Kinematics, Pose Control, and Motion Planning of Mobile Hexapod Robot

S. Adam Stringham

April 26, 2023

Abstract

Mobile hexapod robots offer versatility when navigating difficult terrain environments, but their over-actuated nature can lead to difficult kinematics solutions. We developed a solution by creating an omni-directional walking gait system with dynamically adjustable parameters. Under this system, the body pose orientation can be chosen independently of the gait foot trajectories, and the pose can be controlled using classic PID control schemes. We applied geometric-based kinematic constraints to the body pose to ensure kinematic feasibility such that all legs can maintain their workspace, while maximizing the range of body pose motion. Open-source navigation and path planning tools have been integrated with this system to provide a global motion planning approach. All of these techniques have been implemented using ROS1 Noetic software and modeling techniques, and validated on a physical, affordable hexapod robot.

1 Introduction

Recent advancements in computing and manufacturing have made hobbyist robotics more affordable than ever. Powerful processors and actuators can be obtained on a budget and have paved the way to make robotics accessible to hobbyists and researchers alike. Low-cost sensor hardware has also followed the trend of becoming more affordable and available, which allows mobile robots to accessibly understand and interact with their environment. While the hardware is now easily obtainable, the software and logic of a robot can be more difficult to implement.

This project uses the Freenove Big Hexapod robot kit [4] named "Wanda" as an affordable mobile haxapod robot platform platform. A combination of off-the-shelf (OTS) sensors and servo actuators are used to collect information and move the robot through the environment. A budget-friendly single board computer is used to control the system using custom Robot Operating System (ROS) [15] software packages.

This project builds on work done previously by the author to add capabilities to this platform. Specifically, this project focuses on three objectives:

1. Geometrically constrain the body pose to maximize the workspaces of the body pose and the feet, while preventing kinematically infeasible configurations from being commanded to the system.
2. Implement a body pose controller that maintains a desired orientation relative to inertial gravity.
3. Implement a global motion planning approach to allow the robot to navigate a cluttered environment.

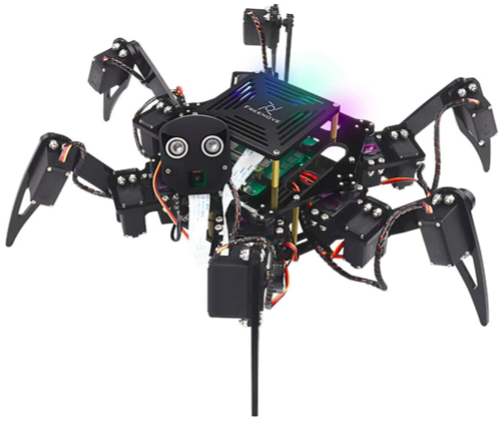

Figure 1: Freenove Big Hexapod robot kit

2 Related Work

The fundamentals kinematics and motion of mobile hexapod robots are well understood in the literature [19] [6]. Body pose estimation is done using various sensor schemes [8]. With robust locomotion possible, omni-directional motion planning has been implemented on hexapod robots before [9] [2].

2.1 Previous Projects

Wanda has been used as a project platform by the author since summer 2021 for graduate course work semester projects. Each semester, new capabilites were added to the system. A summary of this timeline is as follows.

- Summer 2021: the robot was first assembled and calibrated. Simple kinematics were established, the operating system was set up, and the hardware and software were able to communicate.

- Fall 2021: a simple forward wave gait was implemented on the robot to allow it to walk forward in a straight line. The sensors were used to detect its environment.
- Winter 2022: all software was ported over to ROS1 Noetic. A more robust, omni-directional walking gait was implemented. The camera was used to identify and follow simple blue targets.
- Summer 2022: manual teleoperation was introduced using an Xbox controller. This allowed direct user control of the walking direction, the seeker angles, and all 6 degree of freedom (DOF) of the body pose, with very simple and limiting constraints on the commanded pose.
- Fall 2022: distributed processing was set up to offload much of the computational processing to a powerful desktop, allowing the embedded computer to run under less load. ORB_SLAM3 [1] was implemented on the hardware.

The robot uses a forward wave gait to walk. The system receives a commanded velocity twist vector defined in the ground frame below the robot, ξ_g, with linear x and y and angular yaw ψ components. A gait trajectory algorithm increments the position of each foot by translating and inverting this twist into each foots current position and integrating with the time step dt. Transition between stride and support phase is scheduled by incrementing a master stride phase based on how far all of the feet move away from the center of their workspaces. When a foot enters stride phase, the foot is picked up off the ground and moved back to the center of its workspace relative to the ground frame. This motion is executed as a polynomial trajectory with zero initial and final velocity and acceleration in the $\hat{z}$ direction.

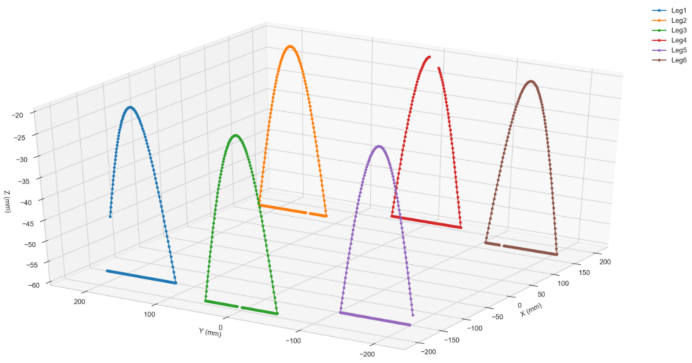

Figure 2: Foot trajectory for simple forward walking gait

The pose of the body can be independently controlled from the feet trajectories. This pose is calculated relative to the ground frame under the robot. The desired position of each foot and the body pose are then both passed to a simple inverse kinematics solver, which returns the commanded joint angles. Thes angles are then sent to the servos to actuate the robot. This scheme can be seen in Figure 3.

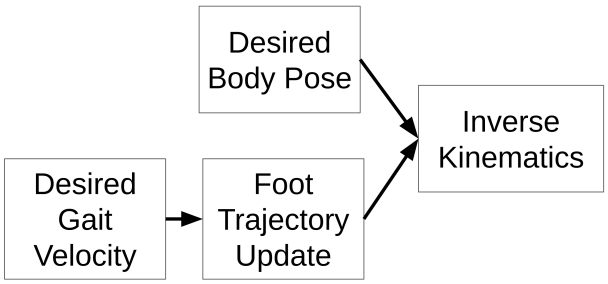

Figure 3: Previous walking gait design block diagram

3 Robot System

A physical hexapod robot system was used to design and test the methods presented in this paper. The hardware and software used on the robot are described below.

3.1 Hardware

The physical hardware that made up the robot computational system are described in this section.

3.1.1 Hexapod

This project is based around the Freenove Big Hexapod kit. This OTS kit comes with a six-legged robot chassis, where each leg has three degrees of freedom (DOF) that can be individually controlled. Each joint is actuated by an individual MG-996R servo motor, which can be commanded to obtain joint angles of $0 - 180^{\circ}$ given a 50 Hz pulse width modulation (PWM) signal. To manage communication to all 18 leg servos, the servos are controlled by a PCA9685 servo driver module, which has a standard Python library to allow software control of the PWM signals.

3.1.2 Embedded Processor

The Raspberry Pi (RPi) 4 acts as the main processor of the robot assembly. The RPi's CPU is a Broadcom BCM2711B0 with stock clock speeds of 1.5GHz, which has been over-clocked to 1.9 GHz for improved performance. The model RPi used in this project includes 8MB of DDR RAM. A heat sink and active fan are used to keep CPU temperatures manageable during strenuous processing. The RPi is also setup to boot from an external SSD connected through USB 3.0, rather than the typical SD card boot hardware; this improves read-write speed and boot times.

Figure 4: The Raspberry Pi 4 single-board computer

3.1.3 Seeker Gimble

The robot is equipped with a two-axis gimbal called the "seeker". The azimuth and elevation angles can be individually controlled with two MG90S servo motors receiving signals from the RPi via the motor driver board, similarly to the leg servos. The seeker is used to point an ultrasonic range sensor and a camera mounted to it; the camera is not used in this project.

3.1.4 Ultrasonic Range Sensor

The HC-SR04 is an OTS ultrasonic range sensor that provides contactless ranging of objects from 2cm to 400cm away from the seeker [5]. The module emits a series of pulses at 40kHz that reflect on objects in front of the sensor. The sensor detects an echo reflected off an object, and the time between when the signal is sent and received can be used to estimate a range to the object.

Figure 5: HC-SR04 ultrasonic range sensor used on the Freenove Hexapod

General purpose input-output (GPIO) pin 27 on the RPi is used to trigger the pulse, while pin 22 is used to register a detected echo. The HC-SR04's VCC and GND pins can be connected to any +5V and GND pin of the GPIO Module.

3.1.5 Inertial Measurement Unit (IMU)

The MPU-6050 is an affordable, OTS inertial measurement unit (IMU) that is capable of providing linear acceleration, angular velocity, and temperature measurements to the Raspberry Pi at 50 Hz [17]. The IMU is hard-mounted to the motor driver board and is located just underneath the RPi. The IMU connects to the RPi via I^2C on the RPi's GPIO module.

This low-cost IMU presents inconsistent bias values for each signal which must be removed to prevent pose estimation drift. To perform this, a simple calibration routine was implemented in the IMU node which assumes the robot body is static and level with the ground. The sensor is polled for some fixed time period (0.5s), and an average of all readings is computed for all accelerometer and gyroscope channels individually. Gravitational acceleration of $9.81 m/s^2$ is removed from the $\hat{z}$ component of the accelerometer. These bias values are then removed from all future IMU measurements before publishing the data out as a ROS topic. Calibration is automatically performed at sensor boot time, but this routine can be performed again at any time without disrupting the flow of published data.

3.1.6 Desktop Computer

To aleviate the computational burden on the RPi, the distributed processing capabilities of ROS1 are exploited for this project by using a desktop computer to supplement the processing. This computer contains an AMD Ryzen 7 3800X 8-core CPU, an Nvidia RTX 2060 Super graphics processor, and 32Gb of onboard RAM running Ubuntu 20.04 LTS. The RPi ran all hardware-dependent ROS nodes, namely the ultrasonic range sensor and IMU nodes and the servo actuator nodes. All other ROS nodes, the ROS core, and any extra analysis tools, were run on the desktop machine. Communication was handled wirelessly using the local network WiFi.

3.2 Software

The RPi embedded on the robot runs with the Ubuntu Mate 20.04 LTS operating system. All control software for the robot is written as ROS1 Noetic [15] nodes, written in Python and C++. A mix of custom written and open-source ROS nodes were used, as described in later sections.

4 Methods

This project focused on three primary areas related to this hexapod robot: analytically constraining the body pose orientation, controlling the body pose, and implementing global motion planning. The approach for each of these tasks are described in this section.

4.1 Body Pose Kinematic Constraints

The over-actuated nature of hexapod robots provides opportunity for great flexibility in controling the body pose orientation, in addition to manipulating the legs. However, kinematically infeasible configurations could easily be commanded to the system. If not well handled, these unachievable body poses could cause errors in inverse kinematic calculations, singularities in the legs, or conflicts in leg and body workspaces. This paper presents a simple geometry-based system to dynamically constrain and

maximize the body pose and feet workspaces to feasilbe configurations. This system builds on the gait design and inverse kinematics previously described.

First, the center of each foot's workspace, $\vec{F}_c$, must be defined, where the workspace is a circle with radius R. For this system, $\vec{F}_c$ is set along the unit vector from the body center each legs shoulder, $\hat{S}$, a distance of of $L_1 + L_2$ away from the shoulder joint q_2 when the body pose is set to the identity transform, where L_1 is the coxa length and L_1 is the femur. Note that R is the same for all legs.

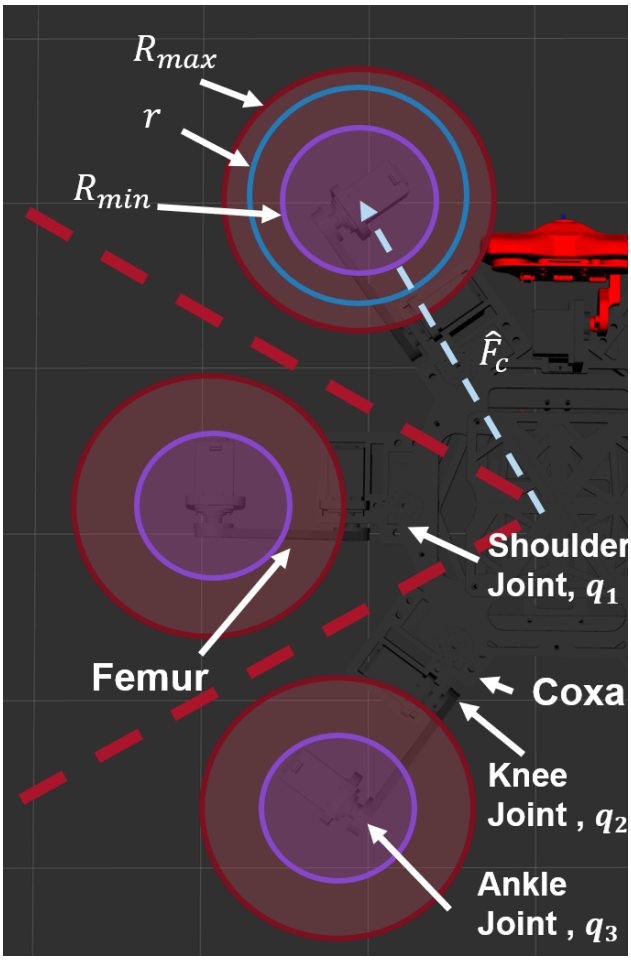

Figure 6: Foot workspace bounds, R_{min} and R_{max}, foot workspace center, and joint labels

The maximum feasible foot workspace radius R_{max} is then set as the minimum value after applying the following constraints:

- The workspaces shall not overlap, ensuring the feet do not contact each other while walking.
- The workspace shall not overlap with the coxa when the body is in the identiy pose, ensuring the feet do not go under the first joint. This is done both for system stability and for ease in solving inverse kinematics.
- The outer edge of the workspace must be reachable by the foot given an identity body pose with the belly on the ground, so it must be at most $L_2 + L_3$ away from the second joint, where L_3 is the length of the tibia.

Using the above conditions, the workspace radius R_{max} could be such that the leg must be fully extended for the foot to reach the outside edge of the workspace. This results in an undesirable joint singularity. To avoid this, the chosen magnitude of R_{max} is scaled down by some value < 1; a value of 0.9 was applied for this project.

In addition to the maximum foot workspace, the minimum workspace R_{min} must also be defined. This ensures that the feet have some room to move while walking. This project sets R_{min} as half the size of R_{max}.

These minimum and maximum workspace bounds for all feet are used to check if a desired body pose configuration is valid. To prevent an invalid body pose command, any desired body pose hypothesis must pass all of the following conditions:

- The belly of the robot cannot collide with the ground. The z element of each shoulder position in the body frame cannot be less than 0 when transformed to the ground frame.
- The maximum edge of each foot workspace must be within reach of each leg (must be less than distance $L_2 + L_3$ away from the knee joint, q_2).
- The knee joint of each leg must not be less than L_1 away from the closest point on the minimum workspace projected into the planar shoulder frame, ensuring the foot does not go under the coxa.

If all of these conditions are met for all legs, the desired body pose is treated as valid and allowed to be sent to the rest of the system.

4.2 Dynamic Stride Length

Once the desired body pose is calculated and accepted as valid, the actual stride length r for all of the legs must be set, bounded between $R_{min} \leq R_{max}$. Two bounds are checked to set r, with the smallest result being the new actual stride length:

- The magnitude of the distance from $\vec{F}_c$ to the knee joint, projected onto the ground. The smallest value for all legs is chosen.
- The farthest each leg can reach out from $\vec{F}_c$, with the shortest reach being used.

As the commanded body pose changes, the workspace of all feet changes to ensure the body does not collide with the feet and the feet can reach anywhere inside their workspace. Workspace changes are passed into foot trajectory update algorithm, as seen in Figure 7, and the walking gait phase is naturally updated as the size of the foot workspace grows and shrinks.

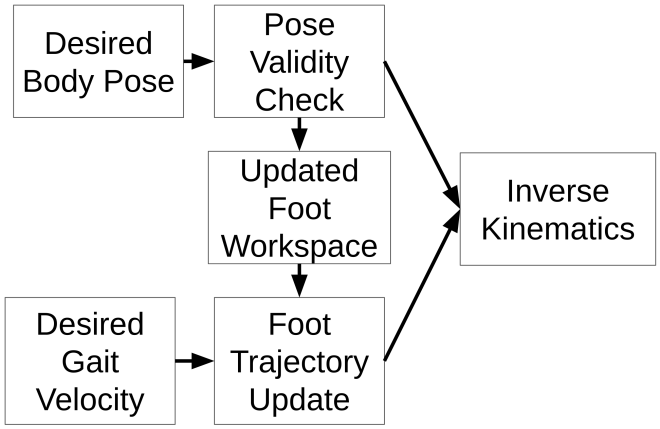

Figure 7: Functional block diagram for body pose and foot trajectory kinematics

4.3 Dynamic Workspace Centers

The location of the workspace centers $\vec{F_c}$ need not be static. As these positions change, a new F_{min} and F_{max} must be recomputed as above, and the stride length r should similarly be updated. Feasible values of $\vec{F_c}$ should be checked before updating them, by ensuring $\vec{F_c}$ is within reach of the foot. This project allows $\vec{F_c}$ to be incremented along $\hat{S}$, but other applications may move this elsewhere, like for maneuvering over rough terrain as in [3].

4.4 Body Pose Control

With the body pose constrained to kinematically feasible limits, a body pose controller can be implemented without fear of corrupting the kinematics. A PID controller was designed to maintain a desired roll θ_c and pitch ϕ_c orientation relative to gravity.

4.4.1 State Feedback

The embedded IMU is used to provide state feedback on the current body pose orientation relative to gravity. The raw IMU data is filtered using an open source Complementary Filter ROS node [7]. This node estimates the inertial orientation of the IMU relative to gravity. The measured roll θ_m and pitch ϕ_m channels are extracted from the quaternion orientation estimate. Yaw is ignored because the IMU does not have a magnetometer, which is important in providing an absolute yaw orientation estimate.

4.4.2 Controller Design

A classic PID controller is set up to obtain the desired inertial roll and pitch angles relative to gravity. After extracting desired and measured Euler angles from the quaternian-based commanded and measured orientations, the adjusted angles are calculated at new time $t+1$ as follows.

$$\Delta\theta_t = \theta_m - \theta_c \tag{1}$$

$$\Delta\dot{\theta}_t = \frac{\Delta\theta_t - \Delta\theta_{t-1}}{\Delta t} \tag{2}$$

$$\int \theta_t = \int \theta_{t-1} + \theta_t \Delta t \tag{3}$$

$$\theta_{t+1} = \theta_t + \Delta\theta_t K_p + \Delta\dot{\theta}_t K_d + \int \theta_t K_I \tag{4}$$

Where K_p, K_d, and K_I are the controller gains. Compensated roll and pitch angles are calcualted in the same manner. These angles, along with an uncompensated yaw angle, are then reconverted into quaternian form. This new compensated pose is then checked for validity as above, and if it is valid this new pose is sent to the rest of the system. This new pose is passed to the system using the ROS tf2 coordinate transformation manager [18].

4.5 Motion Planning

To take advantage of the mobility of this hexapod robot, a global motion planning approach was implemented, built on the open source ROS1 Navigation Stack [14].

4.5.1 Map

This 2D motion planning scheme takes advantage of an a priori environment map. An approximate map of the environment was created using a published floor plan, with furnature and other static obstacles added. This map is saved as a *.jpg* file and read in using the ROS map server [12]. This is used to generate a global cost map to inform global motion planning.

4.5.2 Odometry

For continuous navigation, the open source Robot Localization ROS package [10] was used to implement an extended Kalman filter (EKF) for pose estimation. This filter is fed a commanded twist velocity vector ξ_g in the ground frame, as well as IMU sensor data, and estimates the robot's pose relative to a fixed odometry frame over time.

This filter is tuned primarily by the covariance matrices σ^2 for each data source. The smaller the covariance for a given input source, the more the filter will bias towards that measurement. A custom ROS node was written in C++ to dynamically calculate the covariance of the linear x and y and angular yaw ψ components of the commanded walking velocity. If any component is commanded to be less than some small value, its uncertainty σ is set very small. Otherwise, the uncertainty is proportional to the magnitude of the velocity. The full 6x6 covariance matrix is the calculated as follows.

$$\sigma^2 = \begin{bmatrix} \sigma_x^2 & \sigma_x\sigma_y & 0 & \dots & 0 & \sigma_x\sigma_\psi \\ \sigma_x\sigma_y & \sigma_y^2 & 0 & \dots & 0 & \sigma_y\sigma_\psi \\ \vdots & & & \ddots & & \\ \sigma_x\sigma_\psi & \sigma_y\sigma_\psi & 0 & \dots & 0 & \sigma_\psi^2 \end{bmatrix} \tag{5}$$

The IMU data is similarly weighted by its covariance matrix, but this is not set dynamically. Instead, the magnitude of each sensor channel (accelerometer and gyrometer) is set statically and tuned based on the expected noise level of the sensor hardware.

4.5.3 Obstacle Avoidance

While the a priori map is useful for maneuvering around known static obstacles, the robot must be able to avoid obstacles not present on the map. The ultrasonic range sensor is used for this purpose. This sensor is polled at a rate of 20Hz. A simple filter is applied to the observables using a custom ROS node written in C++. To prevent spurious detections from confusing the system with fake obstacles, this filter maintains a rolling average of sensor observables. If 10 valid observables in a row are within a set range tolerance from each other, then it is assumed there is a real obstacle present. These observables are averaged to create a point in space. To account for the wide (30°) sensor beam, four additional points are added at the presumed edge of this beam, resulting in one point cloud for each integrated measurement. This point cloud is then published and passed into the navigation stack as an obstacle to avoid.

4.5.4 Seeker Control

While the ultrasonic range sensor beam has a wide field of view (FOV), it can still only sense within a limited space. To help the robot "look ahead", a seeker gimbal search system was set up. The nominal seeker gimbal angles are calculated based on the commanded walking twist ξ_g. The azimuth angle η is set to look in the direction the robot is walking, with extra bias if the robot is turning, as follows.

$$\eta = \arctan(\frac{\xi_y}{\xi_z}) + K_\psi \xi_\psi \tag{6}$$

where $K_\psi = 0.75$ linearly scales the azimuth angle based on how fast the robot is spinning. To minimize the chance of ground clutter return, the nominal elevation angle ν is set to point the seeker up off the ground.

To search for a wide field of view in front of the robot, these nominal gimbal angles are linearly modulated. η is dithered with a $\pm 20°$ field of view, sweeping at a rate of $7°/s$. ν is similarly dithered with a $\pm 6°$ field of veiw at a rate of $12°/s$. This allows the robot to look across more area in azimuth, while quickly searching up and down for obstacles. These seeker angles are then run through a simple first order digial low pass filter (LPF) to prevent large step change commands being sent to the servos.

4.5.5 Path Following

To calcualte a commanded velocity vector ξ_g, the ROS Move Base node was used [13]. This node uses the global map, the estimated robot pose from odometry, and measured obstacles to calculate both global and local paths for the robot to follow. These paths are constantly updated to give new commands to the robot. This node is designed to modularly accept different types of sensor data, as well as motion planning plugins. Only the default plugins were used for this project. To protect the servos from large step changes in desired velocity, the commanded velocity vector ξ_g is run through a simple digital LPF to smooth out the system commands before sending them to the foot trajectory update system.

4.5.6 Pose Initialization and Commands

The EKF solution needs to be initialized to a true location relative to the map. Rviz in ROS [16] has a built-in tool for this. With the click of a button, users can place and orient the "true" robot pose directly on the map, and that pose will be published to the *initialpose* ROS topic. The Robot Localization EKF subscribes to that topic to snap update the filter solution to the true location. Rviz has a similar tool to publish a goal pose. Users can click and orient the deisred pose, which is published to the *move_base/goal* ROS topic, which *move_base* subscribes to and begins solving.

5 Results

5.1 Constrained Body Pose

The geometric-based constraints on acceptable body pose orientations allows the robot to successfully obtain extreme poses without exceeding kienamtic limitations. An example configuration can be seen in Figure 8.

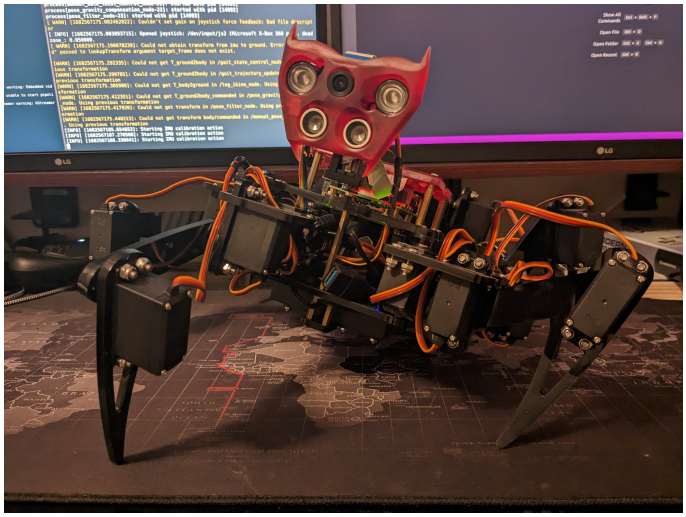

Figure 8: Example extreme body pose on robot hardware

By dynamically updating the feet workspaces based on the body pose, the robot is able to continue walking without interruptions due to changes in pose configuration. R_{min} and R_{max} also change dynamically as the locations of the workspace centers change. However, as the

workspaces shrink, the gait phase occupies less time, resulting in the legs transitioning betweeen stride and support mode faster.

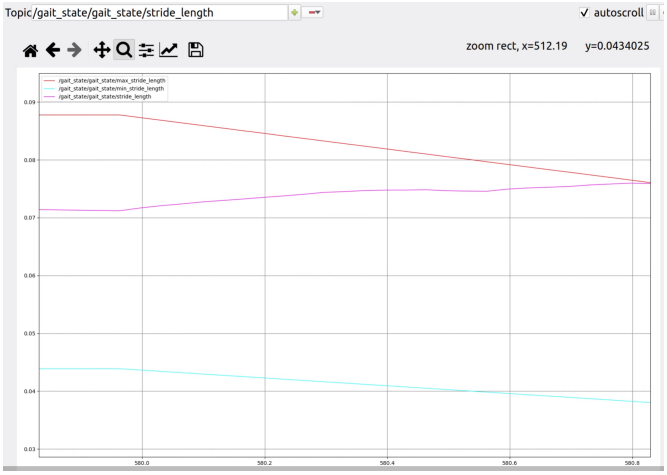

Figure 9: R_{min}, R_{max}, and r plotted from values on hardware, as the body pose and the workspace centers move

5.2 Body Pose Control

The following gain values were chosen through hand-tuning the system for speed and minimal overshoot.

$$K_p = 1.0, K_d = 0.02, K_I = 0.02$$

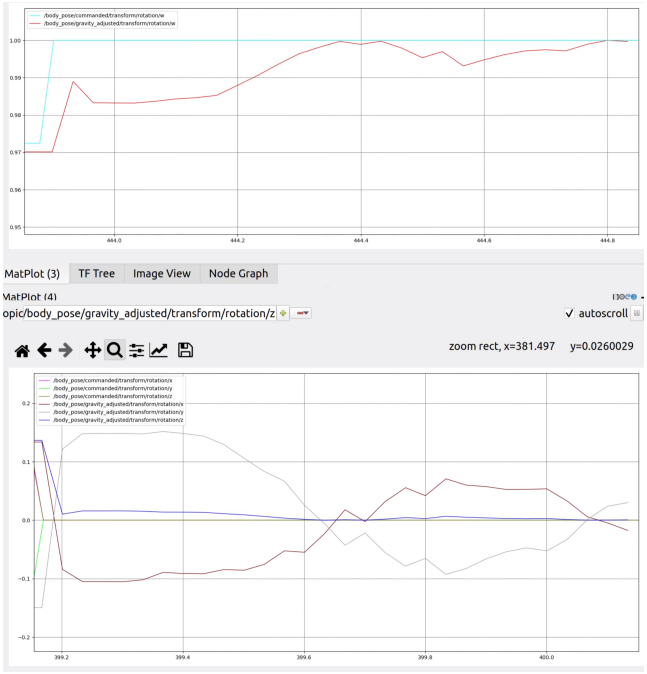

Figure 10: Body pose controller quaternion response to step change

The body pose successfully achieves the desired orientation relative to gravity, despite the noisy sensor data. Even though the system as a whole has a high degree of dynamic complexity, the approach to commanding the body pose results in this controller behaving as expected when tuning a classical PID controller. An example system response can be seen in Figure 10. Notice the overshooting and slight underdamped behavior.

5.3 Motion Planning

After tuning the Navigation Stack parameters, the robot was able to successfully navigate from one pose to another in the global map in emulation, as can be seen in Figure 11. This was done using simple modeled IMU data to feed the EKF, and the system was run assuming that the odometry solution from the EKF is truth. No range sensor data is published, because obstacles were not modeled to return emulated sensor data.

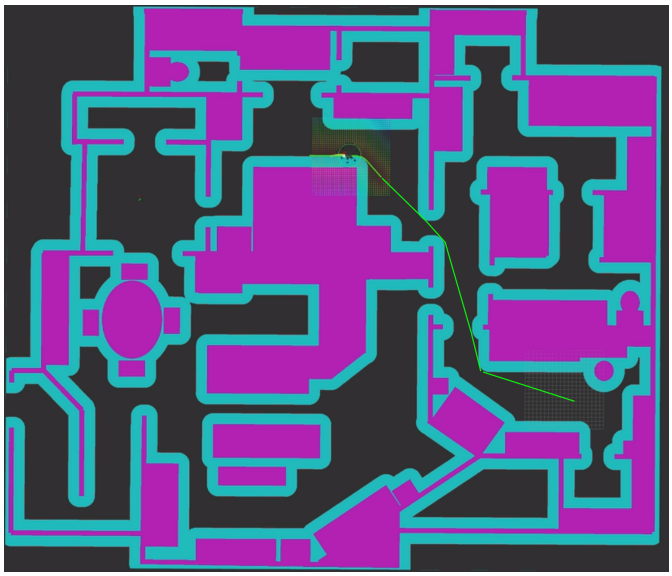

Figure 11: Successful path following with perfect odometry in Rviz

The robot was less successful when implementing this approach on hardware. While this system seemed to work adequately for short distances and time scales, the EKF solution eventually drifts from the true pose of the robot, which occasionally results in velocity commands being sent to the robot that drive it into an obstacle rather than following the desired path.

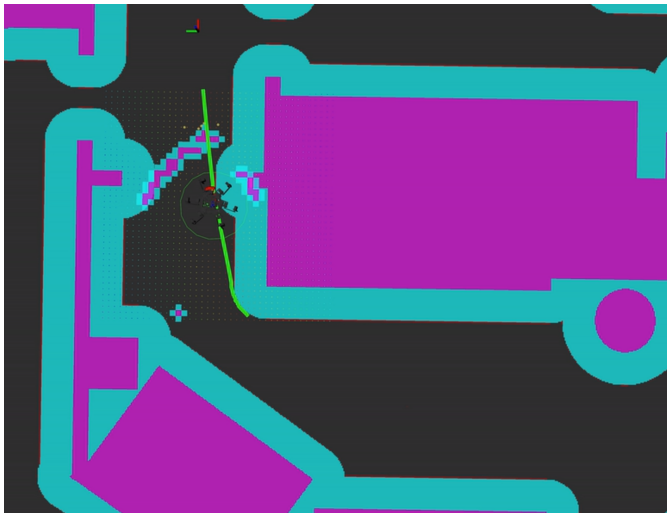

Figure 12: Unsuccessful path following on hardware, with false clutter detections obstructing the path

This drifting is in part due to the noisy IMU sensor. While this sensor was fed into the EKF, its high noise content and discontinuities while walking, as seen in Figures 13 and 14, required its covariance matrix to be large relative to the commanded velocity, or the EKF solution would drift randomly. This prevented the IMU from being able to meaningfully compensate for foot slippage, allowing the filter to drift from truth.

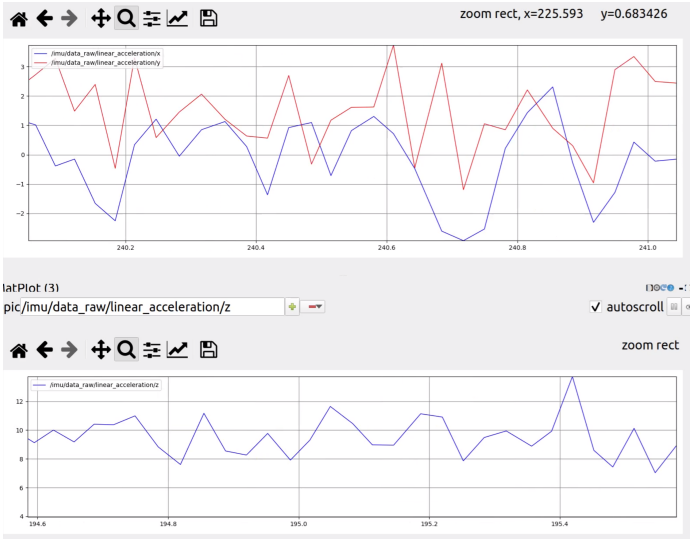

Figure 13: Example IMU accelerometer measurements during walking

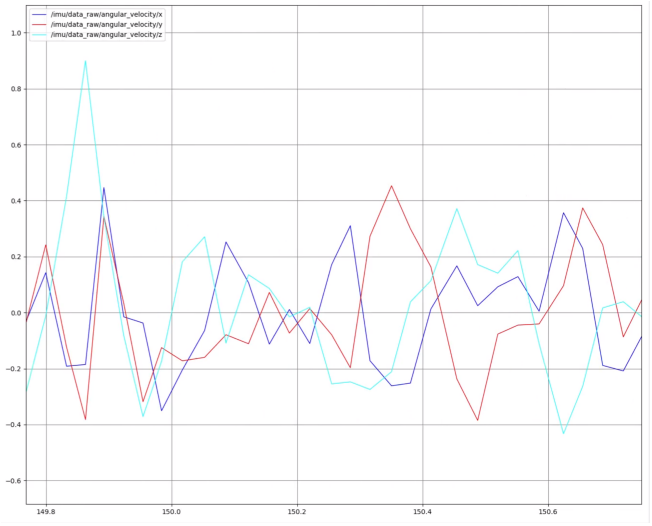

Figure 14: Example IMU gyrometer measurements during walking

The range sensor also proved to often hinder successful path following rather than aid it. The simple coherance filter occasionally prevented real obstacles from populating on the map, allowing the robot to walk into obstacles undetected. However, the filter was not always sufficient to prevent clutter from being passed to the navigation stack. The robot would often detect environment effects and report them as obstacles despite there being room for the robot to walk. This would cause an artificial obstacle barrier to be set in the local costmap, preventing the robot from navigating to the desired goal as seen in Figure 12.

6 Discussion

Analytically constraining the body pose was a safety-critical prerequisite to implementing a classical body pose controller on the robot. The approach presented allowed the body pose and feet to maintian their maximum workspace values while synergistically updated together. After implementing this system, inverse kinematic errors were not observed. This safety check provided confidence in implementing the body pose PID controller, allowing the controller to move the pose as necessary without exceeding kinematic limitations.

Global motion planning was less successful on this system. While the path planning worked effectively and was demonstrated in Rviz, this assumed that the odometry solution returned the truth. In reality, the EKF often drifted significantly from the true pose of the robot. This is largely due to foot slippage when walking. The robot was tested on carpet, and the small footprint of each leg did not provide enough traction to mantain solid contact consistently. The noisy IMU was also to blame, since it was not reliable enough to help correct the EKF much when this drifting occured.

Measured external clutter was another factor in preventing the physical robot from consistently reaching its

commanded goals. Despite filtering the range sensor observables, the robot often returned obstacles that did not appear to be physically present. This often took the form of ground clutter. While efforts were taken to prevent ground clutter return (pointing the seeker up, and checking if the edge of the sensor beam width collides with the ground plane at that range), this needs further refinement. This often resulted in the robot detecting a barrier of obstacles blocking the path that was not present in reality, and the robot could not create a solution to navigate to the goal. While an improved filter for this sensor would likely improve performance, this was beyond the scope of this project.

Ultimately, better sensor hardware and filtering is needed to reliably implement global motion planning and following on this type of hexapod robot. Absolute position sensors like GPS would be useful in localizing the robot, although that may not be useful given the indoor setting and the small scale of motion. Higher fidelity range sensors like a lidar could instead be used to implement a SLAM algorithm like *amcl* [11]. The embedded camera could also be used to contribute to localization, or at least obstacle detection, but this was also beyond the scope of this project.

The hardware design of the feet also contributed somewhat to this poor localization. The leg links are made of a slick, laser cut acrylic. The tibia narrows to a fine point, providing minimal foot contact with the ground. When walking across carpet, this small footprint does not provide enough traction to prevent slippage. A larger footprint with a higher friction coefficient (perhapse using something like a rubber coating) could help minimize this slippage that contributes to EKF drifting. While this would be benificial, dead reckoning odometry drifting can ultimately never be completely eliminated without global localization techniques, and better sensing would still be necessary for the best solution.

References

[1] Raul Mur-Artal et. al. "ORB-SLAM: a Versatile and Accurate Monocular SLAM System". In: *IEEE Transactions on Robotics* (2015). URL: https://arxiv.org/abs/1502.00956.

[2] et al. Bingyi Xia. "Motion Planning for Hexapod Robots in Dynamic Rough Terrain Environments". In: 2021.

[3] Faigl Cizek Masri. "Foothold Placement Planning with a Hexapod Crawling Robot". In: IEEE/RSJ International Conference on Intelligent Robots and Systems (Vancouver, BC, Canada). 2017.

[4] "Freenove Big Hexapod Robot Kit for Raspberry Pi". https://github.com/Freenove/Freenove_Big_Hexapod_Robot_Kit_for_Raspberry_Pi. 2023.

[5] Gus. "Raspberry Pi Distance Sensor using the HC-SR04". 2018. URL: https://pimylifeup.com/raspberry-pi-distance-sensor/.

[6] M. Agheli Hajiabadi. "Analytical Workspace, Kinematics, and Foot Force Based Stability of Hexapod Walking Robots". In: 2013.

[7] "imu_complementary_filter". http://wiki.ros.org/imu_complementary_filter. 2023.

[8] D.E. Koditschek L. Pei-Chun H. Komsuoglu. "A leg configuration measurement system for full-body pose estimates in a hexapod robot". In: *IEEE Transactions on Robotics*. Vol. 21. 2005, pp. 411–422.

[9] Zhao Chai Gao Qi. "Obstacle avoidance and motion planning scheme for a hexapod robot Octopus-III". In: *Robotics and Autonomous Systems* (2018).

[10] "robot_localization". https://github.com/cra-ros-pkg/robot_localization/tree/noetic-devel. 2022.

[11] "ROS amcl". http://wiki.ros.org/amcl. 2023.

[12] "ROS Map Server". http://wiki.ros.org/map_server. 2023.

[13] "ROS Move Base". http://wiki.ros.org/move_base. 2023.

[14] "ROS Navigation". http://wiki.ros.org/navigation. 2023.

[15] "ROS Noetic Ninjemys". 2022. URL: http://wiki.ros.org/noetic.

[16] "ROS rviz". http://wiki.ros.org/rviz. 2023.

[17] A Sanjeev. "How to Interface Arduino and the MPU 6050 Sensor". 2018. URL: https://maker.pro/arduino/tutorial/how-to-interface-arduino-and-the-mpu-6050-sensor.

[18] "tf2". 2023. URL: http://wiki.ros.org/tf2.

[19] Zhang Xia Zhang. "A New Foot Trajectory Planning Method for Legged Robots and Its Application in Hexapod Robots". In: *Applied Sciences* (2021).