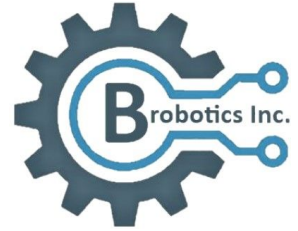


Letter of Transmittal



Mr. James Johnson
The University of Texas at San Antonio
One UTSA Circle, San Antonio, TX 78249

SUBJECT: Operations Manual for Brobotics Inc. "Lil'Bro: An inexpensive open-source quadrupedal robot built for the Robotics and Motion Laboratory". College of Engineering, The University of Texas at San Antonio.

Dear Mr. Johnson,

Enclosed is the operations manual for Team 31, Brobotics Inc., this will include a list of the setup and operation of Lil'Bro, the theory on how the unit functions, and the procedure necessary for troubleshooting. For any questions or concerns, please contact the Team Lead of Brobotics Inc. Steven Farra at zvo618@my.utsa.edu.

Sincerely,

A handwritten signature in black ink, appearing to read 'Steven Farra', is positioned above a horizontal line.

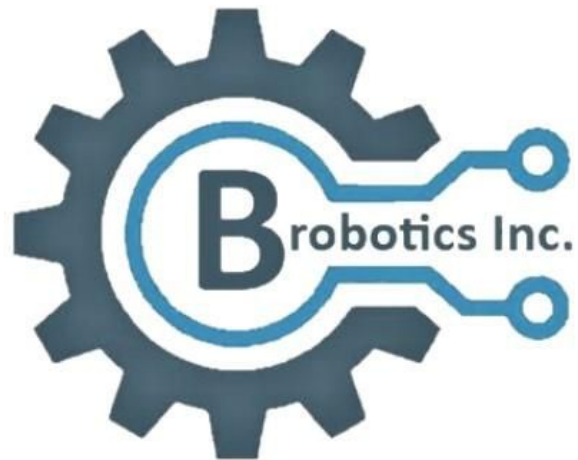
Steven Farra
Brobotics Inc. Team Lead

A handwritten signature in blue ink, appearing to read 'Pranav Bhounsule', is positioned above a horizontal line.

Pranav Bhounsule
Project Sponsor



LIL'BRO OPERATION MANUAL



Prepared by:

Steven Farra, Emiliano Rodriguez, John Carroll, Mario Navarro

Prepared for:

ME 4813 Senior Design II

TABLE OF CONTENTS

1. Product Description	6
1.1 Product Background	6
1.2 Document Scope	6
1.3 Product Overview	7
2. Theory of Operations	9
2.1 Leg Mechanism and Locomotion	9
2.2 Electrical System	13
2.3 Software	18
3. Operating and Setup Instructions	23
3.1 Setup Instructions	23
3.2 Charging Instructions	28
4. Troubleshooting	30
4.1 General Troubleshooting	30
4.2 Motor Calibration Offset Configuration	32
5. References and Contact Information	39
5.1 References	39
5.2 Contact Information	39

LIST OF FIGURES

- Figure 1.1.** Lil'Bro Complete Assembly.
- Figure 1.2.** Right and Left leg sub-assemblies.
- Figure 1.3.** Body sub-assembly (External, and internal views).
- Figure 2.1.** Parts included in a leg sub-assembly.
- Figure 2.2.** Mechanical operation of leg sections.
- Figure 2.3.** Free body diagram of leg.
- Figure 2.4.** Circuit Diagram of Electrical Components.
- Figure 2.5.** Battery, power switch, voltage regulator and ODrive board wiring.
- Figure 2.6.** ODrive to leg assembly connections.
- Figure 2.7.** Raspberry Pi and IMU connection.
- Figure 2.8.** Block diagram of the main thread.

Figure 2.9. Block diagram of the input controller thread.

Figure 3.1. Starting position of the robot

Figure 3.2. Pushbutton power switch.

Figure 3.3. Starting position of the robot.

Figure 3.4. Initial standing position of the robot.

Figure 3.5. Forward walking of robot with controller input shown.

Figure 3.6. Backwards walking of robot with controller input shown.

Figure 3.7. Battery pack.

Figure 3.8. HiTEC Multi Charger.

Figure 4.1. Pre-calibration starting position.

Figure 4.2. Correct shaft position of the outer motor after calibration.

Figure 4.3. New Terminal Window.

Figure 4.4. Odrive tool interface.

Figure 4.5. Odrive tool interface calibration sequence.

Figure 4.6. Left picture: wrong position of motor shaft after calibration, Right picture: correct position after manually moving motor.

Figure 4.7. Odrive tool interface assigning new starting position.

Figure 4.8. Odrive tool interface saving configuration.

Figure 4.9. Odrive tool interface rebooting.

Figure 4.10. Odrive tool interface recalibration.

Figure 4.11. Correct shaft position of the outer motor after calibration.

LIST OF TABLES

Table 1.1. Estimated profile dimensions and mass of unit.

Table 2.1. The Component's functionality in Lil'Bro.

Table 4.1. Solutions to possible errors in operation of Lil'Bro.

Nomenclature

Symbols/Acronyms	Definition
l_1	The upper leg link length
l_2	The lower leg link length
l	The length of the fifth imaginary leg link
θ_1	The angle between the horizontal and rightmost upper leg link
θ_2	The angle between the horizontal and the leftmost upper leg link
m	Meter in length, width and height
kg	Kilogram
$N \times m$	Newton meter
AWG	American Wire Gauge
$GPIO$	General Purpose Input/Output,
IMU	Inertial Measurement Unit
SCL	The serial clock line
SDA	The serial data line
USB	Universal serial bus
$HDMI$	High-Definition Multimedia Interface

1. Product Description

1.1 Product Background

The objective of this project is to design, build, and test an open source quadrupedal, or four-legged, robot for UTSA's Robotics and Motion Laboratory (RAM Lab). The robot, otherwise known as Lil'Bro, will be used by researchers at the RAM Lab for ongoing agile gait locomotion research. The manufacturer of the robot currently used in the lab does not provide the robot's source code with the purchase of the robot. The robot is also expensive and hard to maintain, which is why the Brobotics Inc. team aims to minimize the cost of and maximize accessibility to the robot's design. An online Github repository for the project will be created to allow current and future users to easily access the robot's source code. The team intends to further increase accessibility to users by designing the fabricated parts of the robot to be 3D printed, which has not been done by any other providers of similar-sized robots.

1.2 Document Scope

This document was prepared based on guidelines set forth by course ME 4813 Senior Design II at the University of Texas at San Antonio. For any questions regarding the subject material in this document, refer to Section 5.2 for contact information for the Brobotics Inc. team.

1.3 Product Overview

The overall design of Lil'Bro can be seen in Figure 1.1. below.

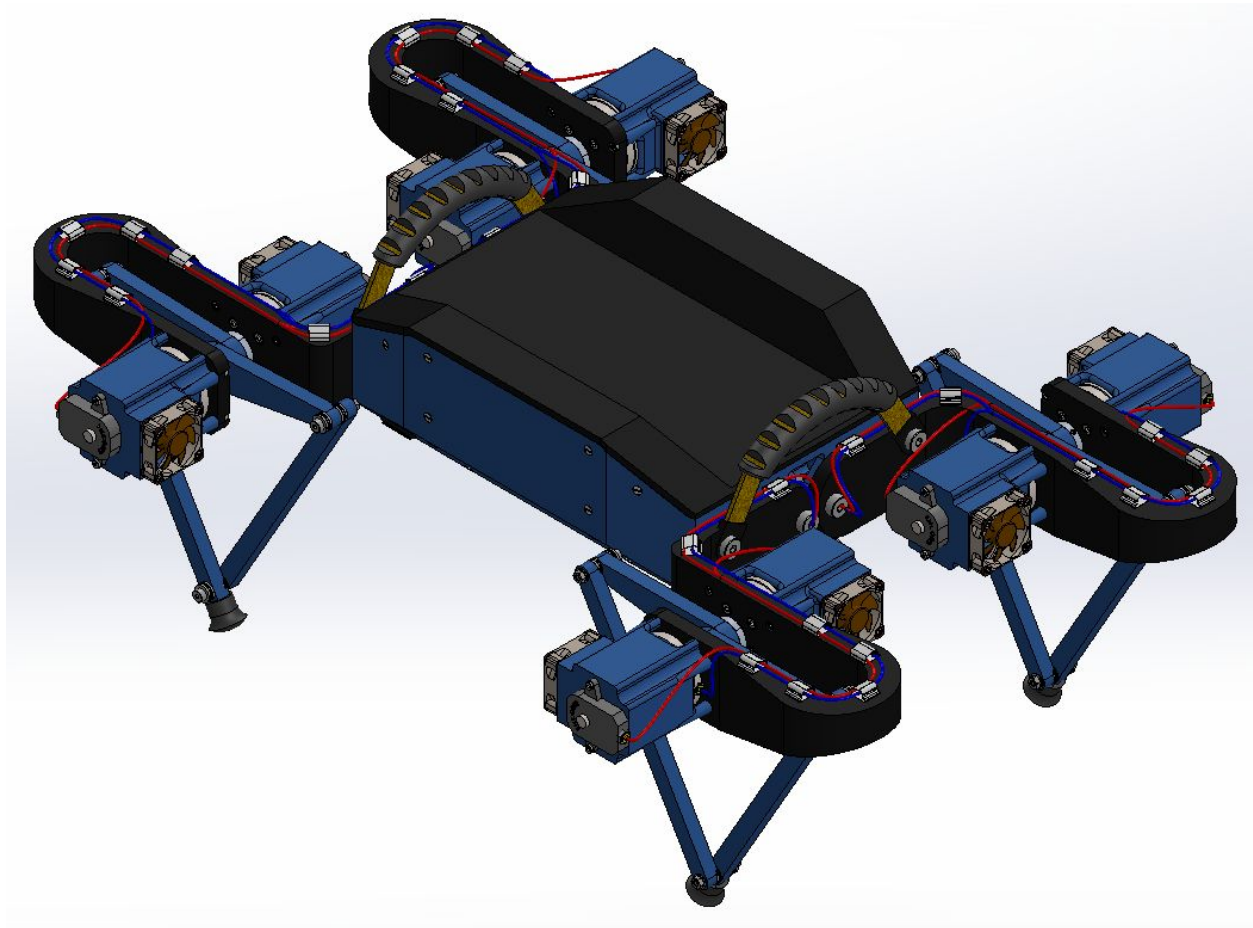


Figure 1.1.. Lil'Bro Complete Assembly.

Lil'Bro will can walk via user interface from a remote controller. The unit's estimated dimensions and mass are shown below in Table 1.1 below.

Table 1.1. Estimated profile dimensions and mass of unit.

Length	0.69 m
Width	0.57 m
Height	0.22 ± 0.08 m
Mass	9.5 kg

There are a total of five sub-assemblies, two left leg sub-assemblies, two right leg sub-assemblies, and a body sub-assembly. These assemblies can be seen in the figures below.

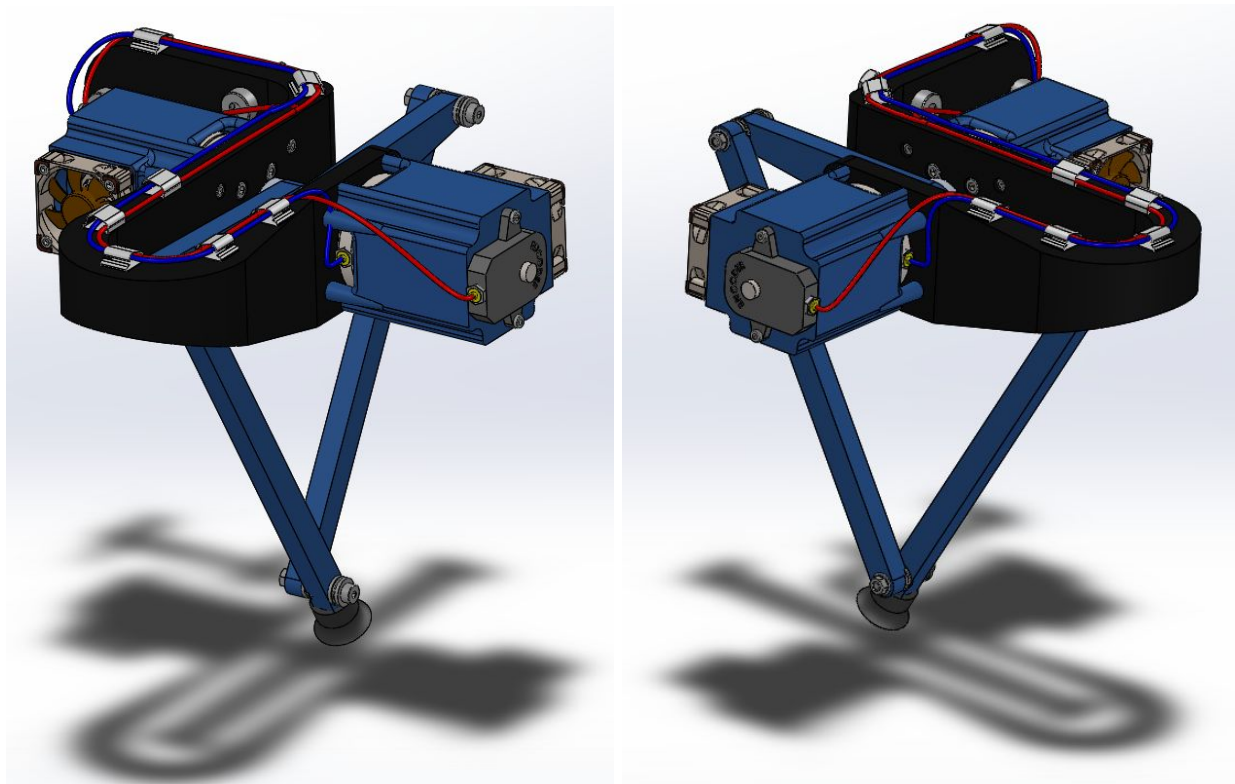


Figure 1.2. Right and Left leg sub-assemblies.

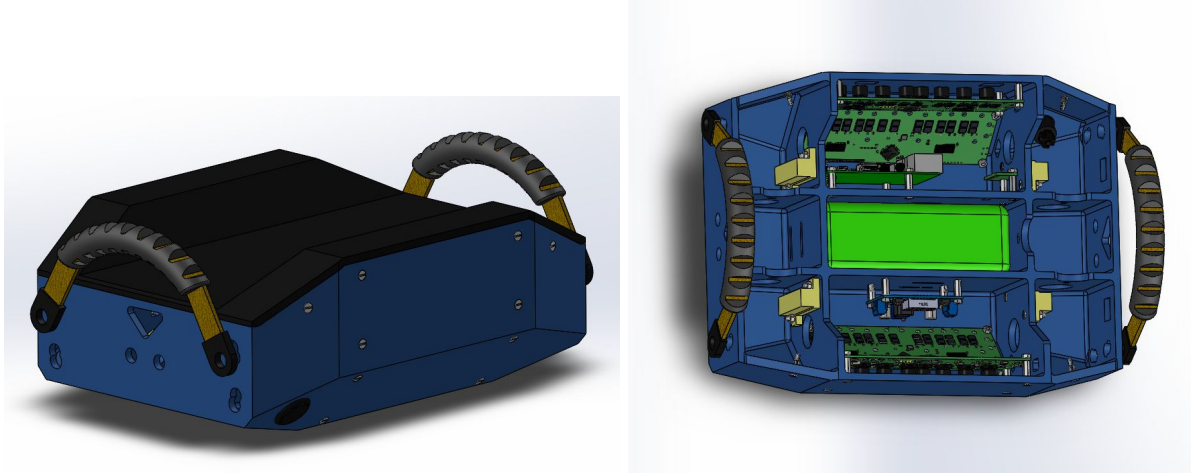


Figure 1.3. Body sub-assembly (External, and internal views).

The four leg sub-assemblies will attach to the body sub-assembly, which will house the necessary components to control and power the legs, these components are listed in section 2.2 of this manual. The leg sub-assemblies are designed for easy removal from the body sub-assembly, this allows for easy assembly of each leg while isolated from the main assembly. The leg sub-assemblies are connected to the body with three nylon shoulder screws, these screws are also used to attach two carrying handles to the assembly, for lifting and transporting the unit.

2. Theory of Operations

2.1 Leg Mechanism and Locomotion

Lil'Bro is propelled by four leg sub-assemblies attached to a body sub-assembly. Each leg is powered independently, and the position and motion relative to the other legs will be controlled via user interface.

Each leg is powered by two motors, that can provide a potential 2.14 Newton meters of torque each. These motors will be attached to the units body using a fabricated motor mount, seen in Figure 2.1 below.

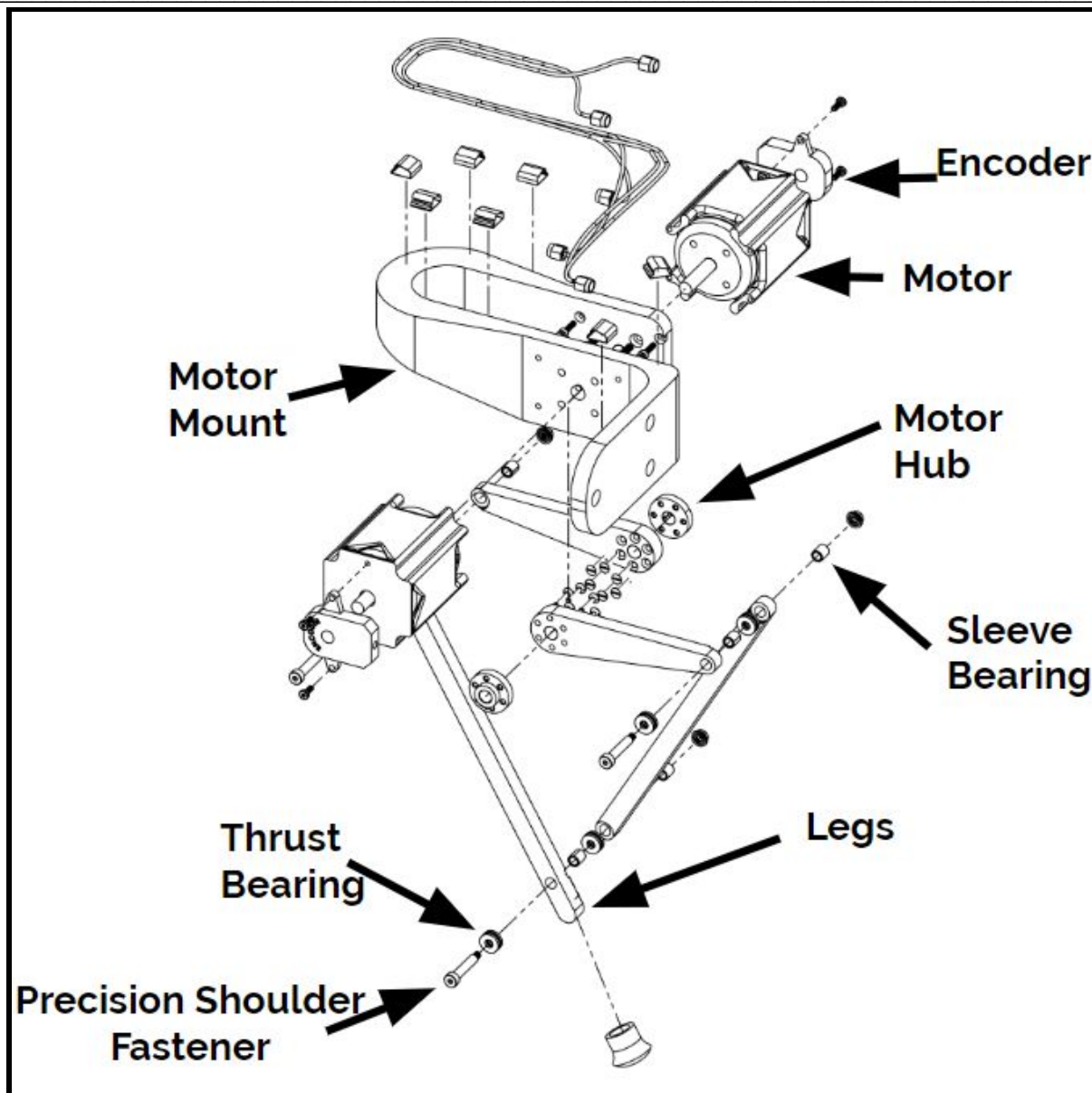


Figure 2.1. Parts included in a leg sub-assembly.

The shafts from the motors will be used to support the legs directly, this is done by using a hub to connect the upper leg sections to each motor shaft. The hubs will also transfer the torque from the motors rotors to the upper leg sections. The upper leg sections of the leg sub-assembly are connected to the lower leg sections using a shoulder screw that runs through sleeve bearings that are pressed into the leg sections; the lower leg sections will be connected using the shoulder screw as well. The shoulder screws will allow the legs to rotate freely at their locations, seen in

Figure 2.2. To securely fasten the leg sections to each other without impeding the free rotation, thrust bearings are placed on the shoulder screw between the leg sections. Each leg is comprised of four leg sections, two upper leg section and two lower leg sections. One lower leg section is longer than the other, this will provide a surface area free of moving parts that will act as a foot for the unit. The upper and lower sections attached to each motor are offset from those on the other motor, this allows for a large range of motion without interference. The design of the leg will theoretically allow for the fabrication of leg sections that are low mass, and will not require designated actuators for the knee joints to control the position of the foot.

The position of the foot relative to the fixed position of the motors will be controlled by applying torque from the motors to the upper leg sections, as seen in the figure below.

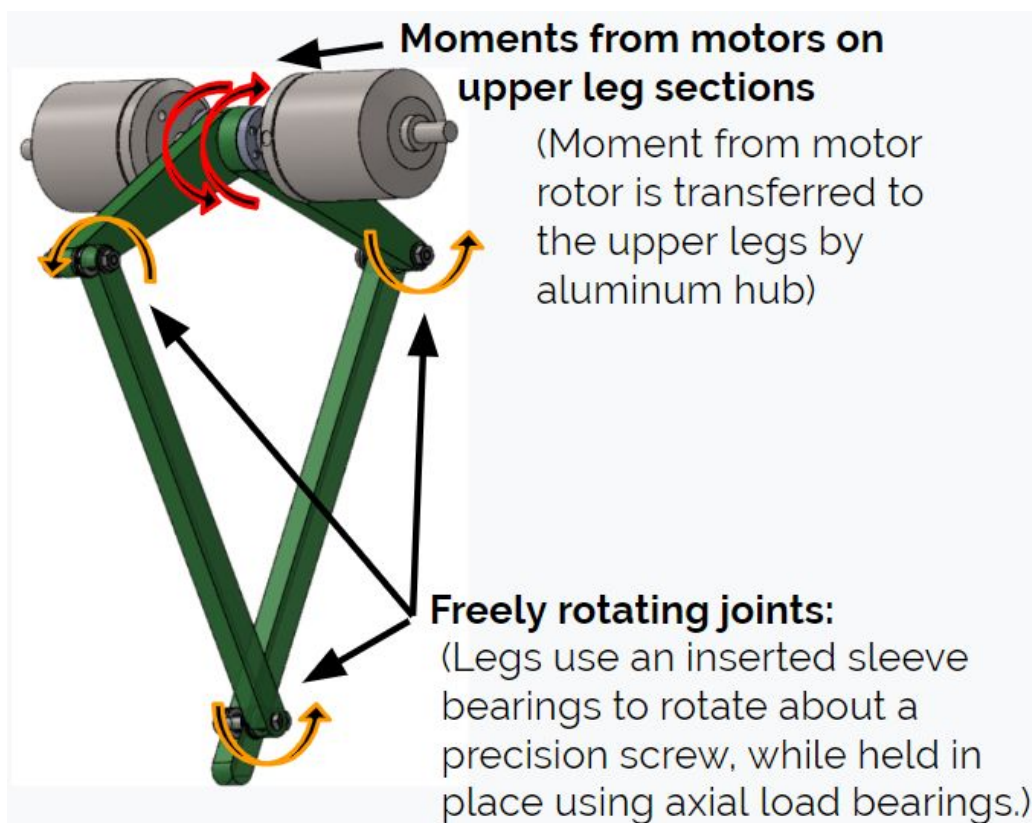


Figure 2.2. Mechanical operation of leg sections.

Controlling the angle of each upper leg section relative to a fixed datum will theoretically allow for the accurate placement of the foot. When the unit is in the standing position and not walking, the weight of the unit will create a moment force on the motor rotors, this moment will be matched by an equivalent torque output from the motors. This will theoretically allow the unit to stand at a fixed height and position.

The unit will walk by controlling the angle of each upper leg section relative to a fixed datum. These angles can be seen in the figure below.

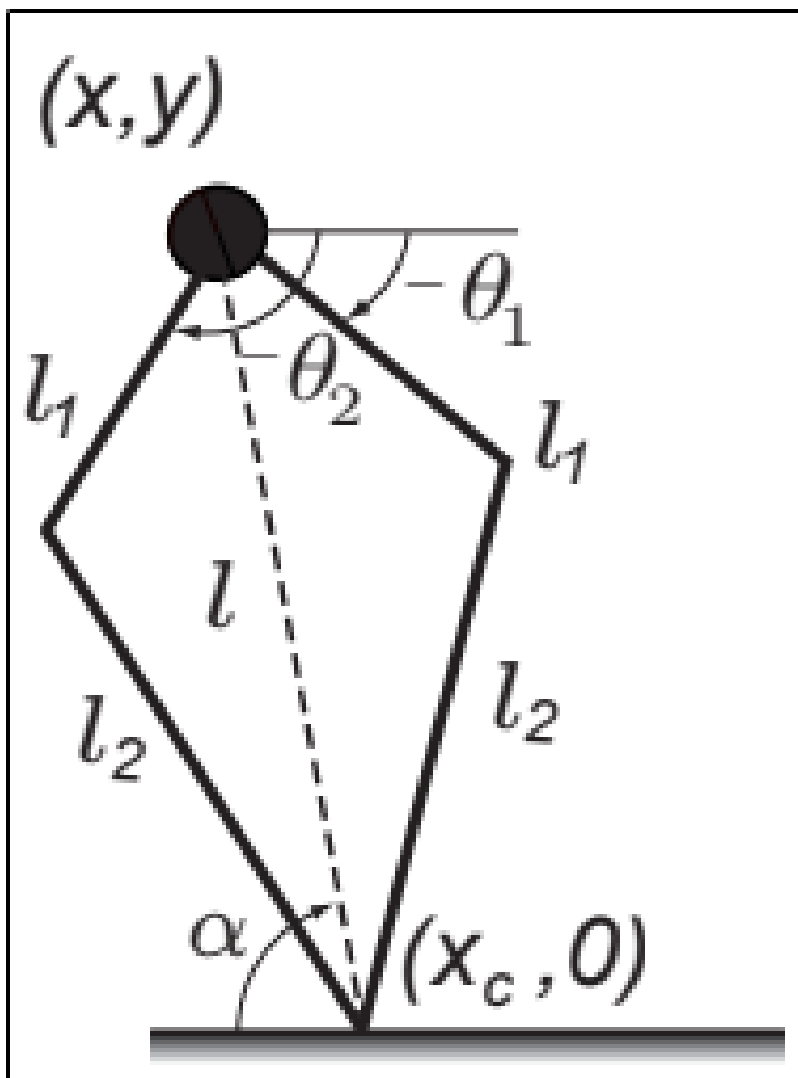


Figure 2.3. Free body diagram of leg.

Using this method of controlling the foot position and speed, the height of a leg can be decreased by increasing the angle of one upper leg section while decreasing the angle of the other. Similarly, the entire leg can be moved forward by decreasing the angle of both upper leg sections simultaneously. These methods of leg motion will be used to make each leg move and theoretically allow the unit to walk.

A paper by Bhounsule, Pusey, and Moussouni^[1] analyzes the symmetric five bar leg configuration, alongside two other configurations, however, they are not of interest for this robot. Figure 2.3 shows a schematic of the symmetric five bar leg configuration.

A walking gait will comprise of three legs propelling the unit in a forward or reverse direction, while the fourth leg elevates and cycles from a predetermined rear position to a predetermined forward position. The relatively lightweight legs will theoretically allow the motors to elevate and transition the legs back to the predetermined forward position in a short period of time, keeping the weight of the unit evenly distributed.

2.2 Electrical System

Lil'Bro is composed of the following electrical components:

Table 2.1. The Component's functionality in Lil'Bro.

Component	Function
4 x ODrive Motor Drivers	<ul style="list-style-type: none">• Provides position, velocity, and current control to two motors.• Receives position and velocity feedback from the encoder attached to the motors• Interfaces with Python
8 x 3-Phase Brushless DC Motors	<ul style="list-style-type: none">• The motors are attached to the upper legs of the robot. The alternating current through the 3 phases of contact produce rotation about the shaft causing the legs to move.• Includes a built-in thermistor which sends an analog signal of motor

	temperature readings to the driver.
4 x Brake Resistors	<ul style="list-style-type: none"> Used for braking energy dissipation
8 x Noctua NF-A4x10mm 5V Fans	<ul style="list-style-type: none"> Will receive a digital input from the Driver once motors reach a certain temperature and will begin to cool the motors down.
8 x Encoders	<ul style="list-style-type: none"> Will provide feedback to the drivers on position and velocity of the motors via analog signals.
1 x 22.2 V Battery	<ul style="list-style-type: none"> Provides 22.2 Volts and 4000mAH, which powers the entire robot.
1 x Power Switch	<ul style="list-style-type: none"> Initializes/Shuts off the robot.
1 x Voltage Regulator	<ul style="list-style-type: none"> Serves as a step down from 22.2 Volts of input to 5 Volts of output.
1 x Raspberry Pi 3 Model B	<ul style="list-style-type: none"> Considered as the brain of Lil'Bro, hosts the main Python script that communicates with the 4 drivers.
1 x Inertial Measurement Unit (IMU)	<ul style="list-style-type: none"> Contains a 3 axis gyroscope and 3 axis accelerometer. For the gyro, the 3 axes are a pitch, a roll, and a yaw. For the accelerometer they are x, y, and z axes. All six axes are analog inputs that are returned to the Raspberry Pi.

As mentioned in the previous section, each leg assembly will include the following components: 2 ODrive motors, 2 Noctua fans, 2 Encoders, 1 brake resistor, and 1 ODrive board. The ODrive boards and brake resistors are not located on the leg itself, but each leg is connected to two motors, a driver, and brake resistor. The ODrive board of each leg assembly will then connect to the Raspberry Pi via USB. Figure 2.4, shown below, is a circuit diagram of the entire system.

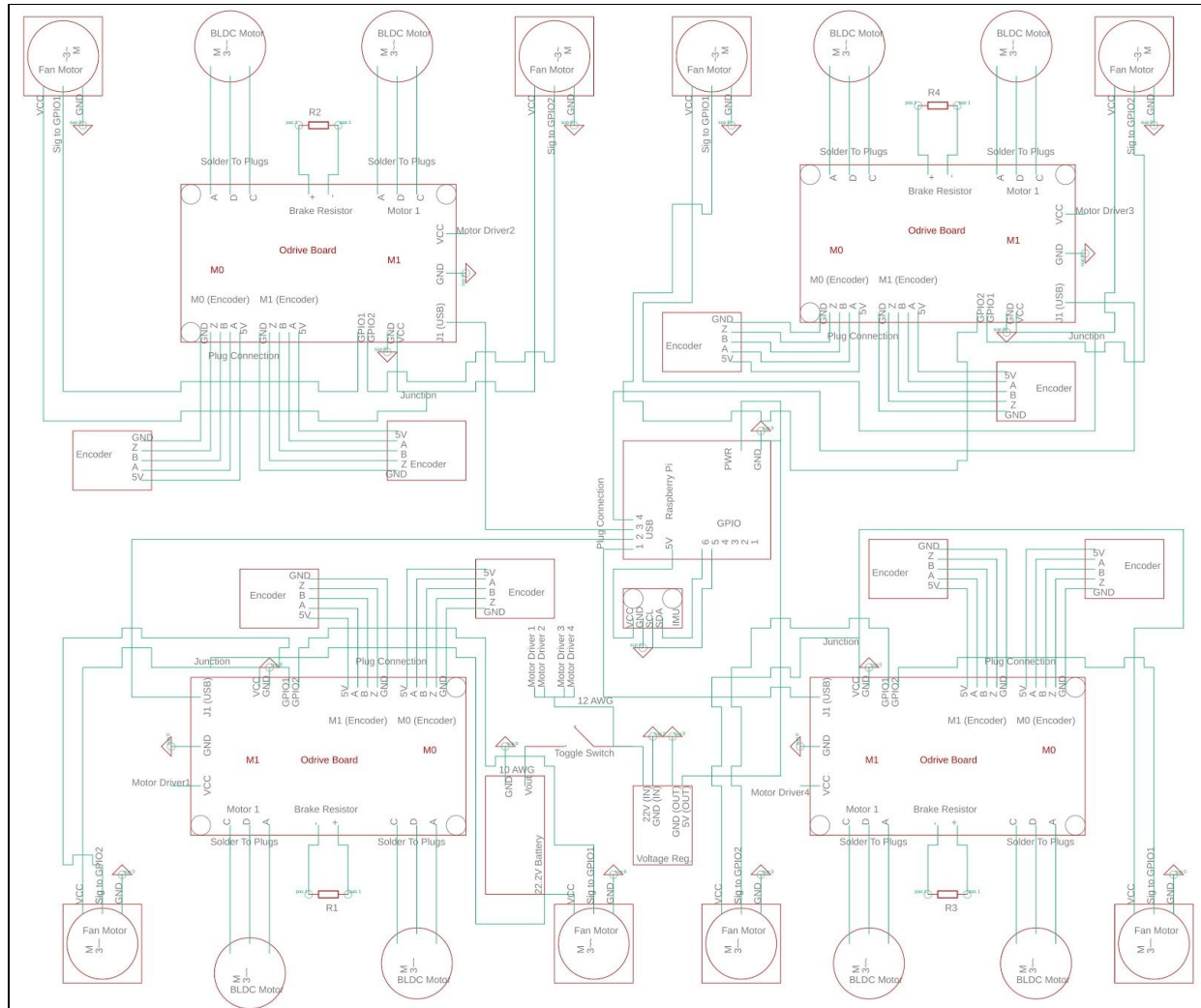


Figure 2.4. Circuit Diagram of Electrical Components.

The following procedure will describe how the wires for one leg assembly will be connected, this process can be repeated for the remaining three leg assemblies. Figure 2.5 shows the primary connections between the battery, power switch, voltage regulator and the 4 ODrive boards.



The battery's two leads connect to the battery's positive and negative terminals, or voltage out and ground. The voltage out will then connect to the power switch. The other end of the power switch will be connected through 12 American Wire Gauge (AWG) wire which splits into two different connections. The first part of the wire provides power to the voltage regulator, and the voltage regulator then provides 5 V to the Raspberry Pi. Each connection is grounded separately. The switch also controls the flow of current to the four drivers by connecting to the 10 AWG wire that carries current from the battery to the four drivers. Once current flows from the power line through the 5 wire junction to each motor driver, it is redistributed by the ODrive board to the leg assembly through the following connections, shown in Figure 2.6.



The ODrives connect to the motors through the plugs labeled ADC which are on the left and right sides of the brake resistor. The three connections represent the three phases of each motors and there is no particular manner in which they should be connected to the motor driver. The brake resistor uses an 18 AWG to connect to the driver. The signal input for the fans will be connected to GPIO pins 1 and 2 using a male to male 20 AWG jumper wires. The power for both fans will come from the connection of the 3.3 volt output of the driver. The encoders will be connected to the driver using the pins VCC, A, B, Z, and GND. This will be accomplished by a male wire of 20 AWG which connects to the female input of the driver. Lastly, the connection between the raspberry pi and the driver is shown below in Figure 2.7.

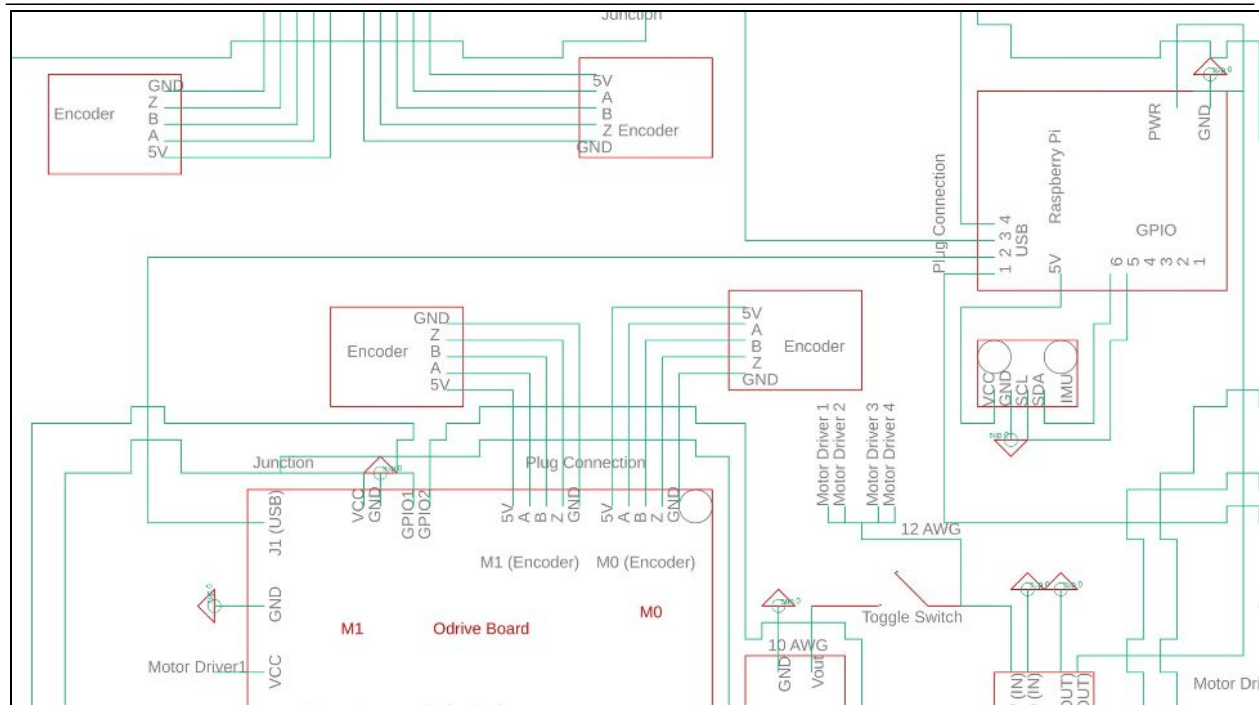


Figure 2.7. Raspberry Pi and IMU connection.

The Raspberry Pi is connected to each driver via USB. This allows the raspberry pi to communicate with the drivers at a higher baud, and still communicate with other components. An inertial measurement unit (IMU) is also connected to the pi. The IMU is supplied 5 volts from and grounded by the Raspberry Pi, and GPIO 5 & 6 will serve as the analog signal inputs from the SCL and SDA pins to the drivers. The signals will be that of the gyroscope and the accelerometer readings.

2.3 Software

The python programming language is implemented on Lil'Bro for operation; This programming language offers an extensive framework for agile robotic software development. The Raspberry Pi is capable of hosting a Python script for the operation of the robot. Multithreading is a process wherein a program executes two threads simultaneously as opposed to the conventional sequential execution. This allows the threads to share computing and processing resources with

one another, resulting in more efficient use of resources and faster runtime. It is used in the software of Lil'Bro to receive inputs from the handheld controller (controller thread) while executing the walking algorithm (main thread). The operation of the robot can be summarized in the following block diagrams, where Figure 2.8 shows the main thread and Figure 2.9 shows the secondary controller thread.

The host script begins by turning the robot on, and the input and output interface is initialized. This establishes a connection between the Raspberry Pi, ODrive boards, and the Dualshock controller. The script is then split into the two threads, receiving inputs from the controller thread while executing commands in the main thread.

Thread #1

This section of the diagram describes the flow of the operation thread.

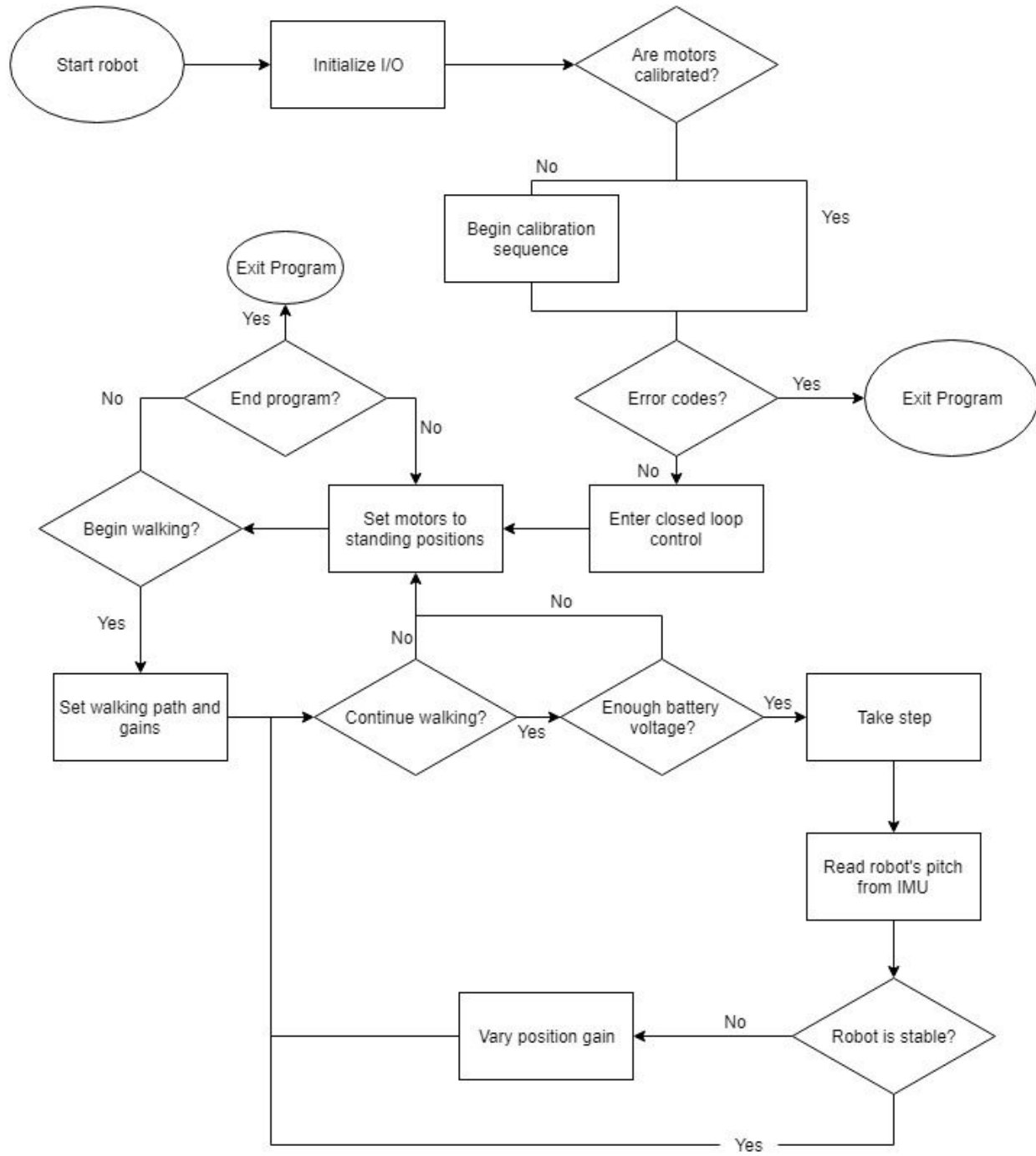


Figure 2.8. Block diagram of the main thread.

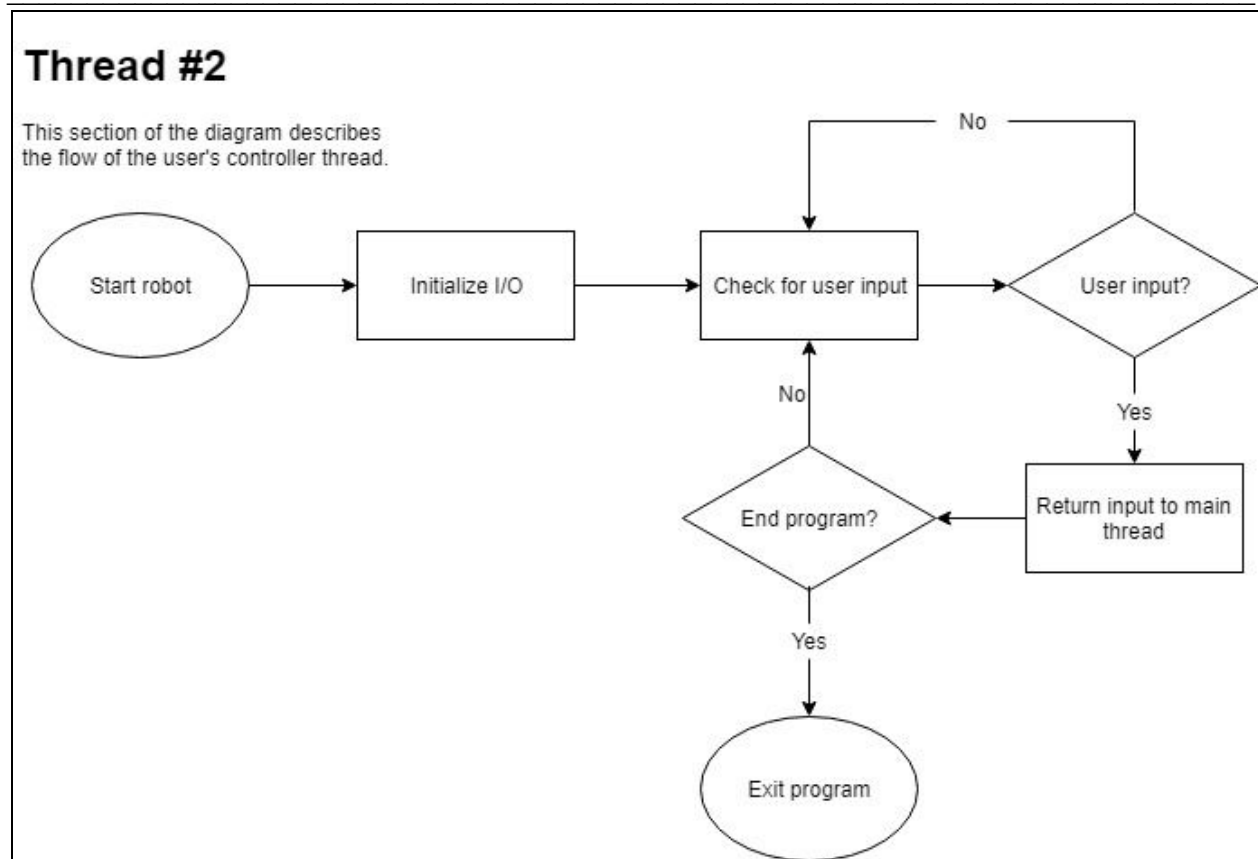


Figure 2.9. Block diagram of the input controller thread.

The main thread executes the calibration sequence once the command is received from the controller thread. For more information regarding the calibration sequence, refer to docs.odriverobotics.com/commands.

Following calibration, the program checks for any error codes that may have been generated. If error codes are not detected, the program enters closed loop control mode when signaled from the user. If an error code is present, the motor(s) that triggered the error no longer respond(s) to commands from the host. The program is terminated for this reason, and Section 4, Troubleshooting, discusses how to clear the present error codes.

Once in closed loop control, all motors are set to operate in position control, which is a closed loop control mode in which motors are set to positions through their encoders. Motors will

attempt to remain at the set position, responding to disturbances by applying current in the opposite direction of the disturbance. Standing is effectively achieved through this control mode because the motors are set to the desired positions which result in the robot standing and independently remaining at those positions. For information regarding the closed loop control mode the ODrive motor drivers use, visit docs.odriverobotics.com/control.

The robot then enters a standby mode, wherein standing is maintained until a user's input is detected. The expected input is for activating the walking algorithm, otherwise, the robot will remain in place. If the input indicates the execution of walking, the walking path is then set according to the position controller's proportional gain. Before taking a step, the program checks if the user has entered an interruption command, which stops the execution of the walking algorithm and returns the motors to standby mode.

If the user wishes to continue with the execution of the walking algorithm, the remaining battery voltage is checked. If it is above the allowable voltage of operation, the walking algorithm executes. If not, the robot returns to standby mode. Instructions on charging the battery are included in Section 3, operating and setup instructions.

With the path and gains set, the robot takes a step. The robot evaluates its stability once it has recovered through its inertial measurement unit (IMU). The IMU's pitch reading is evaluated over the course of landing and the subsequent stabilization. Furthermore, the change in the pitch of the robot from zero after the completion of a step. If the pitch variation is greater than a preset threshold, undesired vibrations may be experienced by the robot that are due to lack of sufficient motor stiffness. This can be attributed to the need for increasing the proportional gain of the position controller. Once each of the four legs takes a step, a single cycle is complete. The robot evaluates the performance of the cycle using the measured pitch, examining the points within the cycle that caused the highest variation in the pitch. The controller(s) of the leg(s) causing the variation at the points of interest is addressed by the script, increasing the proportional gain(s) accordingly.

The script loops until interrupted by the user or by the battery voltage condition. Once interrupted by the user, the robot can be set into walking mode again through the appropriate input. Otherwise, the robot will remain in standby. While in standby, the user is able to place the motor drivers in Idle mode, which removes all torque stiffness from the motors and allows them to freely rotate.

3. Operating and Setup Instructions

3.1 Setup Instructions

The initial setup instructions are shown below and are as follows:

1. Set the robot's body position to the starting (laying down) position as shown in the figure below.

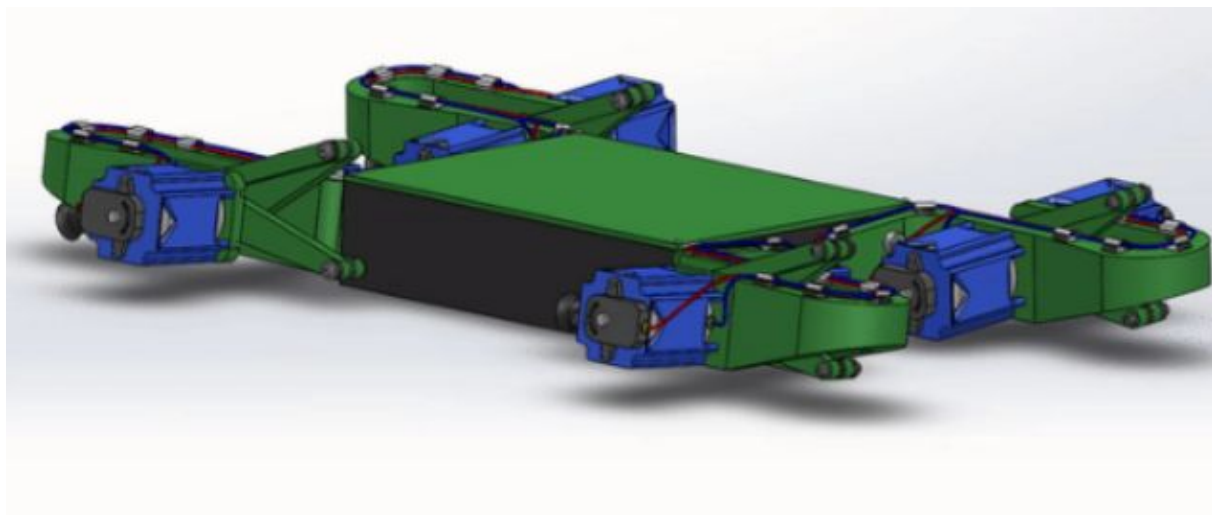


Figure 3.1. Starting position of the robot.

The leg configuration shown above is important for the starting position as the motors need to calibrate and this will provide the correct configuration for calibration (see section 4.2 for calibration instructions).

2. Press the pushbutton switch located on the body to turn on all power to the system.



Figure 3.2. Pushbutton power switch.

Once the button is on (indicated by the LED on button) and the robot boots up (~30 seconds). The user is notified by a green LED on the body, signifying that the system is ready for normal operation and the initial setup process is complete. The user can continue to Operation Instructions below.

Note: if green LED is not displayed on body, the initial setup instructions were not met and the user needs to restart setup instructions, see section 4.1 for general troubleshooting instructions.

After the setup instructions are complete and the system is ready, perform the following steps for the operation of the robot.

The operation instructions are as follows:

1. Make sure the robot is still in the (correct) starting position (laying down) as shown in the figure below. It is important to keep this starting position as motors will calibrate if the system is on for the first time. No user input is needed at this step.



Figure 3.3. Starting position of the robot.

The calibration sequence rotates the motors ONE FULL rotation clockwise and ONE FULL rotation counter clockwise (otherwise check section 4.1 general troubleshooting instructions for encoders). After the calibration sequence is completed (make sure calibration offset is correct for OUTER and INNER motors, see section 4.2 motor calibration offset configuration for verification and further instructions).



Figure 3.4. Initial standing position of the robot.

2. If calibration sequence is correctly done, press the start button on the controller for the robot to begin its initial standing position as shown in the figure above. Successful calibration is indicated by the motors fully rotating in both directions. If calibration was not successful, refer to docs.odriverobotics.com/troubleshooting for help.

This initial standing position is maintained throughout the program if the controller receives no input from the user. This is the base position that the robot will continue to fall under most of the time in order to maintain balance and readiness for movement execution based upon user input.



Figure 3.5. Forward walking of robot with controller input shown.

3. The robot can walk forward with the use of the controller by simply pressing upward on the left analog joystick. This analog input allows the robot to walk forward at varying speeds; the higher the user pushes the joystick, the faster the robot walks forward.

It is important to note that the robot will execute its forward walking gait when the joystick is pushed upward, but when the joystick is idle (not being pushed) then the robot will always go back to its initial standing position (as shown in step 2). Also, directional walking is possible with the use of the right analog joystick on the controller by shifting it left or right depending on desired turn by the user.



Figure 3.6. Backwards walking of robot with controller input shown.

4. The robot can walk backwards with the use of the controller by simply pressing downward on the left analog joystick. This analog input allows the robot to walk backwards at varying speeds; the lower the user pulls the joystick, the faster the robot walks backwards.

Similar to forward walking, it is important to note that the robot will execute its backward walking gait when the joystick is pulled downward, but when the joystick is idle (not being pushed) then the robot will always go back to its initial standing position (as shown in step 2). Similarly, directional walking is possible with the use of the right analog joystick on the controller by shifting it left or right depending on desired turn by the user.

3.2 Charging Instructions

The battery used for Lil'Bro is the **Turnigy Graphene Panther 4000mAh 6S 75C**.



Figure 3.7. Battery pack.

Note: It is important that the battery does **NOT** contain any defects. Inspect battery anytime before operating the robot and confirm that the battery is charged more than 50% of its capacity in order to prevent damage to the robot.

Charging the battery: The **HiTEC multi charger X1 Touch** is the recommended charger to use. But other conventional chargers can also be used provided that it can support the **Turnigy Graphene Panther 4000mAh 6S 75C**.



Figure 3.8. HiTEC Multi Charger.

For charging instructions, read the manual provided in the following link:

https://hitecrd.com/files/X1_touch_manual.pdf

4. Troubleshooting

4.1 General Troubleshooting

Table 4.1. Solutions to possible errors in operation of Lil'Bro.

Symptom	Check
Nothing in the robot turns on	<ul style="list-style-type: none">● Check if switch is pressed properly● Check if the battery is plugged in● Inspect battery connection for damage● Charge battery● Test battery
Robot is on, but motors drivers are not	<ul style="list-style-type: none">● Check if the power connectors from battery to the drivers are plugged in● Inspect motor driver connectors for damage● Measure voltage potential to motor drivers● Contact manufacture
Robot is on, but Raspberry Pi is not	<ul style="list-style-type: none">● Check wire harness from battery for loose connections● Inspect the voltage regulator (display should be on)● Measure voltage potential from regulator● If voltage out is correct, troubleshoot Raspberry Pi● If voltage out is incorrect, contact

	regulator manufacture
All components receive power but robot does not respond to commands	<ul style="list-style-type: none"> • Connect to Raspberry Pi and check if script is running • Check USB connections between Raspberry Pi and motor drivers • Check if DS controller is on
Robot runs script but some or all motors don't begin calibration sequence	<ul style="list-style-type: none"> • Check motor drivers for error codes • Check the connections from drivers to motors • Measure current to motors • Check the connections from drivers to encoders • Check if motor hubs are fastened to motor shafts
Motor(s) do not finish calibrating	<ul style="list-style-type: none"> • Check error codes on motor drivers • Inspect encoder connections for damage
Motors calibrate but a position offset is present	<ul style="list-style-type: none"> • Refer to instructions below on how to correct offset in section 4.2
Motors spin but the legs do not move	<ul style="list-style-type: none"> • Check if motor hubs are fastened to motor shafts
Robot stands but legs are not stable	<ul style="list-style-type: none"> • Increase the proportional position gain through DS controller • Check if motor hubs are fully fastened to motor shafts

One of the legs moves slower than the rest	<ul style="list-style-type: none">● Set all drivers to “IDLE” mode and check the ease of movement of all legs by hand● If one leg requires a greater force to move, loosen the fastening tightness of the motors to the mount for that leg
Knee joint(s) of leg(s) does/do not rotate like intended	<ul style="list-style-type: none">● Set all drivers to “IDLE” mode and apply lubricant on bolt and in bearings● Check contact surface between thrust bearings and legs
Robot is too noisy	<ul style="list-style-type: none">● Increase fastening tightness of motor fasteners● Check for leg components rubbing

4.2 Motor Calibration Offset Configuration

The first time you configure a new robot, you must provide the motor calibration offsets; otherwise the positional values of the motors will be inaccurate and operation will fail to execute correctly.

NOTE: It is important to verify that the motor’s position setpoint of 0 (after calibration) are in the correct position (~10 o’clock for outer motors and 12 o’clock for inner motors, see Figures 4.2 and 4.6 below, otherwise continue to follow the calibration offset instructions below.

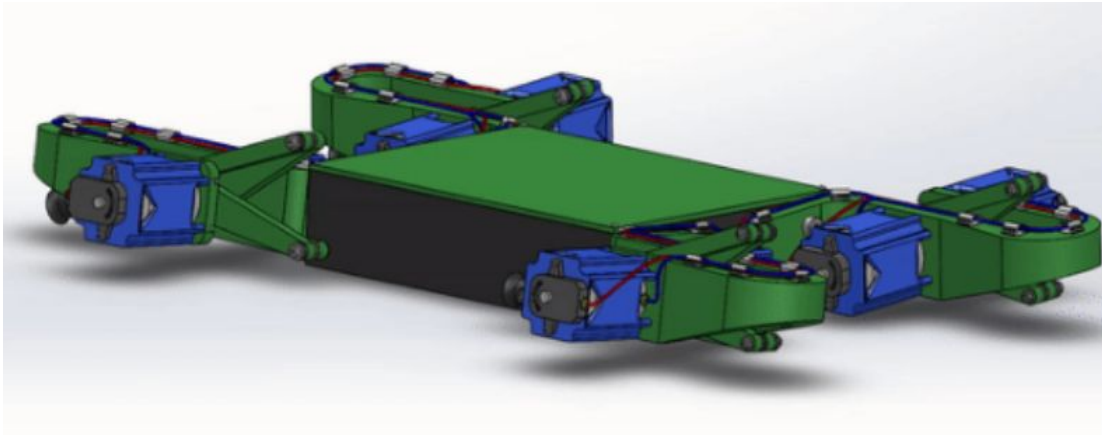


Figure 4.1. Pre-calibration starting position.

The leg configuration (pre-calibration starting position) shown above is important for the calibration sequence, this configuration sets the motor hubs in the correct position for tightening the set screws, and any other configuration will result in failure of operation.



Figure 4.2. Correct shaft position of the outer motor after calibration.

The figure above demonstrates the correct position of the outer motor shaft after calibration. The tick mark indicates the flat side of the motor shaft for tightening the set screws of the motor hub. If this position (~10 O'clock) of the shaft is achieved after calibrating the “OUTER” motors and

(12 O'clock) for "INNER" motors, then there is no need to offset the calibration and can continue with normal operation of the robot (Skip the steps below).

1. Connect to robot interface (via ssh or HDMI) and open up a new terminal window.

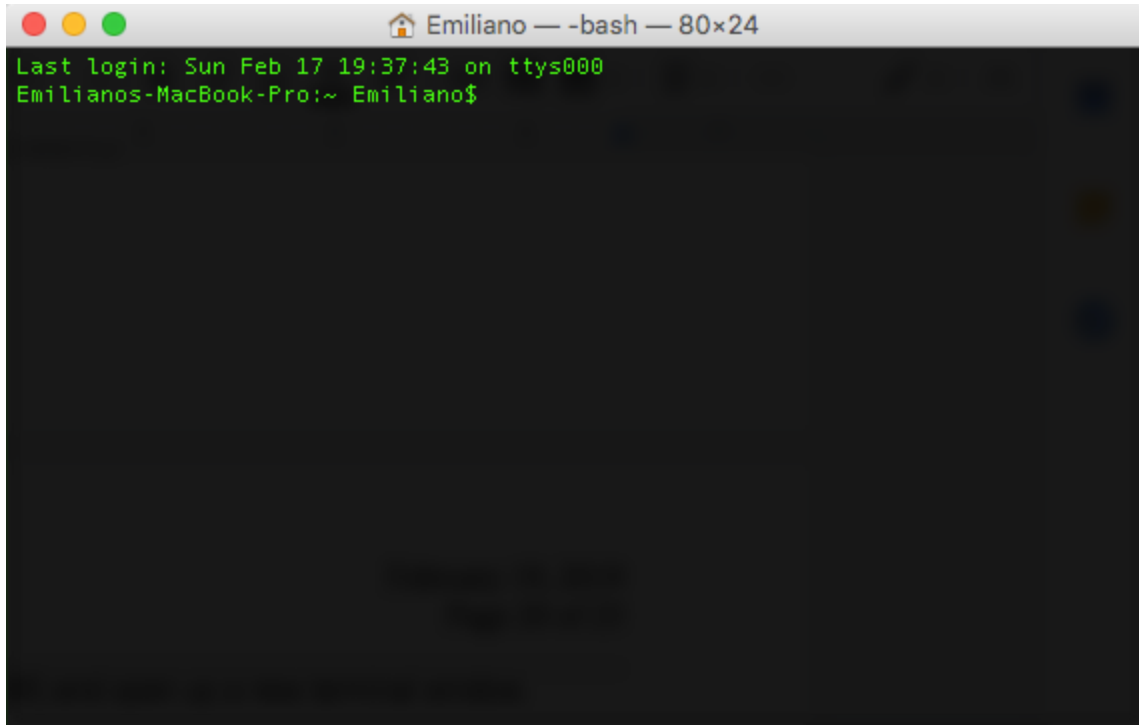


Figure 4.3. New Terminal Window.

2. Make sure odrive motor drivers are connected via USB, type `odrivetool` and press ENTER.

You can read more about odrivetool here:

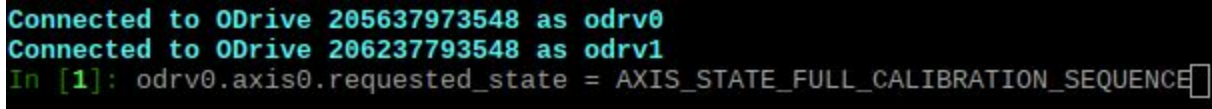
<https://docs.odriverobotics.com/#start-odrivetool>



Figure 4.4. Odrive tool interface.

3. Once the ODrive is connected, type the following command and press ENTER

```
odrv0.axis0.requested_state = AXIS_STATE_FULL_CALIBRATION_SEQUENCE
```



```
Connected to ODrive 205637973548 as odrv0
Connected to ODrive 206237793548 as odrv1
In [1]: odrv0.axis0.requested_state = AXIS_STATE_FULL_CALIBRATION_SEQUENCE
```

Figure 4.5. Odrive tool interface calibration sequence.

NOTE: All the following commands should be typed and executed for all 8 motors.

- *odrvX* is the motor driver name where *X* is an integer value assigned depending on how many boards are connected [0-*n*]. Where *n* = board number.
- *axisX* is the motor name where *X* is an integer value assigned by the connection point on the odrive (can either be 0 or 1).

The command shown above executes the calibration sequence for the motors. If motors are in the correct position after calibration (see figure 4.6), there is no need to perform the following steps and can continue operation as normal.

4. Once full calibration sequence is done and motors are calibrated, manually move the motors (by hand) to correct tick mark position as indicated below.

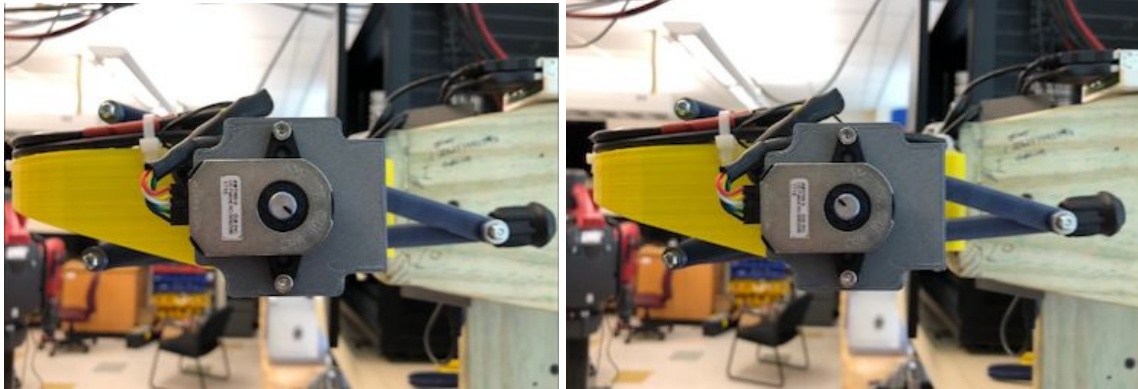


Figure 4.6. Left picture: wrong position of motor shaft after calibration, Right picture: correct position after manually moving motor.

The left figure shown above shows the wrong position after the motor has been calibrated, move the motor (manually by hand) to the correct position shown by the picture on the right above (**Note:** this is for OUTER motors, INNER motor's correct tick position is 12 O'clock).

5. After motor shaft is in the correct position from the previous step. type the following command and press ENTER, this will set the new positional set point of 0.

```
odrv0.axis0.encoder.pos_cpr = 0
```

```
In [5]: odrv1.axis0.encoder.pos_cpr = 0
```

Figure 4.7. Odrive tool interface assigning new starting position.

The command above sets the current position of the motor to zero after calibration. This step is important as this new position will be recognized as zero after calibration of motors.

6. Save the new configuration by typing the following command and pressing ENTER.

```
odrv0.save_configuration()
```


```
In [8]: odrv1.save_configuration()
```

Figure 4.8. Odrive tool interface saving configuration.

The command above saves the motor driver configuration, this stores the new calibrated positions in memory and can be accessed after reboot or shutdown of motors.

7. Reboot the system by typing the following command and pressing ENTER.

```
odrv0.reboot()
```



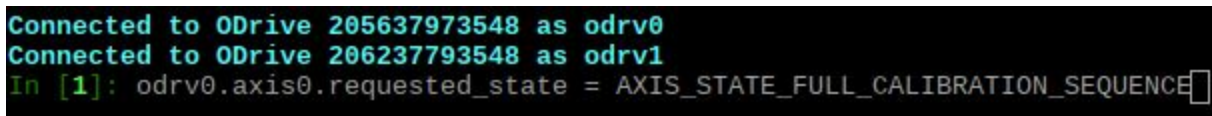
```
In [1]: odrv0.reboot()
```

Figure 4.9. Odrive tool interface rebooting.

Once the command above is executed, the motor driver will disappear from odrivetool and reboot the motor driver. Once rebooted, it will automatically reconnect the driver and recalibration will be necessary in order to use motors again.

8. Once the ODrive restarts and reconnects, re-calibrate the motors by typing the following command and pressing ENTER.

```
odrv0.axis0.requested_state = AXIS_STATE_FULL_CALIBRATION_SEQUENCE
```



```
Connected to ODrive 205637973548 as odrv0
Connected to ODrive 206237793548 as odrv1
In [1]: odrv0.axis0.requested_state = AXIS_STATE_FULL_CALIBRATION_SEQUENCE
```

Figure 4.10. Odrive tool interface recalibration.

9. By visual inspection, make sure the calibration offset is now in the right position (see figureX below, ~10 o'clock for outer and 12 o'clock for inner motors). If correct, tighten the set screws to the motor hubs and continue operation as normal. Otherwise repeat steps 3-9.



Figure 4.11. Correct shaft position of the outer motor after calibration.

5. References and Contact Information

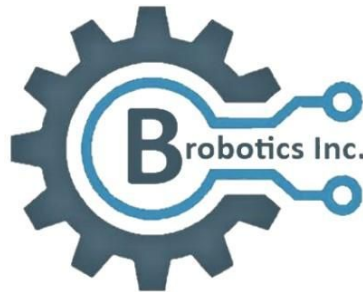
5.1 References

[1] Bhounsule, Pranav A. Pusey, Jason. Moussouni, Chelsea. “A comparative study of leg geometry for energy-efficient locomotion”.

[2] <https://odriverobotics.com/>

[3] https://hitecrd.com/files/X1_touch_manual.pdf

5.2 Contact Information



MEMBERS:

Steven Farra
zvo618@my.utsa.edu
(210) 776-9117

Emiliano Rodriguez
pyn109@my.utsa.edu
(830) 463-1396

John Carroll
kor929@my.utsa.edu
(512) 229-7924

Mario Navarro
nmario597@gmail.com
(210)232-4750