EEL 5683 Project 1 Report

Terrance Williams

February 20, 2023

Contents

1 Introduction

The EEL 5683 course is a course on Autonomous Mobile Robotics. The primary objectives is to learn how robots proceed through the perception, localization, path-planning, and execution phases of autonomous operation. In other words, we study the "see, think, act" cycle. This course is project-based, allowing students to work with and gain physical intuition for concepts learned in class.

This report will serve as a platform report for the JetHexa ROS robot produced by Hiwonder, a robotics company. The JetHexa platform was chosen to provide an opportunity to learn ROS, work with a new robot configuration (hexapod), as well as work with advanced perception tools such as the 3D depth camera and Lidar. In keeping with those objectives, Project 1 served as a "grounding" project. Time was primarily spent working with the robot-specific tools, reading provided documentation to understand its movement and features, as well as learning the fundamentals of ROS, mainly its underlying architecture and basic navigation and operation commands.

Figure 1: JetHexa ROS Robot

2 Hiwonder JetHexa ROS Hexapod Platform

Hiwonder's JetHexa platform is a pre-assembled, open source ROS robot designed for advanced learning. The version used this semester is the Advanced Kit, which, in addition to its impressive kinematics algorithm, includes numerous perception-based features including color detection, facial recognition, 2D and 3D mapping, and navigation. The robot also has the ability to synchronize with other JetHexas and perform formations, but that is beyond the scope of this project.

2.1 Anatomy and Hardware

The JetHexa platform is an 18-DoF hexapod robot. Each of its six legs has three degrees of freedom via HX-35H servo joints which allows for a wide range of complex and simple motions through serial bus.

The platform is controlled by an Nvidia Jetson Nano Developer Kit. A powerful piece of hardware, the Jetson Nano has a quad-core CPU processor and 4GB memory, leading to possibilities such as multi-threading, computer vision, and even artificial intelligence projects[1]. Additionally, the computer sports a gigabit Ethernet port, ports for display and HDMI output, and multiple USB ports for additional peripherals and data transfer. The robot has two major peripherals: its 3D Depth

(a) JetHexa Crouching

(b) JetHexa Twisting

Camera and its Lidar, both of which aid in more advanced autonomous robotics projects that include path planning and localization. Detailed information regarding the specifications of these components may be viewed in the Appendix.

2.2 Movement

In terms of movement, the JetHexa platform operates using a Cartesian coordinate system whose origin rests at the center of the platform's front-face. This means that from the frame of the robot, forward is the +x direction, left is the +y direction, and upward is the +z direction. The robot is able to translate across an entire defined plane (x-y axis) without changing orientation and is also able to move in the z direction by modifying both its body height and step height. This allows the robot to climb steps and descend steps and inclines.

In addition to translational movement, JetHexa is also able to rotate, reorienting its body about the z-axis with a 360° range. It's also able to twist about this

[1] https://developer.nvidia.com/embedded/learn/get-started-jetson-nano-devkit

[2] Image Source: JetHexa Tutorials 7.3.1 - Build Coordinate System.

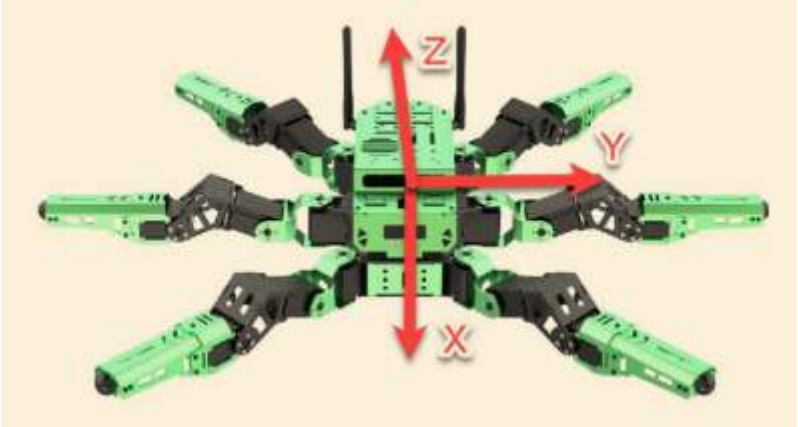


Figure 3: JetHexa's coordinate axis [2]

axis. While more limited, the robot can also rotate about the x- and y-axes to provided forward, backward, left, and/or right tilts. This ability, utilized effectively, allows the robot to maintain balance about its center of gravity for especially inclined environments. While not tested in this project, the robot has been shown to dynamically maintain balance [3].

As stated, the robot can both translate and reorient about its coordinate axes. The method in which this is done also varies. JetHexa has movement parameters which allow the operator to customize the bot for desired behavior. The first of these parameters, stride length, determines how far the robot moves in one movement cycle (all legs have moved). The stride length ranges from 0 to 65 mm, allowing a large range of behavior. Step height, how high JetHexa raises a given foot, ranges from $0:50$mm. Finally, the robot's movement speed is adjustable, however, the concrete speed range is not known by the author at this time as the speed of the robot depends on the stride length and the time it takes to complete a cycle, both of which are controllable. In fact, the Hiwonder-provided control application only lists speed in terms of percentage since stride length is also variable.

Other parameters such as rotation angle, total number of steps to move, movement interruption, etc. are also able to be modified, giving the user even finer control over the robot's behavior. One important parameter, gait, determines *how* JetHexa completes a movement cycle. Being a six-legged platform, it can represent $N = (2k-1)! = 11! = 39.9\times10^6$ distinct events and therefore even more gaits which can be daunting. However, Hiwonder provides the user with two gaits, tripod and ripple, that determines both the translational and z-axis-orienting motion.

The tripod gait consists of alternate movement of JetHexa's legs in sets of three. In the first half-cycle, the front-right, middle-left, and back-right legs move. In the

[3]JetHexa Self-balancing: https://www.youtube.com/watch?v=SGzgRffqfDQ

latter half-cycle, the front-left, middle-right, and back-left legs move. Movement in this way allows the robot to remain dynamically stable because its center-of-gravity remains within the triangle defined by the supporting legs.

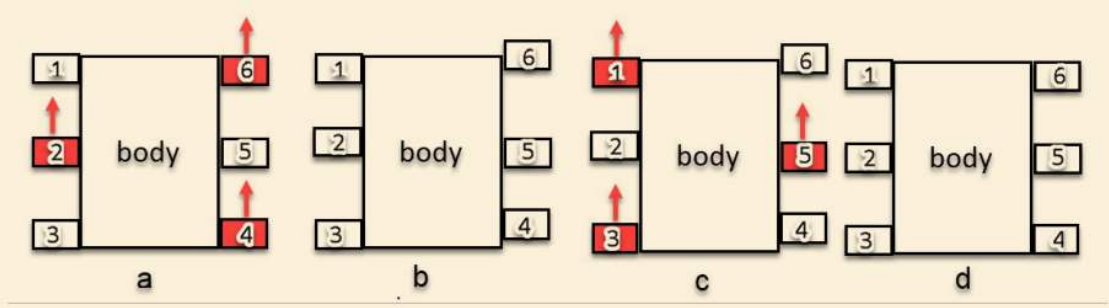


Figure 4: Tripod Gait Diagram [4]

Ripple gait is little more complex to grasp intuitively. Essentially, in this gait, two legs are always in stride while the remaining four are in support. The lifted legs alternate based on diagonals where if, for example, the middle left leg and the back-right leg were in motion, the back-right leg would touch the ground, and the front-right leg would then lift. The middle-leg would then touch the ground, and the back-left leg would raise. This pattern repeats until all legs have been lifted and swung (Ex. 3–> 5–> 4 –> 2 –> 6 –> 1 –> 5; see Figure 5). Qualitatively, this gait resembles a spider crawling (though spiders have eight legs) and appears to be a more fluid motion than tripod gait.

[4]Image Source: JetHexa Tutorials 7.1.2 - Tripod Gait Analysis.
[5]Image Source: JetHexa Tutorials 7.1.3 - Ripple Gait Analysis.

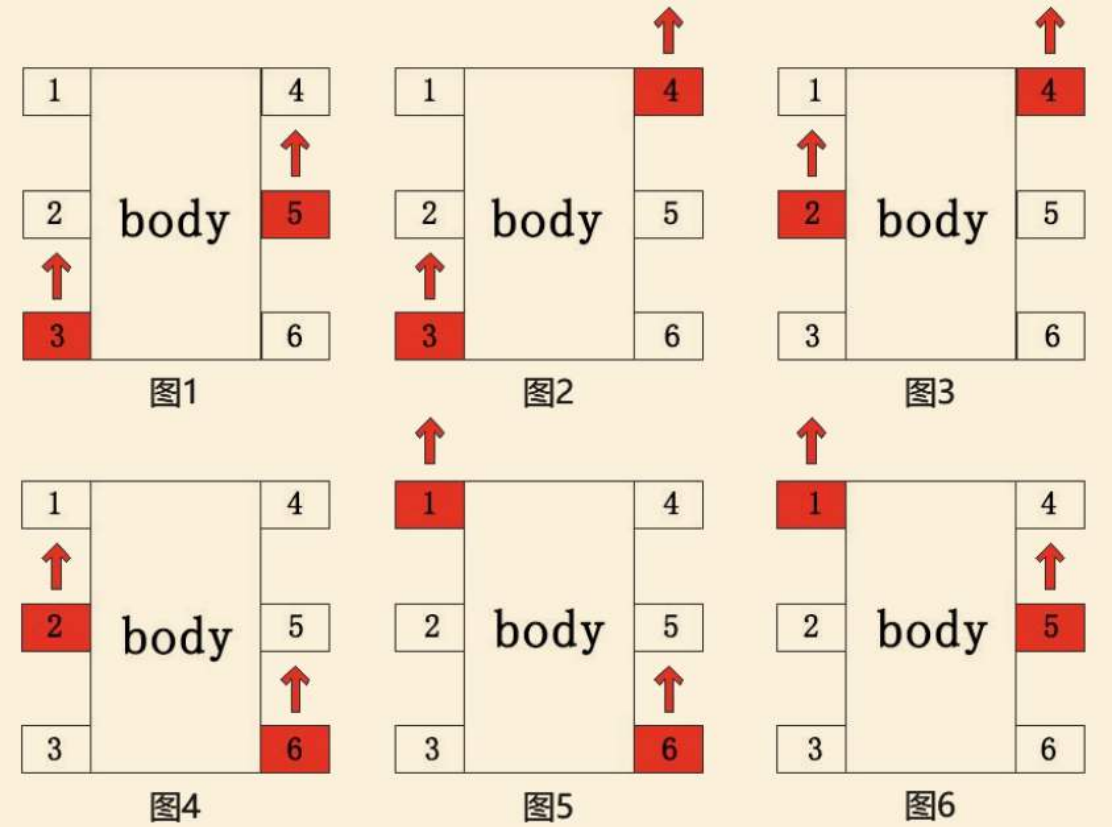


Figure 5: Ripple Gait Diagram [5]

2.3 Software Control

In terms of controlling the robot platform, Hiwonder provides three options. The first option is a mobile app (WonderAi) that allows the user to select four modes to gain familiarity with the robot's features. The other options take the form of wireless control via a provided PS2-styled controller and a more "traditional" ROS control by remote desktop commands in NoMachine.

To use the WonderAi app, the user connects to the JetHexa-generated local Wi-Fi network which allows the app to detect the platform. It has four modes of control ("games") for the JetHexa: *Robot Control*, *Robot Performance*, *Lidar*, and *Gesture Control*. Each mode provides an easily-accessible way to use the robot and become familiar with its abilities.

2.3.1 Robot Control

Figure 6: WonderAi *Robot Control* Mode

The first WonderAi game is *Robot Control* [6]. This mode provides a general piloting method for the platform, granting the user the ability to translate and reorient at multiple speeds, modify step height and body height, as well alternate between the two defined gaits. Additionally, the game includes a window that displays live images from the JetHexa's on-board camera, allowing the user to continue to pilot the robot when it is out of the line-of-sight and within the connection range. The

[6] A Hiwonder-provided video of this app in action may be seen at https://www.youtube.com/watch?v=aaXqHSAWNvE

Robot Control mode was instrumental in gaining preliminary understanding of the robot's performance. It will be noted, however, that due to the connection latency as a result of the relatively weak Wi-Fi connection, this mode is likely not a viable option for piloting a full-project.

2.3.2 Robot Performance

The second game mode is *Robot Performance.* If the *Robot Control* mode is helpful for understanding the JetHexa's general movement, *Robot Performance* provides insight into complex movements and sequences the robot can perform. This mode consists of nine buttons and one toggle slider. The nine buttons are for 'Actions', pre-determined movement sequences. Eight of the nine buttons are Hiwonder-provided actions, and the ninth button is for user-defined custom actions. The pre-defined actions are[7]

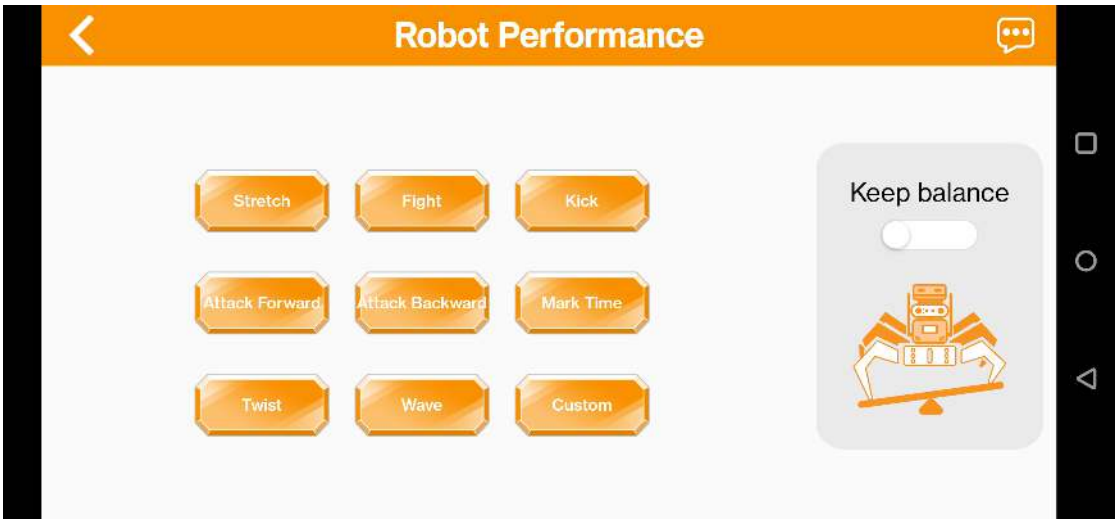


Figure 7: WonderAi *Robot Performance* Mode

- Stretch - the robot spreads out, lying down flat on its current surface (typically the ground). It should be noted that coming out of this position is easier on surfaces such as rugs than hard-wood.
- Fight- The robot stands on its back four supporting legs and uses its front two legs to synchronously wave as if posturing for a fight.

[7] A few actions and the self-balancing mode may be seen via the following link: https://www.youtube.com/watch?v=UDpsKOi9D98

- Kick - The robot uses its front left leg (from the perspective of a viewer directly facing the robot) to swiftly swipe left to right (like kicking a soccer ball)
- Attack Forward - The robot sequentially swings leg pairs forward beginning from the hind legs, resulting in a lunging motion.
- Backward Forward - The reverse direction of the "Attack Forward" action.
- Mark Time - The robot marches in place for a brief moment.
- Twist - JetHexa runs through a sequence of body tilts to move its body in a circular fashion (picture a hula-hoop or a precessing top)
- Wave - JetHexa uses its front leg to simulate a friendly wave.

The JetHexa tutorial series comes with instructions for creating, exporting, importing, and editing actions. This ability may prove incredibly valuable for any user who wants to create complex motions such as dances or robot aerobics.

(a) Top View (b) Isometric View

Figure 8: JetHexa Stretch Action

Finally, the mode includes a self-balancing toggle. When toggled on, the JetHexa will be able to adjust its body posture (thereby its center of gravity) in order to maintain its balance on surfaces of changing incline.

2.3.3 Lidar

The *Lidar* mode allows the user to play with pre-coded Lidar functions. It has three Lidar modes: "Avoid Obstacle", "Lidar Following", and "Lidar Guarding". Each mode uses Lidar to accomplish a different task. In Avoid Obstacle mode,

the JetHexa platform continuously moves forward. If the Lidar detects an object within a given distance (which we can set from 0.5m to 1.5m), it will move around the object or reverse course.
In the Lidar Following mode, the robot will follow an object at a pre-determined distance. If the object is too close, JetHexa will move away from it, whereas, it will move closer if the object is too far away.

Finally, in Lidar Guard mode, the JetHexa will continuously re-orient itself be front-facing toward an object in its detection range[8].

2.3.4 Gesture Control

In this mode, the user can use finger gesture to control the JetHexa's movement. The JetHexa uses its camera and runs computer vision to detect a hand structure. It is able to identify the number of fingers displayed as well as certain hand shapes. One can send translational movement commands to the robot by doing the following: Show the JetHexa an open palm. It will identify five fingers. Next, display only an index finger; this lets the robot know a command is about to be drawn. Upon hearing the beep that alerts the user to begin drawing, draw a straight line in any cardinal direction (left = move left, right = move right, up = move forward, down = move backward). To signal the end of a command, an open palm is again displayed. The robot will then execute the received command.

It was observed when testing this that gesture commands are misinterpreted rather easily. This could be a user-operator error, but if this feature were to be used in a programmatic fashion, it may be beneficial to include a form of user confirmation for the command JetHexa interprets. Currently, the robot will execute the command it believes is sent without confirmation.

2.4 Wireless Control

The second method of JetHexa control consists of using a wireless controller to communicate with the provided receiver (connected to one of JetHexa's USB ports). The user is able to pilot the robot in real-time, with minimal latency compared to the *Robot Control* WonderAi method.

The controller itself is modeled after the Sony DualShock 2. It features analog sticks as well as a direction pad (D-pad) for movement control, polygonal buttons for rotation and tilting, bumper buttons for speed and stride length adjustment, and finally, START and SELECT buttons for pose and settings resets. The complete layout is shown in Figure 9.

[8]Each mode is displayed in the following video:
https://www.youtube.com/watch?v=-lRkl6Zw-zM

Figure 9: JetHexa Wireless Controller Layout

2.5 Programmatic Control

Perhaps the most versatile of the control methods is the ability to control the JetHexa platform with a programming language such as Python using ROS packages. Hiwonder allows the user to access the JetHexa files via "remoting" into the JetHexa's Jetson Nano controller. This is done by connecting to the Wi-Fi network generated by the Jetson Nano and connecting to the remote server IP through the remote desktop software NoMachine. Since the JetHexa platform is ROS compatible, it runs a Linux distribution, allowing remote users to use the Linux shell to interact with the robot's system files.

Since the robot is advertised as open-source, JetHexa has made the entire source code available for users to study. Theoretically, this means the tasks and features the user can implement is only limited by the hardware capabilities of the JetHexa platform, the readability of the source code, and the programming ability of the user(s).

The JetHexa tutorials come with information on basic movements and rotations, advanced speed adjustments and polygonal line movement, inverse kinematics, Lidar, computer vision, and more, giving the user a wealth of valuable information for adapting these features for his or her own ends. It is important to note that initially, when attempting to control the robot's translational movement, the ROS master would raise an error due to a duplicate node name. After contacting Hiwonder's support department, the following fix was provided:

sudo systemctl stop jethexa_bringup.service

3 Flow

As stated previously, the JetHexa platform is powered and controlled by a Jetson Nano computer. The computer runs Ubuntu 18.04 LTS Linux distribution and within that distribution, it runs ROS Melodic, an operating system for robot infrastructure. It is at this level that the JetHexa's subsystems communicate in order to produce an array of behaviors. For this project's purposes, three subsystems are identified: computer vision, kinematics control (motion), and Lidar localization. The coordination between the JetHexa's components is shown in Figure 10. The ROS subsystems are connected with dotted lines to show that their communication is facilitated by the ROS architecture.

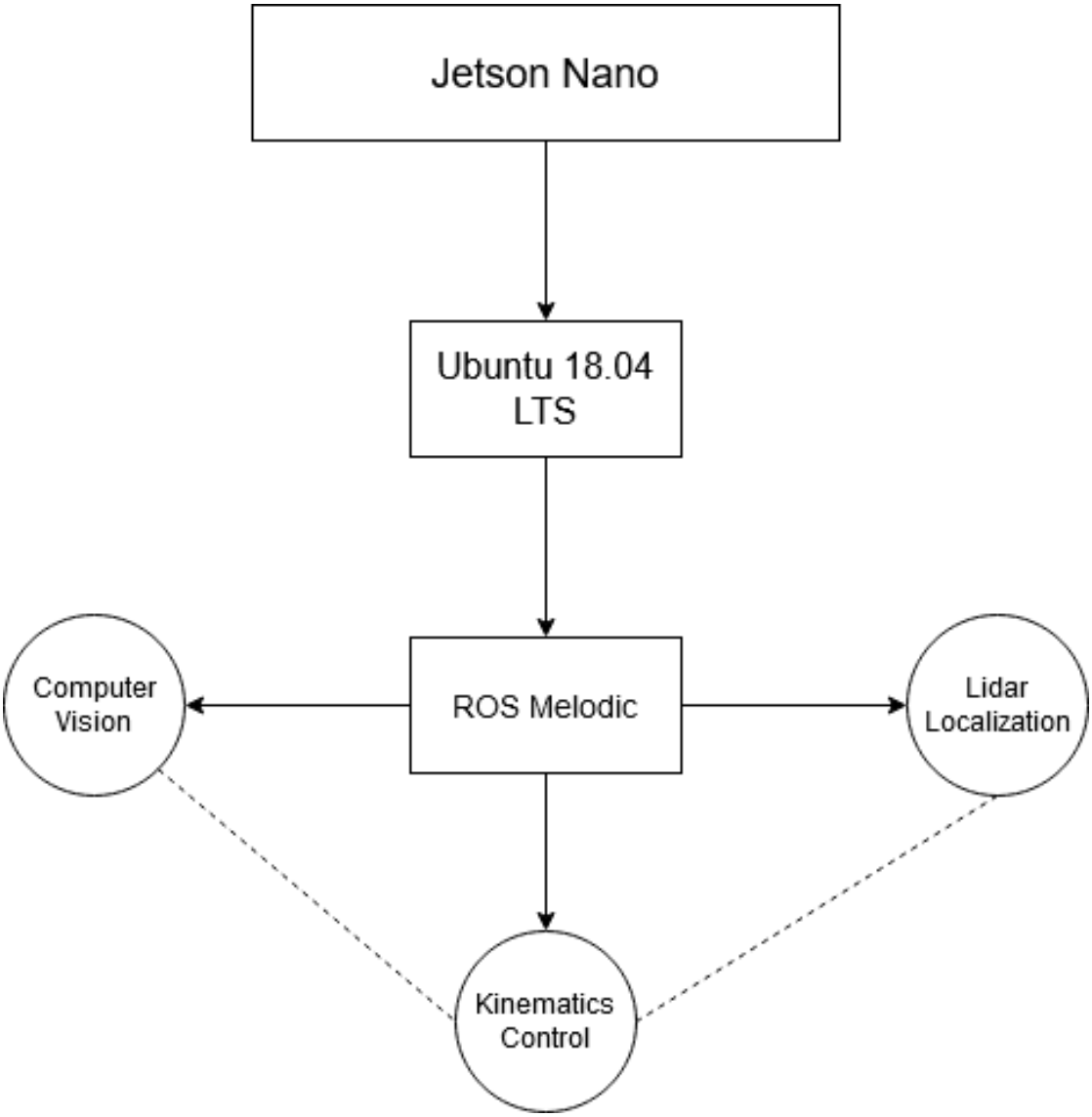


Figure 10: General JetHexa System Flowchart

For a specific example of subsystem interplay, consider a JetHexa motion system that uses gestures to control the robot while expecting it to navigate around obstacles. To do this, the robot must take images from its 3D camera and run a gesture detection algorithm to identify if a command is being sent and what the specific command is. This is done by the computer vision subsystem. From here, the JetHexa knows it needs to move in some specified way. The movement instructions are then passed to the kinematics control subsystem, causing the robot to move as specified. As the robot moves, its Lidar system scans the surrounding environment, identifying potential collision points within its detection range. If

an object is detected, it sends an alert which is passed to the kinematic control to induce a corrective motion, preventing collision. A visual depiction of this process is shown below.

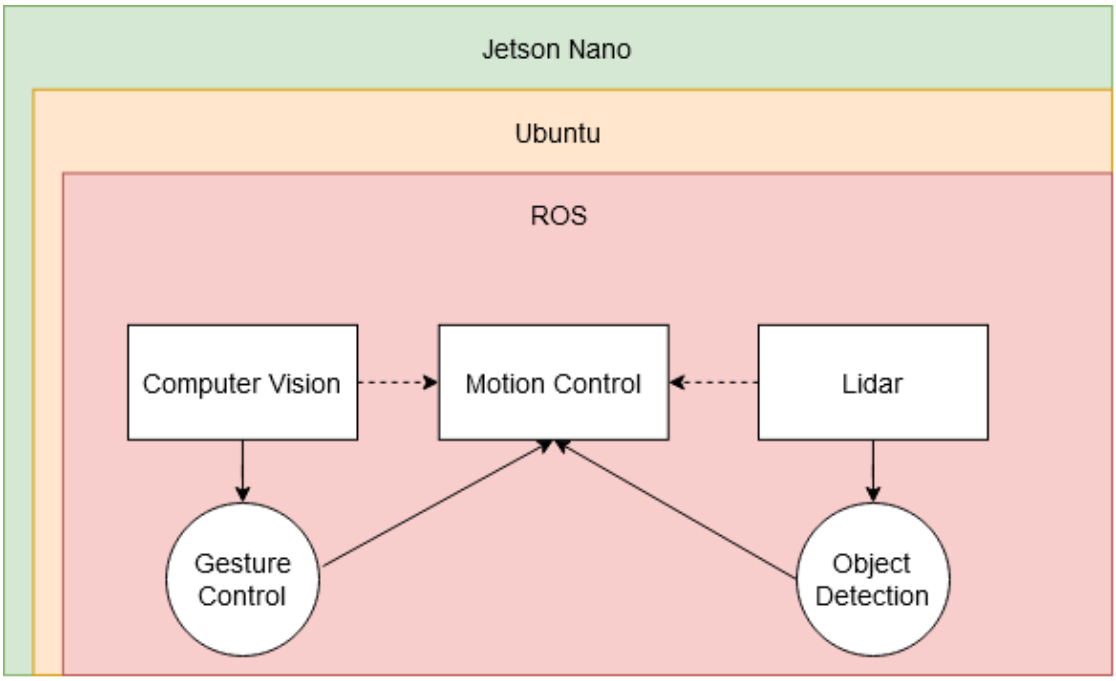


Figure 11: Example of Subsystem Interplay

4 Summary

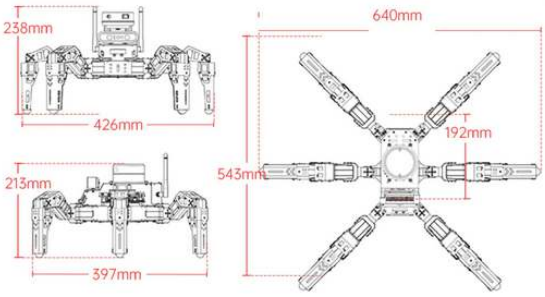
In this project, the JetHexa ROS Robot's features and behaviors were explored. The goal in doing so was to foster a sense of familiarity with the robot's operation as well as understand how to interact with the robot on multiple levels. This process included piloting the robot with the Hiwonder-provided Android app, the wireless controller, and programmatically via remote desktop.

Project 2 will continue this learning, moving into learning how the Lidar and 3D camera systems are controlled by the JetHexa platform. With this and continued study of ROS, it is the goal of the author to develop a JetHexa ROS package that can ultimately be used for obstacle navigation and other showcasings of the robot's capability.

A Specifications

The following images highlight the specifications for the various components of the JetHexa ROS robot[9].

A.1 JetHexa ROS Robot



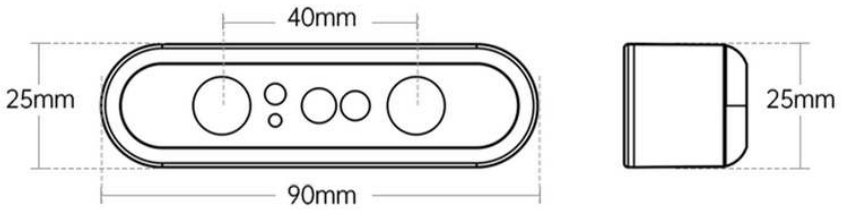
JetHexa Parameter

Product weight	2.5 kg
Material	Full-metal hard aluminum alloy bracket (anodized)
Monocular camera pan-tilt	2 DOF
Battery	11.1V 3500mAh 5C Lipo battery
Battery life	60min
Robot DOF	18DOF
Hardware	ROS controller and ROS expansion board
Operating system	Ubuntu 18.04 LTS + ROS Melodic
Software	PC software + iOS/ Android APP
Communication method	USB/ Wi-Fi/ Ethernet
Programming language	Python/ C/ C++/ JavaScript
Storage	32GB TF card
Servo	HX-35H intelligent serial bus servo
Control method	Computer/ phone/ handle control
Package size	387 * 356 * 210mm (length*width*height)
Weight (with package)	3.6 kg

[9]Images Sourced from the Hiwonder website: https://hiwonder.hk/products/hiwonder-jethexa-ros-hexapod-robot-kit-powered-by-jetson-nano-with-lidar-depth-camera-support-s variant=39876752670807

A.2 3D Depth Camera

3D Depth Camera Parameter

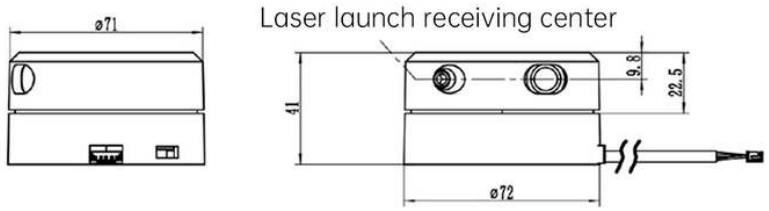


3D technology	ORBBEC binocular structured light
Accuracy	1m: ±6mm
Field of View	H 71° V 43.7°
UVC	support
Resolution @ frame rate (depth mode)	640x480 @30fps/ 320x240@30fps
Support operating system	Android/ Linux/ Windows
Data interface	USB2.0 Micro USB
Power consumption	<2W
Safety	Class1 laser

Working range	0.3 - 3m
Field of View (FOV)	H 67.9° x V 45.3°
Depth processing chip	MX6000
Close protection	support
Resolution @ frame rate (RGB mode)	1920x1080@30fps/ 1280x720@7fps/ 640x480@30fps
Working scene	indoor and outdoor
Size	90*25*25 mm
Operating temperature	10 - 40℃

A.3 EAI G4 Lidar

EAI G4 Lidar Parameter



Model	EAI G4 Lidar
Recommended scene	indoor scanning and positioning
Supply voltage	5V
Scanning range	360°
Measuring radius	0.12 - 16m
Communication rate	230400bps
Sampling rate	9k
Scanning frequency	5Hz - 12Hz

Angular resolution	0.28@7Hz
Supply current	1000mA
Pitch angle	0.25 - 1.75Deg
Output interface	UART serial port
Operating temperature	0 - 50℃
Ranging accuracy	2.0% (1m < distance ⩽ 8m) 2cm (distance ⩽ 1m)
Size	72 * 71 * 41mm

EEL 5683 Project 2 Report

Terrance Williams

March 31, 2023

Contents

1 Introduction

In Project 1, the JetHexa was tested for movement and basic peripheral functionality. A general sense of the platform's performance was ascertained and the fundamentals of ROS were learned. Project 2 extended the work done in Project 1, diving further into JetHexa's numerous features by studying its navigation module in aims of autonomous movement. Ultimately, the goal of Project 2 was to be able to use the JetHexa's navigation module to perform Simultaneous Localization and Mapping (SLAM), model/display the map using software for the user's viewing, and move the robot to a user-specified point.

Figure 1: The JetHexa Robot

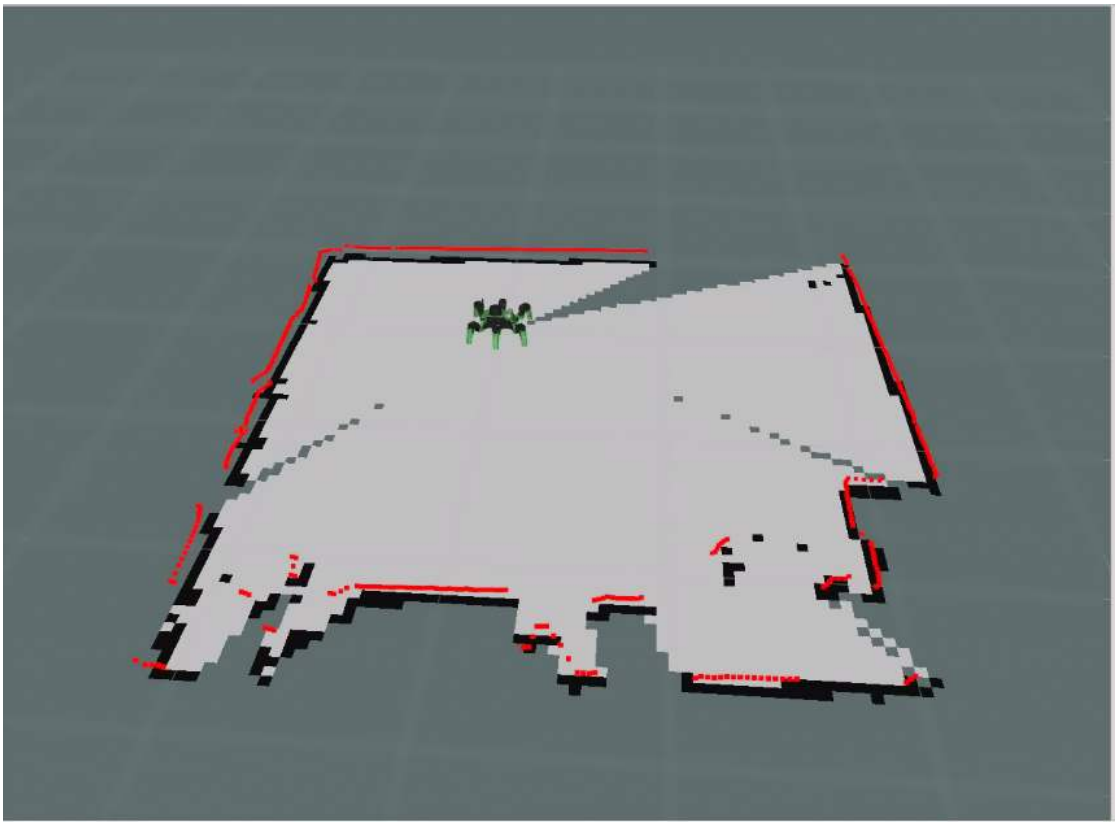
2 JetHexa Navigation

Hiwonder follows the *See, Think, Act* philosophy of autonomous motion. JetHexa uses laser-based perception, wielding its EAI G4 Lidar to gather data on its environment. Specifically, the robot uses triangulation lidar [1], with a ranging frequency of 9000Hz, a typical scanning frequency of 7Hz, and can scan in 360° [2]. Distance-wise, the lidar can scan objects from 0.12m to 16m with a resolution of 0.28 at the specified scanning frequency. Finally, in terms of error, the lidar has an absolute error of 2cm when the ranging distance is less than one meter and a relative error of 2.0% when the ranging distance is between one meter and eight meters. The provided documentation does not specify the error for distances greater than eight meters. In addition to these parameters, the documentation also provides information such as wiring as well as the device's method of communication, including serial information such as baud rate and HIGH/LOW voltage ranges. After the perception method is implemented, JetHexa performs localization and path-planning during the *Think* portion of the cycle.

2.1 Localization

In this stage, the JetHexa localizes and produces a map using a user-specified SLAM method. There are four map-making algorithms the user can use: Cartographer Mapping, Karto Mapping. Hector Mapping, and Gmapping. Each mapping algorithm has its benefits and drawbacks based on use of odometry and laser scanning, error mitigation, resource costs, etc. The Hiwonder tutorials utilize Gmapping, so that algorithm is used in this project. However, testing the efficacy of JetHexa's navigation using the other three methods could be a worthwhile pursuit for future work.

Gmapping is an algorithm developed by Grisetti, Stachniss, and Burgard that is a modified Roa-Blackwellized particle filter (RBPF) [3]. Essentially, it is an RBPF that also utilizes odometry and laser scanning to reduce the number of particles needed to perform the operation, thereby saving a considerable amount of memory and computation time [JetHexa 12.6]. This algorithm was used to have the JetHexa map an office room (see Figure 2). Once a map is generated, it can be viewed by the user using the program Rviz, a ROS package. Not many features of the room may be seen in the map, but the robot was able to outline out the borders of the room rather well. JetHexa also has the ability to save maps to its memory for future use.



(a) Generated Map

(b) Actual environment

Figure 2: Rviz representation of the Gmapping of the office room JetHexa is stored in versus the room itself.

After generating a map, the JetHexa system can then perform its second localization stage. To do this, it uses the Adaptive Monte-Carlo Localization ROS package (*amcl*) which is a package that uses laser-based maps, scans, and transformations to represent pose information, giving a probabilistic representation fo the robot's position [4]. With this package, the robot is able to perform an informed guess to determine its location on the map. Once this localization is done, the path-planning stage begins.

2.2 Path-planning

The idea behind the path planning for this application is that a user can specify a desired point within the environmental map and have the robot move itself to that position, avoiding obstacles and boundary collisions in the process. Hiwonder provides the scripts needed to do this, so the user need only understand how to operate it via mouse-click. However, for custom behavior, a user-provided method of control could also be used. The path-planning itself is done through the ROS Navigation Stack, a metapackage of multiple important ROS packages. Specifically, the *move_base* package is utilized via global and local planners. Visually, the user can specify the desired goal points and view the path taken in Rviz.

3 Complete Navigation Process

The previous section served as a baseline to understand the goal of the JetHexa Navigation package. This section will detail the complete process of generating autonomous motion with the robot[1]

3.1 JetHexa SLAM: Map Genesis [5]

To begin, the user connects to the robot's WiFi network and remotes to the robot using NoMachine. The first thing to do when attempting to run JetHexa's navigation package is to stop the JetHexa's startup process. This is the process that runs when by default whenever the machine is powered on and can interfere with processes a user wants to run[2] Run the following command to stop the program:

sudo systemctl stop jethexa_bringup.service

Next, the map of the robot's working environment must be generated for future use. Place the robot at a convenient starting point and run the JetHexa SLAM package with Gmapping:

roslaunch jethexa_slam jethexa_slam.launch slam_methods:=gmapping

This starts the mapping process. Indicators that the program is working successfully is a single beep followed by the JetHexa moving to its Reset/Home pose. The Lidar will also begin to continuously rotate as it scans the room. To view the map, launch Rviz:

roslaunch jethexa_slam jethexa_slam_rviz.launch

This will start Rviz under a specific configuration defined by HiWonder. At this stage, the user can explore the robot-map accuracy by piloting the robot. Hiwonder provides a keyboard control program to do this
(roslaunch jethexa_slam jethexa_keyboard_control.launch), but the user also has the option to use the joystick controller. Controlling the robot manually at this stage is useful to gain insight to how the robot's motion affects its representation on the map in Rviz.

Saving the map is next. This is done via the *map_sever package*. First, navigate to the directory where JetHexa stores its maps:

roscd jethexa_slam/maps

Now run *map_server*'s "map_saver" node.

[1]The author assumes the reader is able to properly remote into the robot using NoMachine.
[2]This was the cause of the performance issue initially seen in Project 1.

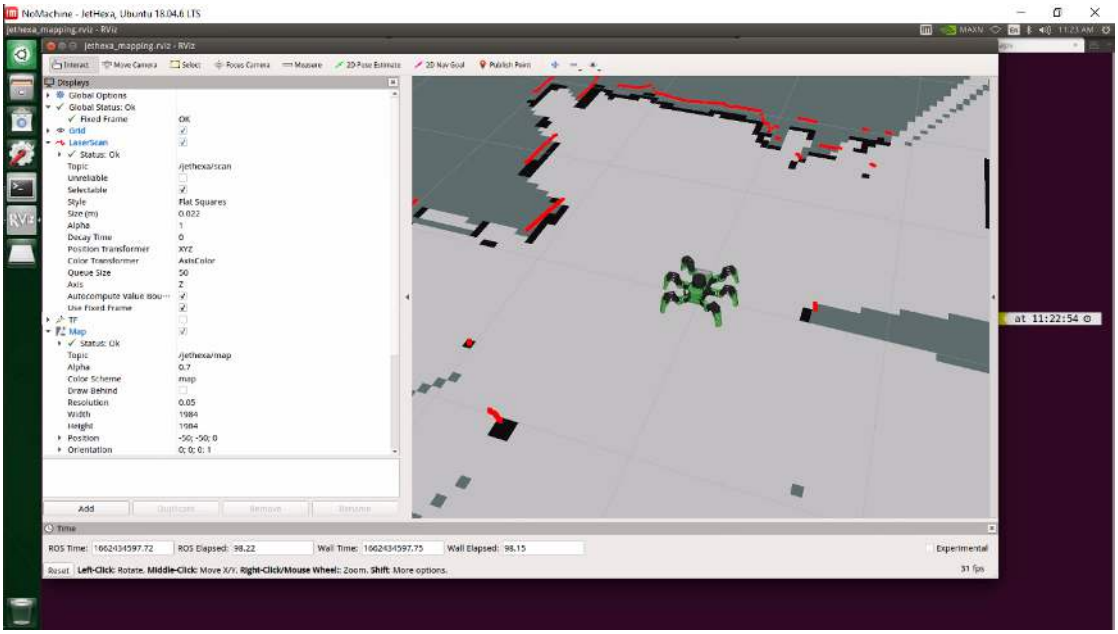


Figure 3: Rviz with options and parameters displayed.

rosrun map_server map_saver -f map_name map:=/jethexa/map

The map is now saved for future use. The SLAM process is complete and only needs to be completed if a new maps needs to be generated.

3.2 JetHexa Navigation [6]

Now for navigation. If it hasn't been done already, disable the startup program using the command in the previous subsection. This prevents interference and duplicate node name issues. To begin the navigation process, first load the map of the environment:

roslaunch jethexa_navigation jethexa_load_map.launch map:=map_name

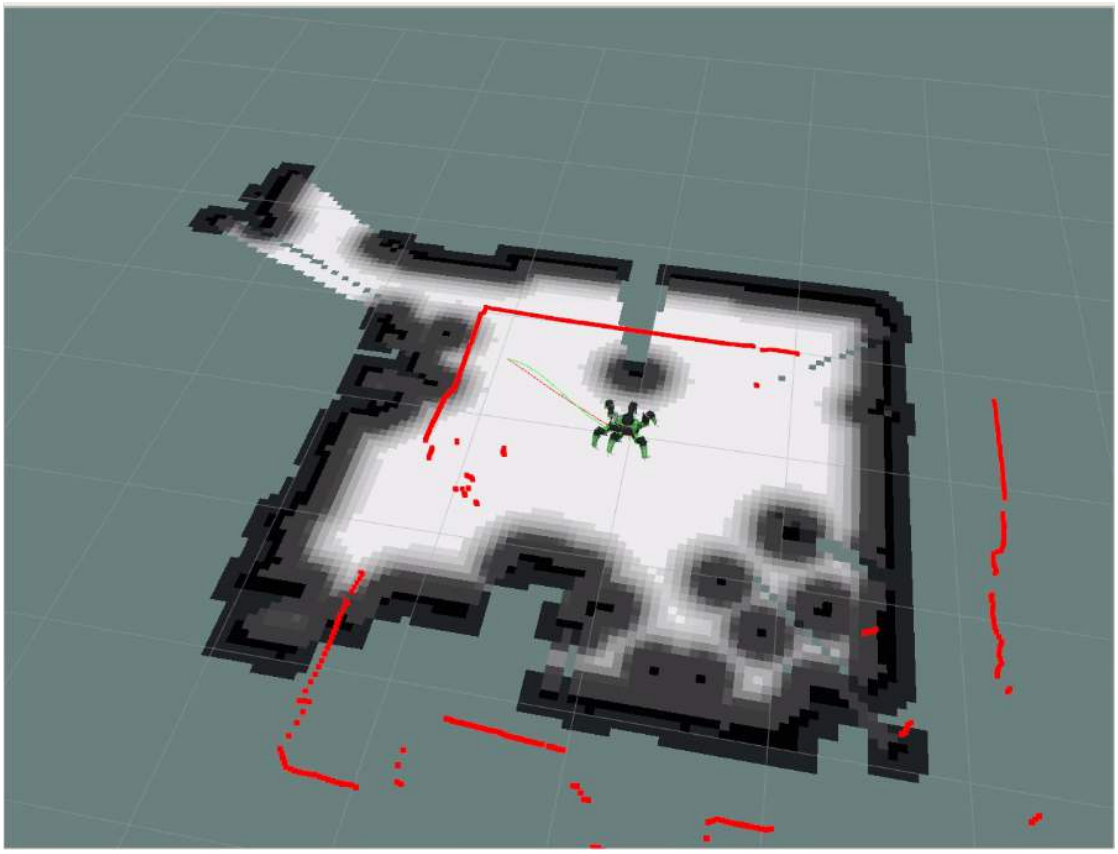
Next, launch the navigation program:

roslaunch jethexa_navigation jethexa_navigation.launch

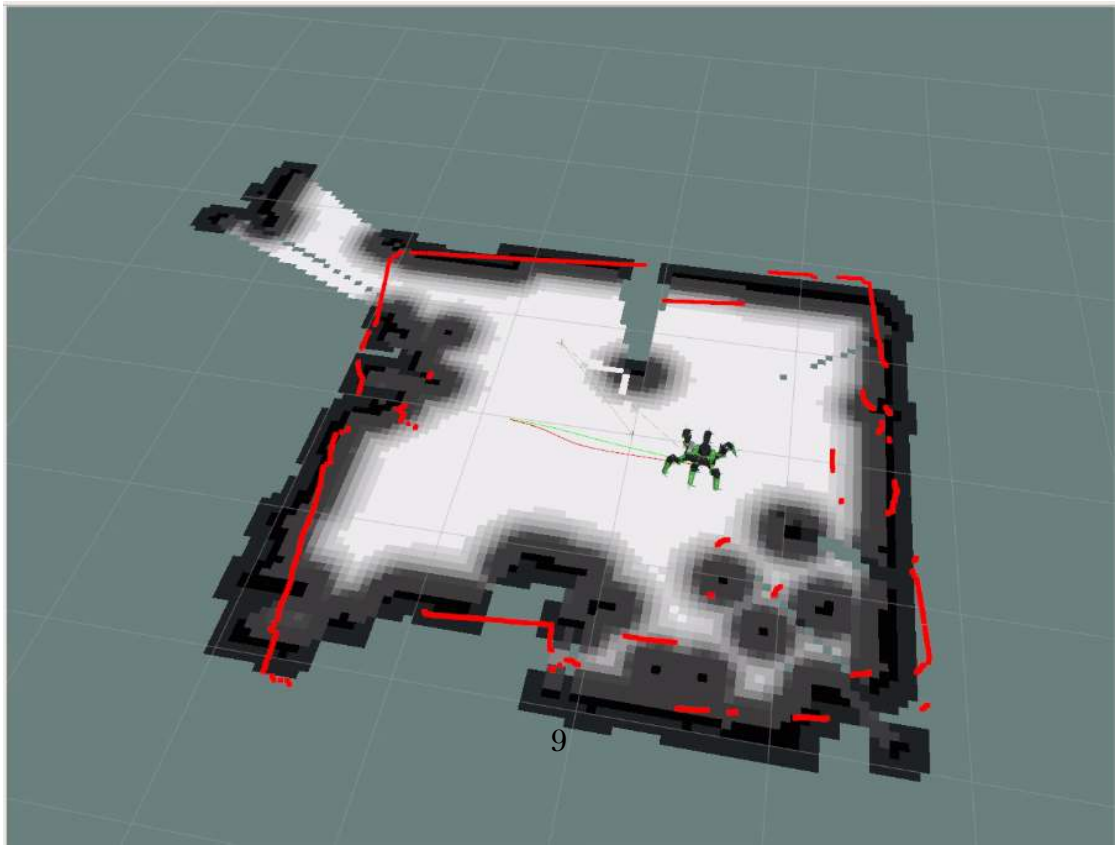
Finally, after waiting for the navigation program to finish initialization, launch Rviz under navigation configuration:

roslaunch jethexa_navigation jethexa_navigation_rviz.launch

With the completion of this step, the robot may now be given destination points. The user can set these points one at a time using *2D Nav Goal* or set multiple points at once using *Publish Point* to create a route [7]. JetHexa will calculate the best route to the goal point(s) and navigate there. In Figure 4, two examples of path planning are shown. The red trajectory is the straight-line path from the JetHexa to the goal point. The green trajectory is the planned path the robot calculates to reach this trajectory. The latter trajectory is malleable and does change as the robot executes its movement.



(a) View 1



(b) View 1

Figure 4: Views of JetHexa Navigation path planning.

3.3 Note on Performance

One thing to note regarding the autonomous navigation is that JetHexa currently collides with obstacles when moving to its goal position. Static, pre-determined points of collision are avoided, but when inserting an obstacle such as an empty box into the robot's movement path, it will collide with the object, pushing through it to continue to its goal. In other words, the robot at this time can move to its goal position, but may collide with undesired objects in the process. Improvements on this performance will be pursued for Project 3.

3.4 JetHexa Navigation Summary

JetHexa utilizes many packages of the ROS Navigation Stack to achieve autonomous navigation. In general, the process is to create and save a map of the environment using Gmapping SLAM, load the map via *map_server*, launch the navigation module, and control via Rviz inputs. A complete list of the commands are presented below for the reader's convenience:

Linux Commands

Before beginning either map generation or navigation, run the following:

sudo systemctl stop jethexa_bringup.service

Map Generation:

1. sudo systemctl stop jethexa_bringup.service
2. roslaunch jethexa_slam jethexa_slam.launch slam_methods:=gmapping
3. roslaunch jethexa_slam jethexa_slam_rviz.launch
4. roslaunch jethexa_slam jethexa_keyboard_control.launch
5. roscd jethexa_slam/maps
6. rosrun map_server map_saver -f map_name map:=/jethexa/map

Navigation:

1. roslaunch jethexa_navigation jethexa_load_map.launch map:=map_name
2. roslaunch jethexa_navigation jethexa_navigation.launch
3. roslaunch jethexa_navigation jethexa_navigation_rviz.launch

4 Flowchart:Ros Launch Breakdown

Along with the goal of using autonomous motion in the JetHexa platform, the other goal for the project was to gain more knowledge and experience with the Robot Operating System (ROS). One of the most prominent advantages of ROS is its modularity; multiple packages and configurations may be run through the use of .launch files, which are essentially XML-formatted files that specify what packages and nodes to run and what values to pass to do so [8]. This allows users to combine seemingly disparate ROS packages into a system that creates the desired outcome.

Hiwonder utilizes .launch files to coordinate the order of subsystem process initializations. For this section, the *jethexa_navigation.launch* file will be analyzed to understand the order of operations[3].

The .launch file begins by declaring multiple arguments (basically variables) and sets them to default values. It then enters into launching the subsystems. First, the URDF robot model is loaded. Next, the lidar is booted by finding and running the relevant lidar .launch file. After the model is loaded and lidar initialized, the motion controller is launched, with both the main controlling program and the joystick control being used. This means the user will be able to control the robot via software and/or manually via control stick. Finally, the last step in launching the initial peripherals is to begin the lidar's odometry simulation and integrate the relevant filter.

After these files are launched, the file begins launching the material relevant to navigation. First, an *amcl* node is launched for localization purposes. Next is the path-planning algorithm via the use of *move_base*. Finally, to enable multi-point routing, the *publish_point.py* script is run, creating an additional ROS node. This end of this step marks the conclusion of the *jethexa_navigation.launch* file. Other launch files such as the *jethexa_slam.launch* file can also be studied the same way. A flowchart of the launch calls is displayed below in Figure 5, and the full *jethexa_navigation.launch* file is found appended to this report.

[3]This analysis assumes knowledge of the basic structure of .launch files which can be studied at the following link: https://wiki.ros.org/roslaunch/XML

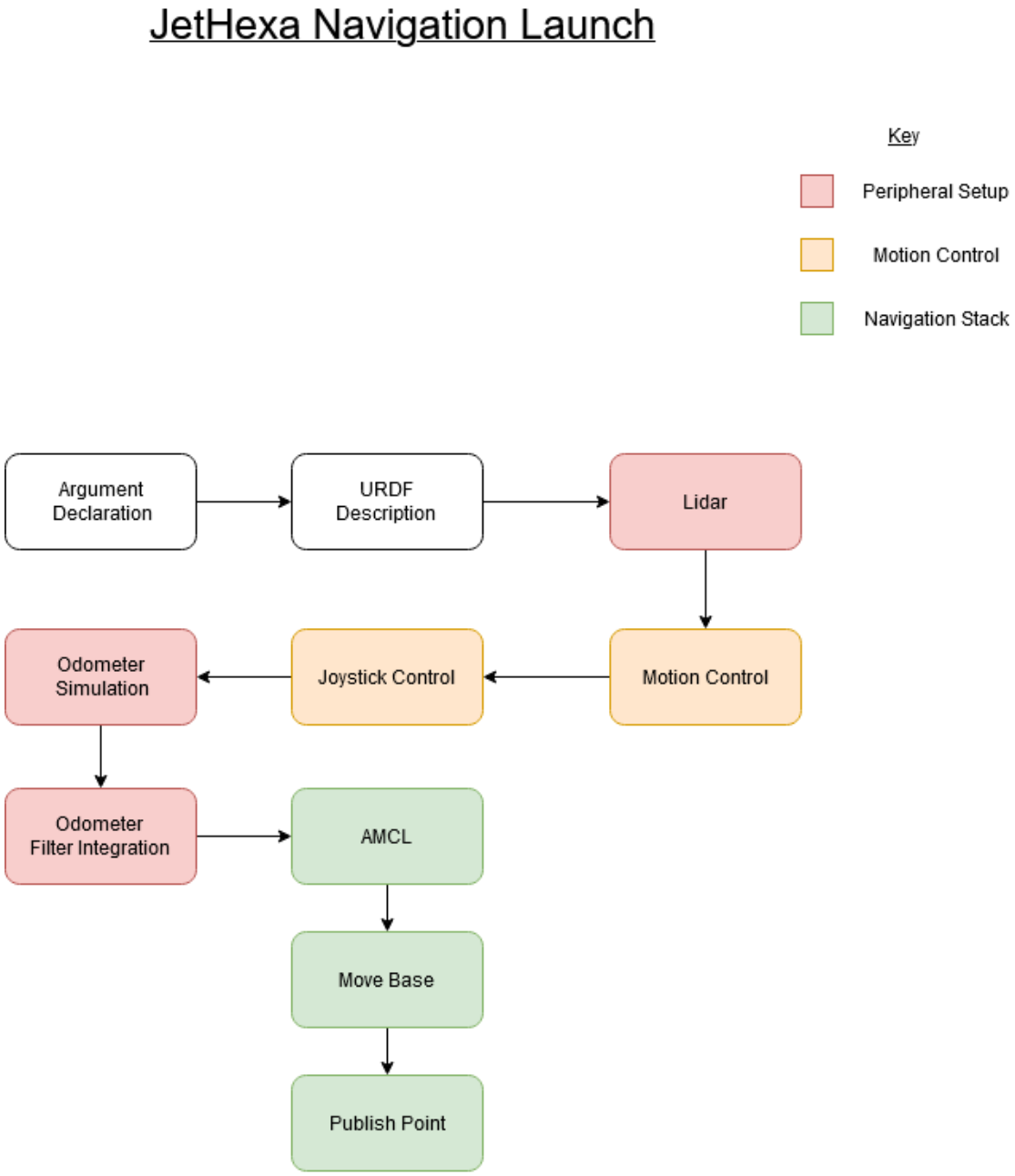


Figure 5: General order of execution of *jethexa_navigation.launch*

5 Conclusion

In this report, the work for Project 2 was discussed, specifically the structure of the JetHexa Navigation package and the entire process of autonomous motion from SLAM map generation to point publishing. It was determined that while the process itself was smoothly implemented, the JetHexa currently collides with obstacles placed in its path mid-operation. The aim for Project 3 is to remedy this behavior, allowing the JetHexa to avoid obstacles in real-time by adjusting its path on-the-fly.

References

[1] Hiwonder, *JetHexa Tutorials 12.1.2 - Basic Lidar Knowledge*, Hiwonder.

[2] Hiwonder, *JetHexa Tutorials 12.1.1 - Introduction to Lidar*, Hiwonder.

[3] *GMapping*, https://openslam-org.github.io/gmapping.html, Accessed: 2023-03-30, OpenSLAM.

[4] Hiwonder, *JetHexa Tutorials 12.2.1 - Adaptive Monte Carlo Localization*, Hiwonder.

[5] Hiwonder, *JetHexa Tutorials 12.1.6 - Gmapping Mapping Algorithm*, Hiwonder.

[6] Hiwonder, *JetHexa Tutorials 12.2.2 - Single-point Navigation and Obstacle Avoidance*, Hiwonder.

[7] Hiwonder, *JetHexa Tutorials 12.2.3 - Multi-point Navigation and Obstacle Avoidance*, Hiwonder.

[8] *roslaunch/XML*, https://wiki.ros.org/roslaunch/XML, Accessed: 2023-03-30, Open Robotics.

```xml
<?xml version="1.0"?>
<launch>
    <arg name="master_name" default="$(env MASTER_NAME)"/>
    <arg name="robot_name" default="$(env ROBOT_NAME)" />
    <arg name="tf_prefix" default="$(arg robot_name)" />

    <arg name="map_topic" default="/$(arg master_name)/map"/>
    <arg name="map_frame_id" default="$(arg master_name)/map"/>

    <arg name="use_depth_camera" default="false"/>
    <arg name="scan_topic" default="scan/raw"/>
    <arg name="odom_topic" default="odom/filtered"/>
    <arg name="cmd_vel_topic" default="jethexa_controller/cmd_vel"/>
    <arg name="odom_frame_id" default="$(arg tf_prefix)/odom"/>
    <arg name="base_frame_id" default="$(arg tf_prefix)/base_link"/>
    <arg name="move_base_result" default="move_base/result" />
    <arg name="clicked_point" default="clicked_point" />

    <group ns="$(arg robot_name)">
        <!-- URDF description of robot -->
        <include file="$(find jethexa_description)/launch/description.launch">
            <arg name="robot_name" value="$(arg robot_name)" />
            <arg name="tf_prefix" value="$(arg tf_prefix)" />
        </include>

        <!-- Lidar -->
        <include unless="$(arg use_depth_camera)" file="$(find
        jethexa_peripherals)/launch/lidar.launch">
            <arg name="tf_prefix" value="$(arg tf_prefix)" />
            <arg name="scan_topic" value="$(arg scan_topic)" />
        </include>

        <!-- robot motion control -->
        <include file="$(find jethexa_controller)/launch/jethexa_controller.launch">
            <arg name="robot_name" value="$(arg robot_name)" />
            <arg name="tf_prefix" value="$(arg tf_prefix)" />
            <arg name="tf_enable" value="false" />
            <arg name="odom_enable" value="false" />
        </include>

        <!-- handle control -->
        <include file="$(find jethexa_peripherals)/launch/joystick_control.launch" />

        <!-- posture sensor -->
        <!--
            <include file="$(find jethexa_peripherals)/launch/imu.launch">
                    <arg name="tf_prefix" value="$(arg tf_prefix)" />
                    <arg name="freq" value="50" />
            </include>
            -->

        <!-- Lidar simulates odometer -->
        <include file="$(find jethexa_slam)/launch/include/rf2o_laser_odometry.launch">
            <arg name="scan_topic" value="$(arg scan_topic)" />
            <arg name="odom_topic" value="odom/laser" />
            <arg name="odom_frame_id" value="$(arg odom_frame_id)" />
            <arg name="base_frame_id" value="$(arg base_frame_id)" />
            <arg name="laser_frame_id" value="$(arg tf_prefix)/laser_link" />
        </include>

        <!-- odometer integrates filter -->
        <include file="$(find jethexa_slam)/launch/include/jethexa_ekf.launch">
            <arg name="tf_prefix" value="$(arg tf_prefix)" />
            <arg name="robot_name" value="$(arg robot_name)" />
        </include>

        <!-- start the AMCL Adaptive Monte Carlo Localization algorithm package -->
```

EEL5683 Project 3 Report

Terrance Williams

April 23, 2023

1 Introduction

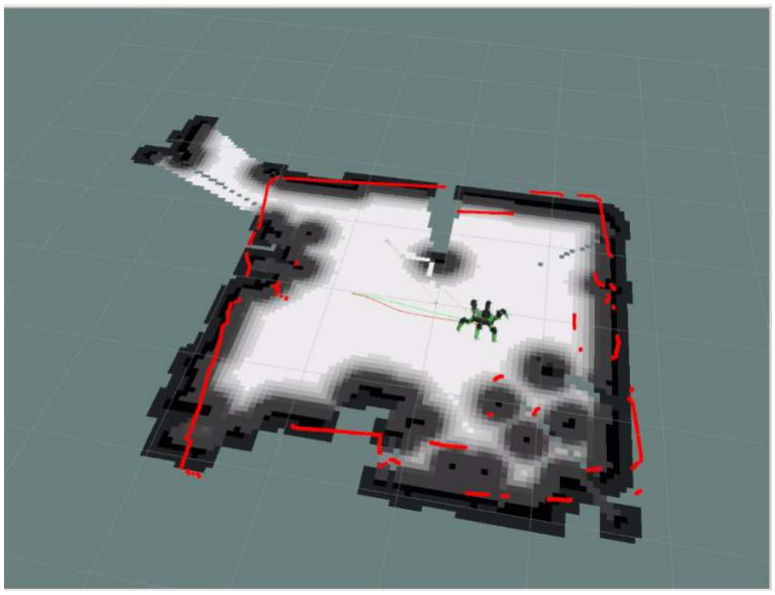
For the past semester, the Hiwonder JetHexa robot has been used to explore its range of features and performance capabilities. In Project 1, the hexapod's motion control, as well as its basic perception and navigation subsystems were tested, utilizing the WonderAi app as well as the provided Bluetooth controller. Basic movement, predefined action sequences, and the robot's lidar system were the subjects of study. In Project 2, tests transitioned to being conducted primarily through the Robot Operating System (ROS) by way of remote desktop control (NoMachine). SLAM mapping, localization, and navigation methods through the JetHexa Navigation package were used to test the robot's autonomous navigation. During this testing, it was found that the robot would collide with obstacles placed directly along its planned path; this behavior is the subject of Project 3.

The goal for Project 3 was to correct this behavior, enabling real-time obstacle avoidance. This was done through additional layers in the robot's global and local costmaps.

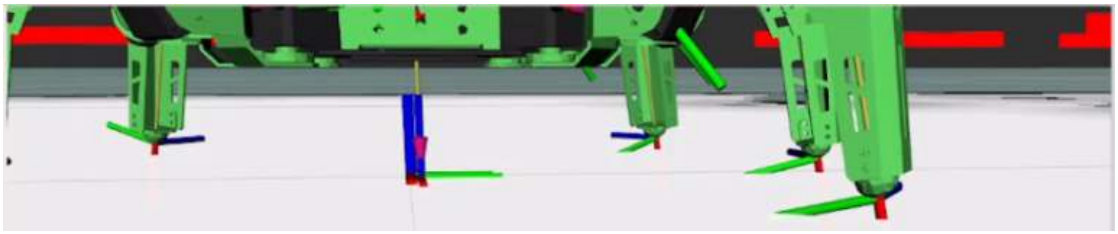
2 Note on Path Trajectory

Before detailing the above, one question received as a result of Project 2 concerned the trajectory the JetHexa robot takes to a provided goal point. It was observed that the robot would take a curved path even in cases where straight paths seemed the more efficient, which could prove to be problematic in terms of predicted behavior.

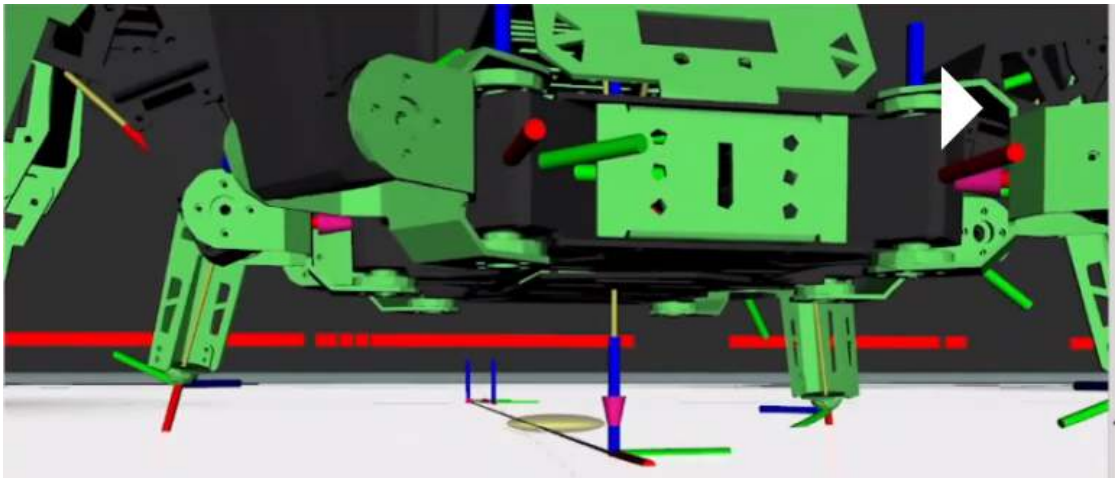
After investigation, an explanation for this behavior may lie in the *way* the robot navigates itself. Upon map generation, a coordinate frame is saved, recording the robot's position and orientation at the time the map's created. Upon moving to a goal point, the robot not only aims to move to the (x, y) position of the goal



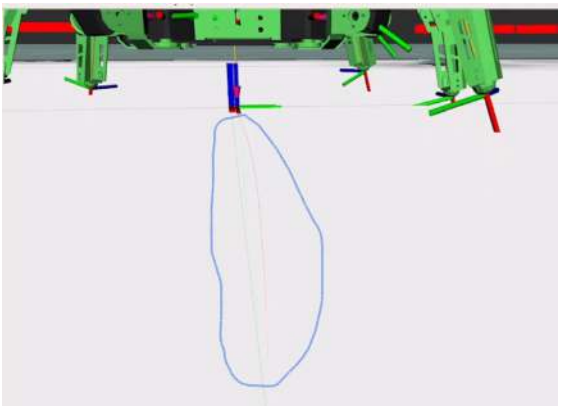
point, it does so while ending its central frame in an orientation parallel to the map frame. If a goal point is provided that is a direct-line path ahead of the map origin frame, the robot proceeds in a (mostly) straight trajectory, as seen below. The trajectory is thinly plotted, but the green line is the straight-line trajectory and the red line is the trajectory planned by the robot. Both are enclosed by the blue curve. It can be seen that the two trajectories are pretty close to one another.



(a) Before movement. Robot central frame is aligned with map frame



(b) After movement. Robot central frame is aligned with map frame



(c) Trajectories (circled in blue)

3 Cost Map Layers

JetHexa's navigation package uses the ROS Navigation Stack to achieve its behavior. In terms of path planning and execution, the package uses the **move_base** package. This package relies on another package in the Navigation Stack, **costmap_2d** [1]. If **move_base** is the means of executing a navigation plan, **costmap_2d** provides the data for a plan to be formed.

According to the Package Summary, **costmap_2d** takes in sensor data as its input—laser scans in the case of this project—and constructs an occupancy grid with inflated costs based on user-configured inflation radius [2]. The occupancy map can be either 2D or 3D (voxel), but this project uses the 2D representation. The package creates layered costmaps using three layer plugins: static layer, obstacle layer, and the inflation layer. Each layer provides additional data that informs the navigation task.

3.1 Static Layer

The static layer is the aptly-termed layer that utilizes unchanging data. This data is sourced externally, typically through provision of a map [3]. This layer appears to be the "base" layer onto which the other two layers add their data.

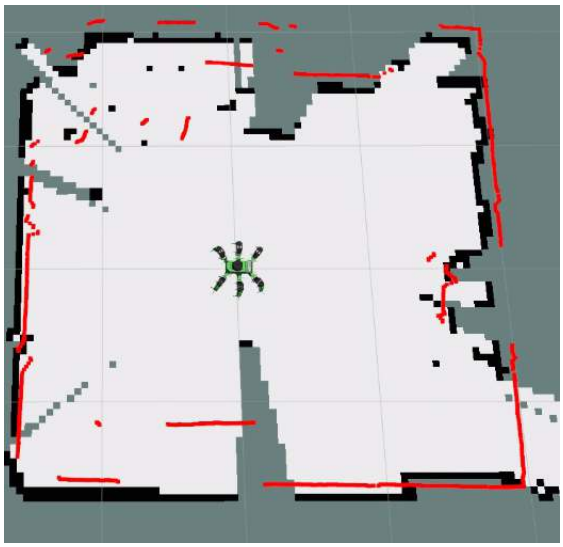


Figure 2: The static layer

3.2 Obstacle Layer

The obstacle layer [4] serves to mark and clear obstacles. Given sensory data (ex. LaserScan), this layer identifies and tracks obstacles in the environment, marking them by iinserting information to the costmap when they are within the scanning range of the sensor and clearing them when outside that range by removing said data. The latter operation is done via raytracing. When superimposed onto the static layer, the obstacles are highlighted in two colors: red for the static layer and magenta for the obstacle layer.

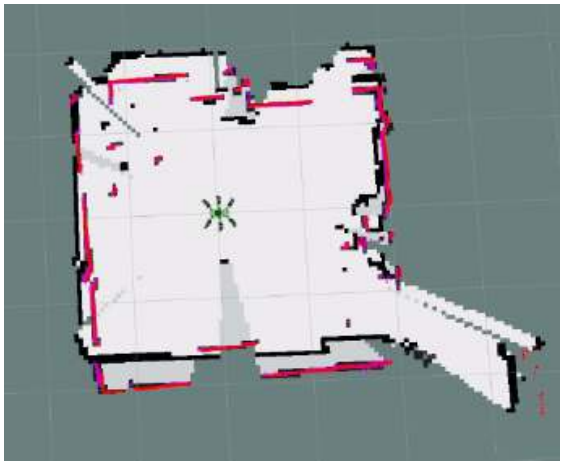


Figure 3: Rviz rep. with both static and obstacle layers enabled

3.3 Inflation Layer

The inflation layer is the layer that accounts for robot radius. The JetHexa does not run a footprint model by default, so the navigation program assumes a point model. Because the robot is represented as a point, the obstacles and other obstructions must be inflated by values consistent with the robot's actual dimensions to preclude collisions.

This layer "sits" on top of the static and obstacle layers and categorizes costs in the occupancy grid in five ways [5]. When a cell in the grid *actually* contains the physical obstacle, the cell gets the "Lethal" cost, the highest cost value. If the robot is in this cell, a collision has definitely occurred. Propagating out from this cell is a circle of inscribed radius. Cells within this area are given the "Inscribed" cost, which means that the robot will make a collision with certainty if its center is in any of these cells.

Next is the "Possible circumscribed" cost. This cost is associated with cells located within the circumscribed circle about the robot's footprint but outside of

the more lethal cost areas. If the robot's center is in these cells, it hasn't necessarily made a collision, but it may do so depending on the robot's orientation. After implementation of the three layers, most collisions observed were a result of cells with this cost. Finally, the robot has "Freespace" and "Unknown" costs. The former means the robot can traverse the cell with no problem. The latter's interpretation is user-discretionary.

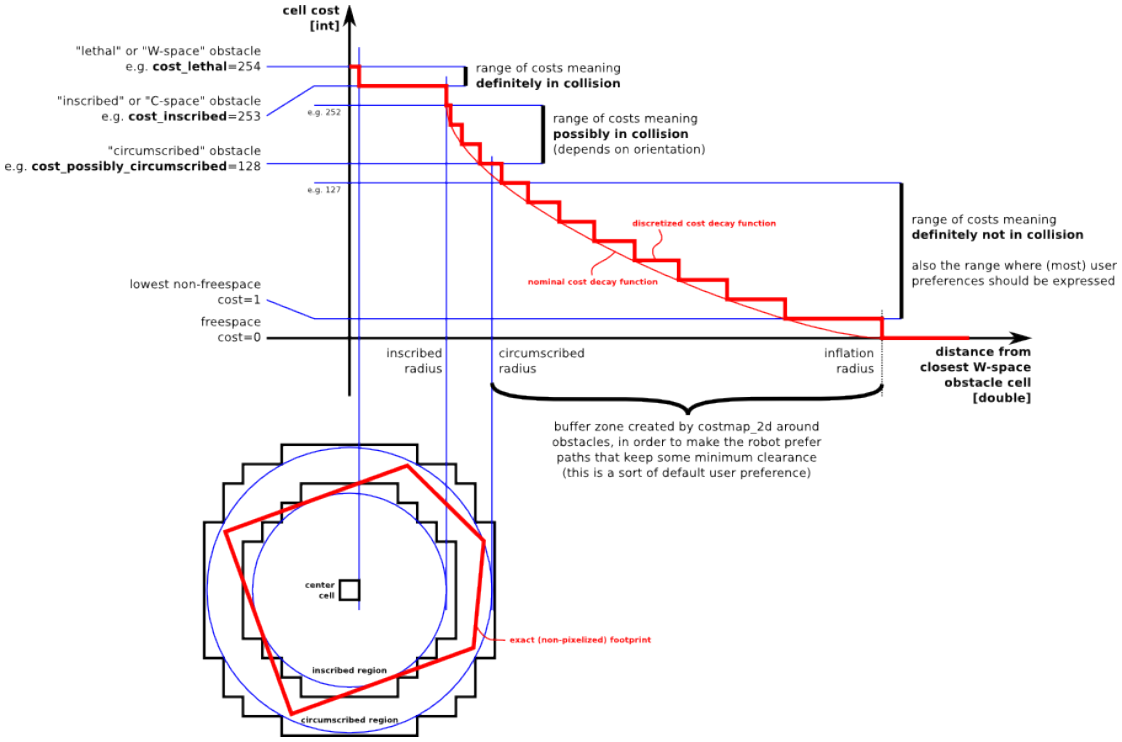
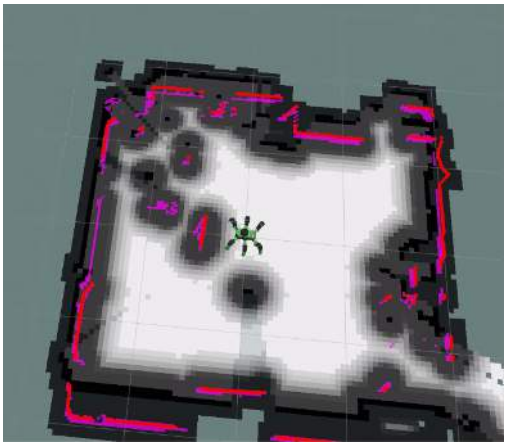
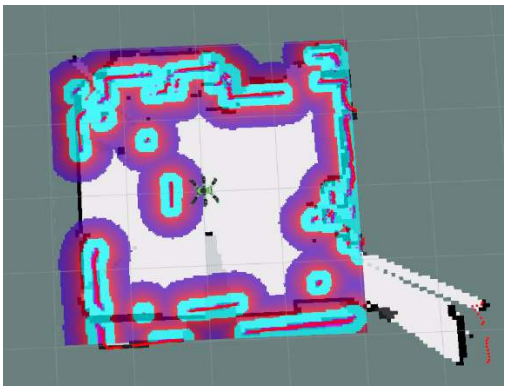


Figure 4: Open Robotics' Inflation Layer Diagram [5]

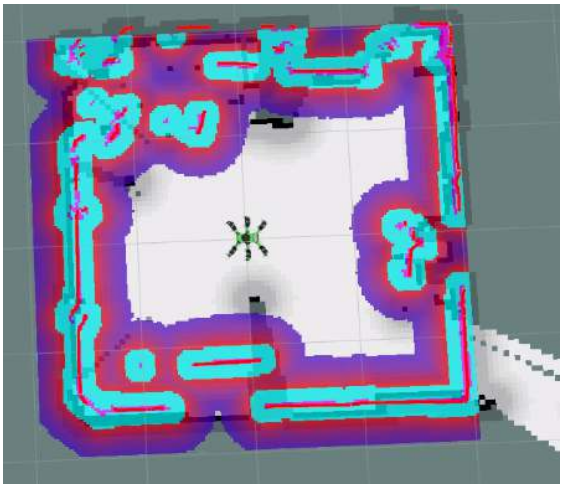
Interestingly, the navigation program has inflation layer parameters for both the global and local map. After performing tests, it was determined that they do affect the map differently, but it is uncertain what the technical difference may be. Visually, the global inflation layer consists of grayscale entries on the map while the local layer is colored in teal (Inscribed cost) and purple (Possibly circumscribed cost).



(a) Global



(b) Local

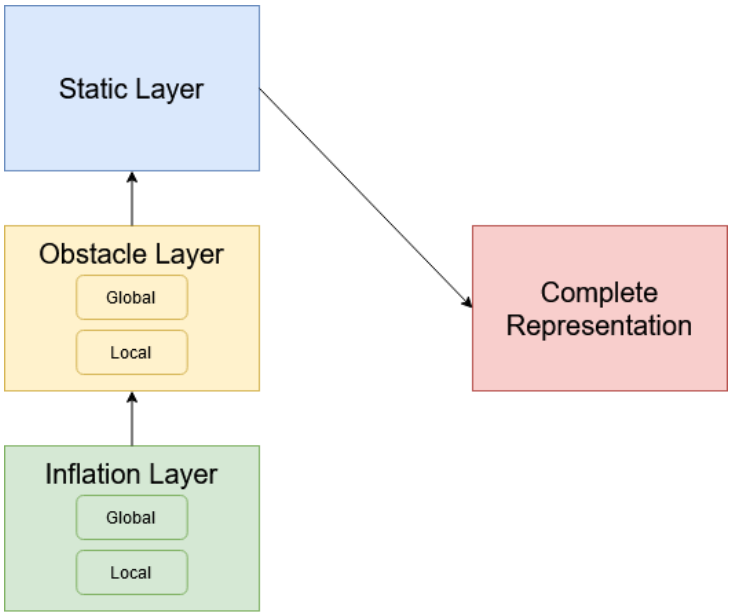


(c) Combined (all layers enabled)

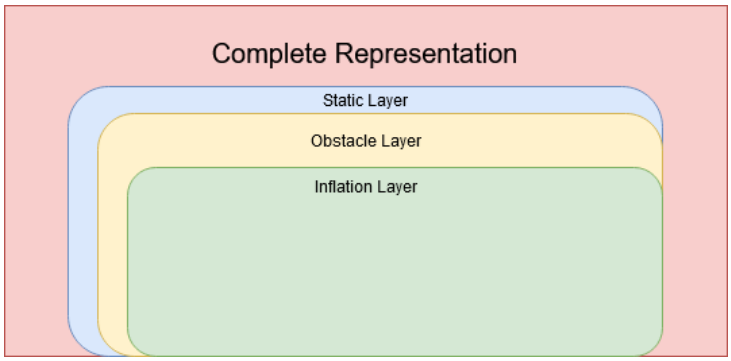
Figure 5: Inflation Layer Types

3.4 Chart

A visual representation of the layers' contribution to a cost representation of the environment is displayed below.



(a) View 1



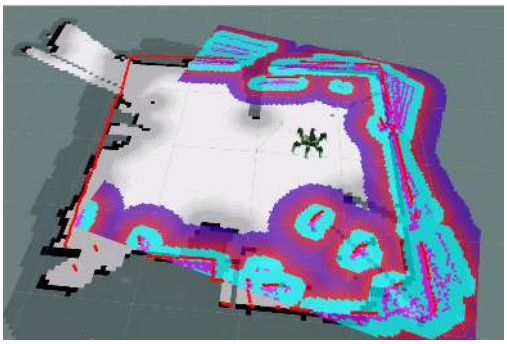
(b) View 2

Figure 6: The three costmap layers

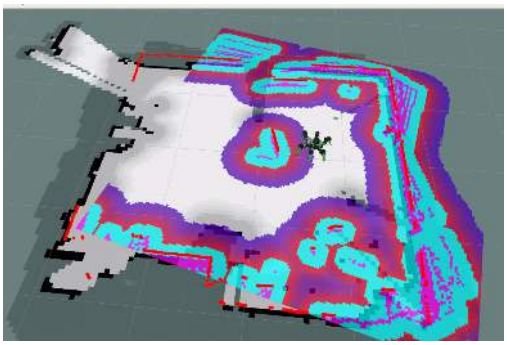
4 Results

Enabling the additional map layers resulted a more robust obstacle avoidance by the JetHexa. The robot is able to detect an object placed in its path and alter its planned trajectory to navigate around it. One thing noticed, however, was the size of the obstacles as a result of the inflation layer. Two obstacles that have space between them may have a representation that does not capture this spacing. The inflation layer applied to the two objects can result in the obstacles "overlapping" on the map. This results in a loss of fidelity when compared to the physical environment.

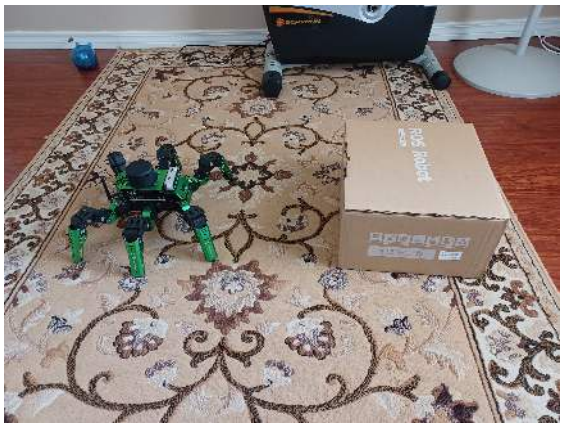
To account for this phenomenon, one could decrease the inflation radius parameters in the relevant costmap parameter files or relegate the use of the JetHexa for more spacious environs, leading to more space to navigate around said objects.



(a) No obstacle



(b) With Obstacle



(c) Size comparison for reference

Figure 7: Effect of the Inflation Layer

5 Future Work

The JetHexa has potential for many research applications in terms of traversability, perception, localization, and a host of other topics. There are many features left to explore such as the robot's ability to communicate with other JetHexas, opening the door for swarming or "battalion"-style behavior.

On the practical side, if possible, it may be beneficial to find a replacement battery with larger capacity to extend battery life/charge cycle. Currently, the robot's battery lasts about an hour before needed to be recharged, which prove cumbersome when performing experiments and trials.

Overall, however, the platform is, quite frankly, fun to work with and provides a great opportunity to work with advanced features.

References

[1] *move_base*, http://wiki.ros.org/move_base?distro=noetic, Accessed: 2023-04-20, Open Robotics.

[2] *costmap_2d*, http://wiki.ros.org/costmap_2d, Accessed: 2023-04-20, Open Robotics.

[3] *costmap_2d/hydro/staticmap*, http://wiki.ros.org/costmap_2d/hydro/staticmap, Accessed: 2023-04-20, Open Robotics.

[4] *costmap_2d/hydro/obstacles*, http://wiki.ros.org/costmap_2d/hydro/obstacles, Accessed: 2023-04-20, Open Robotics.

[5] *costmap_2d/hydro/inflation*, http://wiki.ros.org/costmap_2d/hydro/inflation, Accessed: 2023-04-20, Open Robotics.