That is a script written by past TAs to run the first 5 commands, and you need to start the `apriltag_navi.py` manually by right clicking because that program can be seen as a *main* program in this test and others are auxiliary.

If the code is run correctly, you should see the robot localization result in `rviz` when you hold the AprilTag in front of the Kinect. The result may be noisy depending on the distance to the tag, so the robot may *fly around*. However, the controller we will implement can deal with that. As long most of the estimation is correct, and the robot is gradually approaching the target, the navigation should still work.

Question 1 *Why do you think this noise happens? How could you deal with this noise?*

5 Visual Navigation using AprilTag

Now turn on the motors on your robot. In this last section, we use AprilTag to estimate robot pose in world frame. Now we can run a navigation policy, which receives the robot pose, and outputs wheel

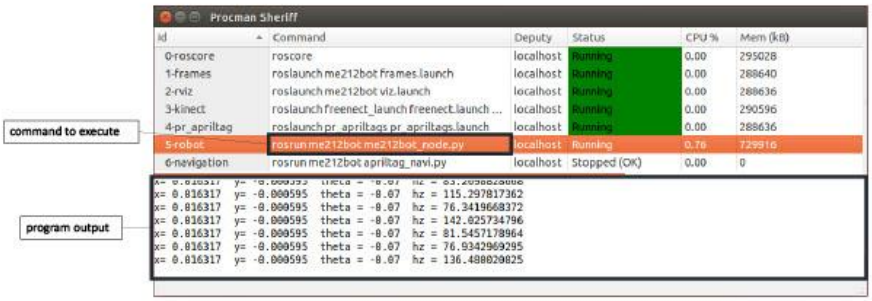

Figure 3: GUI of pman.

velocities for each iteration of the controller loop. The target pose is set to $(x, y, \theta) = (0.25, 0, 3.14)$ in the world frame. Here since the robot is constrained to have planar motion on the ground, so we use planar representation just like in Lab 7. We can project the 6D pose estimated to 3D. In the `navi_loop` function in `apriltag_navi.py`, a simple example navigation policy can be found:

- `if` AprilTag is not in view:
  ⇒ stop moving
- `else if` robot position and orientation is in the target region:
  ⇒ stop moving, and keep stopped
- `else if` robot position is in the target region:
  ⇒ turn to the target orientation
- `else if` robot faces toward the target position:
  ⇒ go straight forward
- `else` :
  ⇒ turn to face the target position

Then, you put the robot on the ground, in a 2m×2m square in front of the AprilTag, and facing the tag. Use `pman` to start all the processes including `apriltag_navi.py`. You should see the robot navigate to the target pose.

Question 2 *Given the tag size, tag location, camera pose on the robot, camera field of view, farthest distance in which tag can be detected, etc., that you can measure easily, what is the region of starting pose (x,y,θ) that the robot can successfully navigate to the target? You can reason the (x,y) space given fixed θ values, e.g. 0 deg, 15 deg, and 30 deg. Verify it with experiments.*

Question 3 *This navigation policy uses a bunch of if/else statements. How can we improve this to make it general? Think about the past few labs and how controls works in a broad sense.*

6 Future directions

Here we list some directions that you can pursue to practice ROS further and to complete your final project:

1. Let the Arduino report wheel velocities, and let `me212bot_node` publish the velocities using ROS topic.

2. Here we assume the robot can always see the AprilTag. You can come up with some simple control policy that can turn the robot to find a tag if no tag is in view.

3. Apply PID to control robot pose in apriltag navigation instead of fixed sets of wheel velocities.

4. Build a localization component that fuse information from odometry and AprilTag. Odometry is local but less jumpy, and AprilTag is global but usually noisy.

5. Create a ROS service in `apriltag_navi.py` to change target poses on the fly.