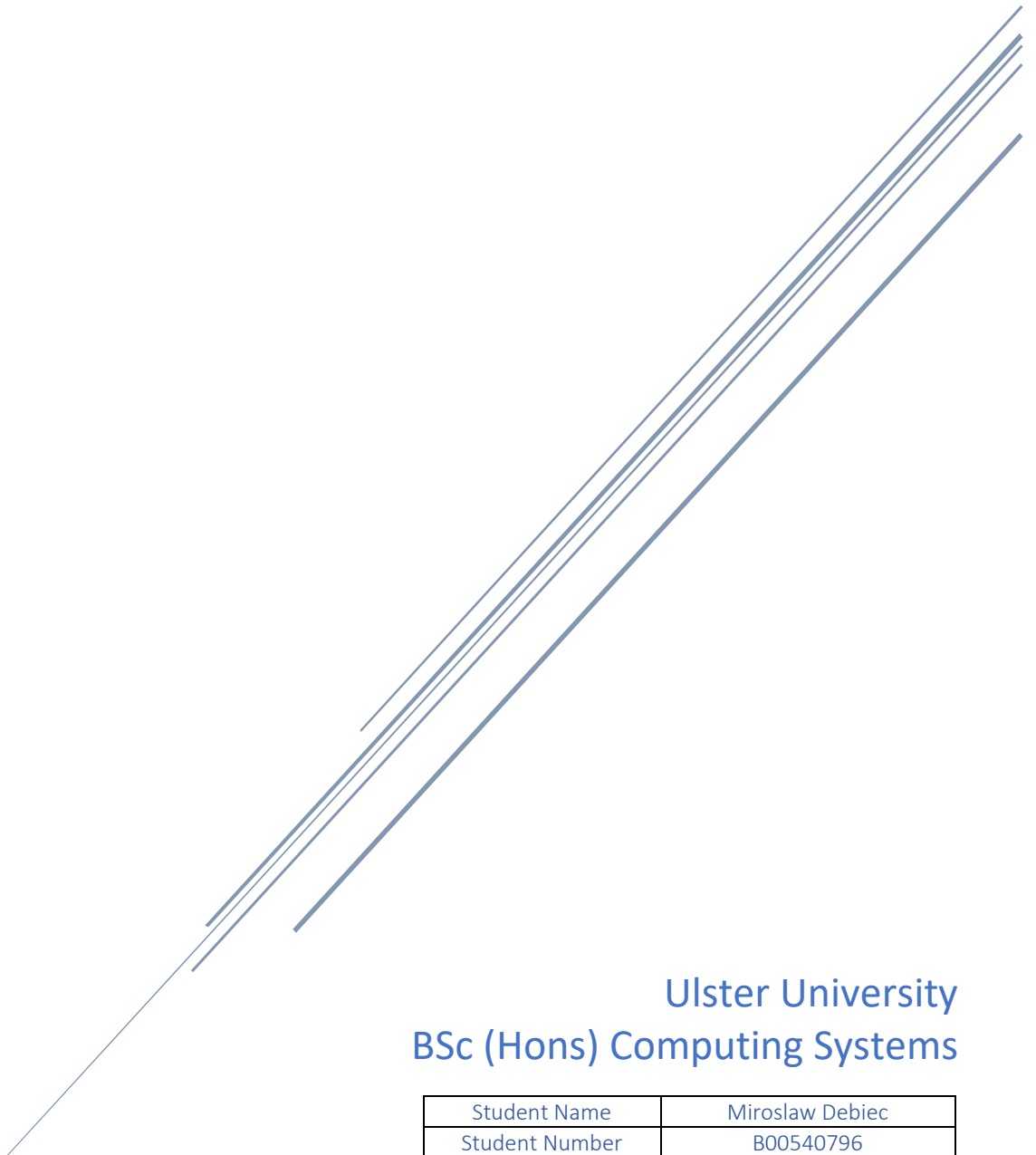


QUADRPUPED ROBOTIC PLATFORM

Final Report – COM667 – Computing System Project



Ulster University
BSc (Hons) Computing Systems

Student Name	Mirosław Debiec
Student Number	B00540796
Supervisor Name	Dr Haiying Wang
Date	April 2021

Abstract

The goal of the project was to create Quadruped Robotic Platform that can be used in connection with single board computers, navigation sensors, computer vision cameras and other external systems allowing for expanding its functionality and provide more capable robotic base that can substituted wheeled or tracked platforms.

The project is hardware intensive and consists of custom designed frame, electronic components such as controller board, power distribution board and battery pack. The robot has built-in Wi-Fi connectivity allowing for accessing and controlling the device from PC or Tablet using only a Web Browser. Device is in self-contained package and does not require any external power or data cables for its operation.

The software component consists of the front-end client UI, back-end firmware, and configuration files. All software elements are stored within flash memory, located on-board of the platform.

The project was developed using modified Agile Software Development Lifecycle including number of prototypes and followed by regular evaluation and testing sessions throughout the project completion.

Taking into consideration level of hardware and software complexity of the project the results are satisfactory as all critical and important requirements were fully or partially achieved.

Acknowledgments

I would like to thank you my project supervisor, Dr Haiying Wang, for his time and guidance through the completion of this project. His feedback and assistance in navigating through the project was invaluable.

Thanks to Dr Don McFall for coordinating sessions with students at the beginning of the final year and for being my course director for majority of my time with university.

Thanks to Dr Mark Donnelly, current course director for providing regular updates and coordinating the course.

Thank you to all lecturers and support staff involved in teaching Computing Systems modules for friendly and kind support during the course.

Content

Abstract.....	0
Acknowledgments.....	2
1. Introduction	9
1.1 Background & Motivation for the Project.....	9
1.2 Project Aim.....	9
1.3 Project Objectives.	10
1.3.1 Hardware Critical:	10
1.3.2 Hardware Desired:	10
1.3.3 Software Critical:.....	10
1.3.4 Software Desired:.....	10
1.4 Outline of Dissertation Structure.....	11
Chapter 1 – Introduction.....	11
Chapter 2 – Literature Review	11
Chapter 3 – Requirements and Risks	11
Chapter 4 – Design.....	11
Chapter 5 – Code Implementation.....	11
Chapter 6 – Project Planning	11
Chapter 7 – Testing and Evaluation	11
Chapter 8 – Conclusion	11
2. Literature Review	12
2.1 Existing Solutions	12
2.1.1 Spot – Boston Dynamics.	12
2.1.2 A1 – Unitree	13
2.1.3 SQ3U – LynxMotion	14
2.1.4 D’Kitty – Trossen Robotics	15
2.2 Products comparison and identifying gaps.....	16
2.3 Value Engineering and Values Analysis.....	17
2.4 Drone industry as example of possible product future.	19
3. Requirements and Risks.....	20
3.1 Functional Requirements.....	20
3.2 Non-Functional Requirements.....	21
3.3 Identified Risks	22
3.4 Risk Evaluation	23
4. Design.....	24

4.1. Hardware	25
4.1.1. Frame	25
4.1.1.1 Core Unit	26
4.1.1.2. Leg Assembly.....	27
4.1.2. Battery Pack	28
4.1.3. Actuators.....	28
4.1.4 Power Distribution Board (PDB)	29
4.1.5 Controller Board.....	30
4.1.5.1. ESP32 DEVKIT v1	31
4.1.5.2. GY9250 (Inertial measurement unit IMU)	31
4.1.5.3. MAX485.....	32
4.2. User Interface	32
4.2.1. Accessing UI	33
4.2.2 General Layout.....	33
4.2.3 Leg Table	34
4.2.4 Actuator Settings	35
4.2.5 Mode Selector.....	35
4.2.6 On-Screen Controllers.....	37
5. Software Implementation	37
5.1. Presentation Layer	38
5.1.1 Index.html	39
5.1.2. Script.js.....	41
5.1.3. Style.css.....	43
5.2 Business Layer	45
5.2.1.1 Main.cpp	45
5.2.2 GaitHandler.h / GaitHandler.cpp	48
5.2.3. InverseKinematics.cpp / InverseKinematics.h (IK)	50
5.2.4 SR518.h / SR518.cpp (Hub)	53
5.3 Data Transfer Objects (DTO)	57
5.3.1 Actuator.h/.cpp.....	57
5.3.2 Step.h/.cpp.....	57
5.4 Persistence Layer	57
5.4.1 JsonHandler.h / JsonHandler.cpp	57
5.4.2 SpiffHandler.cpp / SpiffHandler.h.....	60
5.5 Data Layer	60
5.5.1 Config files.....	61

5.5.2 Gaits	62
Directional gaits - “forward_gait.json”, “back_gait.json”, “left_gait.json” and “right_gait.json”	62
Rotational gaits – “rotateLeft_gait.json” and “rotateRight_gait.json”	62
6. Project Planning	63
7. Testing and Evaluation	65
7.1. Testing	65
7.2 Evaluation of the Requirements	68
8. Conclusion	70
8.1 Complexity	70
8.2. Challenges	71
8.3 Future Improvements	72
References:	73
Appendix 1: Frame Design	74
Appendix 2: SR518 - Control Table	75
Appendix 3: Power Distribution Board (PDB) and Controller Board (CB)	77
Appendix 4: Index.html	78
Appendix 5: Script.js	79
Appendix 6: Style.css	82
Appendix 7: main.cpp	84
Appendix: 8: GaitHandler.cpp / .h	86
Appendix 9: InverseKinematics.cpp / .h (IK)	88
Appendix 10: SR518.cpp / .h (Hub)	89
Appendix 11: Actuator.cpp / .h & Step.cpp / .h	95
Appendix 12: JsonHandler.cpp / .h	97
Appendix 13: SpiffHandler.cpp / .h	98

Tables

Table 1: Spot - Key Characteristics Score	13
Table 2: Key Characteristics Score	14
Table 3: SQ3U - Key Characteristics Score	15
Table 4: D’Kitty - Key Characteristics Score	16
Table 5: Functional Requirements	21
Table 6: Non-Functional Requirements	21
Table 7: Risks.....	22
Table 8: Risk Matrix - Score Representation.....	23
Table 9: Risk Matrix - Risk Assessment without Mitigation Strategy	23
Table 10: Risk Matrix - Risk Assessment with Mitigation Strategy	24
Table 11: SR518 - default specification.....	29
Table 12: Instruction and Status Packet Structure	53
Table 13: Sprints Break-Down.....	64
Table 14: Testing Scenarios.....	68
Table 15: Functional and Non-Functional Requirements Evaluation	69

Figures

Figure 1: Key Characteristics	17
Figure 2: Cost per Capabilities	17
Figure 3: QRP - Design in Principles	24
Figure 4: Frame Design vs Completed Build	25
Figure 5: Core Unit Assembly.....	26
Figure 6: Leg Assembly.....	27
Figure 7: Leg Assembly - side view.....	27
Figure 8: Leg Assembly - top view.....	27
Figure 9: Battery Pack Schematics	28
Figure 10: SR518 Actuator	29
Figure 11: Power Distribution Board	29
Figure 12: Controller Board.....	30
Figure 13: ESP32 DEVKIT v1	31
Figure 14: GY9250 - IMU.....	31

Figure 15: MAX485 IC.....	32
Figure 16: UI – general layout.....	33
Figure 17: Leg Table	34
Figure 18: Configuration Modal.....	35
Figure 19: Model Selector Console	36
Figure 20: Rotational Gait	36
Figure 21: Directional Gait	36
Figure 22: Direction Controller while engaged	37
Figure 23: QPR with adjusted pitch and roll values	37
Figure 24: QPR – Software Architecture	38
Figure 25: Index.html - leg table	39
Figure 26: Index.html - mode selector and height adjustment	40
Figure 27: Index.html - actuator settings modal.....	40
Figure 28: Script.js - initiating actions.....	41
Figure 29: Script.js - buildActuatorsTable().....	41
Figure 30: Script.js - populating modal	42
Figure 31: Script.js - move() and saveConfig()	42
Figure 32: Script.js - right controller event handling	43
Figure 33: Style.css -CSS Grid Layout Model.....	43
Figure 34: Style.css - layout arrangement	44
Figure 35: Style.cc - mode selector (MS) formatting	44
Figure 36 Main.cpp - accessing the credentials and setting WiFi.....	46
Figure 37: Main.cpp - mapping UI with SPIFFS files.....	46
Figure 38: Main.cpp – updating actuator config.....	47
Figure 39: Main.cpp - handling gait and mode selector	47
Figure 40: Main.cpp - loop().....	48
Figure 41: GaitHandler.cpp - Sequence Iteration	49
Figure 42: GaitHandler.cpp - LiftLeg().....	49
Figure 43: GaitHandler.cpp - calculating offsets.....	50
Figure 44: IK - leg top view.....	50
Figure 45: IK - leg side view.....	51
Figure 46: InverseKinematics.cpp - getCoxaAngle()	51
Figure 47: InverseKinematics.cpp - getFemurAngle().....	52
Figure 48: InverseKineamtics.cpp - getTibiaAngle().....	53
Figure 49: SR518.cpp - move()	55

Figure 50: SR518.cpp - calibration loop	55
Figure 51: SR518.cpp - setPosLimitMax()	56
Figure 52: SR518.cpp getTemperature().....	56
Figure 53: JsonHandler.cpp - getJsonElement().....	58
Figure 54: JsonHandler.cpp - getJsonActautors()	58
Figure 55: JsonHandler.cpp - updateActutors());.....	59
Figure 56: JsonHandler.cpp - readGait()	59
Figure 57: SpiffHandler.cpp	60
Figure 58: Actuator.json.....	61
Figure 59: Credentials.json	61
Figure 60: Creep Gait – forward.....	62
Figure 61: Forward_gait.json	62
Figure 62: Rotational Gait – left.....	63
Figure 63: Gant Chart.....	65
Figure 64: Processing of forward gait movement.....	70

1. Introduction

1.1 Background & Motivation for the Project.

Outbreak of Covid-19 pandemic is causing widespread economic hardship across business, consumers, and communities. Organisations need to re-think the ways they are doing their business and address key aspects of their operations such as supply chains or workforce. i.e. many technology companies responded by moving their staff to remote workplaces, [“Big Tech was first to send workers home...”¹](#) and invested heavily in automation and virtualisation technologies.

For the key industries such as healthcare, manufacturing or logistics, remote operation is usually not a feasible option as number of tasks still needs to be performed manually or require human interaction i.e. healthcare specialists attending hospitalised patients, logistics operators delivering goods, manufacturing staff. Even with the best preventive measures in place, a risk of exposure is still very likely, and it can be extremely dangerous for the communities if the key organizations and businesses that they rely on, ceased their operation due shortage of workers i.e. due to illness, or if the environment becomes too contaminated for human presence.

These scenarios can be partially rectified if there were available robotic systems that could step in with little to no configuration and supplement manual labour tasks. There are currently number of robotic solutions within i.e. in logistics such Amazon Warehouse Robots, or in manufacturing such as KUKA automated assembly lines, however, all these are highly inflexible, designed to be task specific and ability to re-configure and deploy it in short time to supplement manual task performed by humans are none existing.

1.2 Project Aim.

The aim of the project is to develop contained quadruped robotic platform with flexibility of movement allowing for traversing uneven surfaces with inclines and obstacles such as stairs while being controlled with mobile device.

The project will also be designed with future expandability in mind by implementing connectivity features for additional modules such as single board computers or external sensors as well as payloads.

¹ Rachel Lerman, Jay Greene, *Big Tech was first to send workers home. Now it's in no rush to bring them back*. The Washington Post, May 18, 2020, <https://www.washingtonpost.com/technology/2020/05/18/facebook-google-work-from-home> (accessed on October 20th, 2020)

1.3 Project Objectives.

Since the projects requires development of both hardware and software components, the objectives can be split based on type and its importance to the project success.

1.3.1 Hardware Critical:

1. Design and assembly of the core unit with integrated battery pack and 1st stage actuators.
2. Design and assembly of the leg units with 2nd and 3rd stage actuators.
3. Design and assembly of the controller board with onboard processing and WIFI capabilities.

1.3.2 Hardware Desired:

1. Pressure sensor functionality for each leg assembly.
2. Integrated 9 axis IMU module.
3. Mechanical and electrical connection for the optional custom modules.

1.3.3 Software Critical:

1. Development of the necessary firmware libraries for responsive operations and control of actuators movement with gait system.
2. Developing clean interface allowing for intuitive operation and monitoring of the robot parameters.

1.3.4 Software Desired:

1. Development and integration of Inverse Kinematic (IK) libraries for responsive movement and operation with IMU module.
2. Development ability to control the device using API or http query requests that could allow 3rd party applications to access the device.

To meet the specified objectives, several activities must be undertaken before commencing the work

- Assess existing solutions in related technologies to identify functional and non-functional requirements as well as value proposition.
- Select hardware and software components for best chance of project success.
- Prototype and evaluate designs for optimal configuration.
- Test both hardware and software components and make evolutionary adjustments.
- Review the project based on its success criteria and adherence to project requirements.

1.4 Outline of Dissertation Structure.

The report is divided into several chapters outlining each section of the project.

Chapter 1 – Introduction

Chapter contains background information behind the project motivation, aims and objectives as well as outlines document structure.

Chapter 2 – Literature Review

Chapter evaluates existing solutions, identifies stakeholders and market gap that can be fulfilled with proposed project.

Chapter 3 – Requirements and Risks

Chapter will establish set of functional and non-functional requirements and assess potential risks and their impact on the project success.

Chapter 4 – Design

Chapter explains in details hardware and user interface (UI) design.

Chapter 5 – Code Implementation

This chapter will focus on code architecture and implementation at each layer.

Chapter 6 – Project Planning

This chapter will examine available Software Development Life Cycle models and define the sprint structure with key deadlines and objectives.

Chapter 7 – Testing and Evaluation

This chapter will present testing results and evaluate completion level against set functional and non-functional requirements from Chapter 3.

Chapter 8 – Conclusion

This chapter will summarise the project by highlighting the complexity, challenges and future improvements.

2. Literature Review

This chapter will focus on evaluating the existing solutions based on key characteristics and identifying possible market gaps using Value Engineering and Value Analysis frameworks.

2.1 Existing Solutions

Currently there are several high-level robotic platforms, however, to make fair comparison, only devices that are commercially available, share same design philosophy and capabilities will be taken into comparison. Wheeled, industrial as well as any non-commercially available projects such as private or research will not be considered for the comparison.

The designs that are taken into consideration are expected to be of quadruped (or '4 legged') design with similar functionality and capabilities that can be quantified and fairly compared. There is limited methodology in terms of the selecting key characteristics for successful robot design, however, after reviewing several articles by [MIT](#)² and [IEEE](#)³, a few aspects can be defined.

- Dexterity can be defined by the device wide range of movements and flexibility.
- Capabilities can be defined as ability to adapt to mission specific tasks throughout external modules, payload expansions and ease of integration with external systems.
- Mobility of the robotic platform can address several technological aspects, such ability to operate untethered, battery run-time, ability to travers uneven terrain or even obstacles but can also be extended to compact size, and ease of transporting the device.
- Autonomy it also becoming increasingly important ability and can be define by the ability of the device to autonomously adopt to changing environment, making decisions supported by AI and sensing data in relation to course planning or obstacles.
- Accessibility could be defined by ease of operation with simple and intuitive controls and minimal setup, additional components
- Costs of design, manufacturing, procurement, and deployment would define availability for different consumer levels, i.e. individual, enterprise, public etc.

2.1.1 Spot – Boston Dynamics.

² Will Knight, *Robot dexterity*, MIT Technology Review, April 2nd, 2020, <https://www.technologyreview.com/technology/robot-dexterity/> (accessed on November 10th, 2020)

³ E. Cha, A. D. Dragan and S. S. Srinivasa, *Perceived robot capability*, 2015 24th IEEE International Symposium on Robot and Human Interactive Communication, <https://ieeexplore.ieee.org/document/7333656> (accessed on November 8th, 2020)

Spot is well known robotic platforms that is currently being introduced into market by Boston Dynamics and it originates from early BigDog design that was developed together with JPL and Defense Advanced Research Projects Agency (DARPA).

Spot is currently the most advanced robotic design that is becoming commercially available. As per manufacturer specification⁴, it incorporates quadruped, dog-like design with three degrees of freedom (3DOF) per each limb and full 360-degree computer vision supported by sophisticated AI. This provides Spot with extreme cross terrain mobility and obstacle navigation with up to 90 min autonomy.

Spot can also be extended with number of custom expansion modules such as long range lidar, processing units, 6 DOF attachable arm as well as other custom modules. The capabilities can be expanded by SDK package that allows for mission planning and more advanced controls, however, a tablet like controller is also provided with the device that allows for easy, and intuitive controls.

The biggest drawback of the device it is high price-tag starting at \$74,500 making it extremely inaccessible for individual use and even small organizations, making it only viable for large sector organisations, heavy industries, military and some public institutions.

Table 1: Spot - Key Characteristics Score



Spot by Boston Dynamics	
Dexterity	****
Capabilities	****
Mobility	****
Autonomy	****
Accessibility	****
Cost	*

2.1.2 A1 – Unitree

⁴ Spot, Boston Dynamics 2020, <https://www.bostondynamics.com/spot> (accessed on 15th November 2020)

A1 by Chinese start-up Unitree⁵, is another advanced robotic platform that is currently being introduced into the market and it very similar to work completed by [MIT's Mini Cheetah](#)⁶ project that share similar quadruped design as Boston Dynamic's Spot but in smaller package.

The device appears to have advantage in terms of mobility and dexterity in comparison with Spot as per manufacturer claims, it can reach higher moving speeds, accelerations as well as athletic performance by completing a consecutive backflip and slightly higher battery running time up to 2.5 hours. It also comes with intuitive gamepad controller and mobile app, however, is lacking more advanced SDK package with advanced controls and mission planning.

The hardware is also limited with no expansion modules and simpler computer vision systems that does not support full 360-degree vision.

The A1 is still very strong and attractive device with significantly lower price tag of \$10,000 in comparison with Spot and it allows to be more accessible but still unlikely to be an option for individual clients.

Table 2: Key Characteristics Score



A1 by Unitree	
Dexterity	*****
Capabilities	***
Mobility	*****
Autonomy	****
Accessibility	*****
Cost	**

2.1.3 SQ3U – LynxMotion

⁵ Unitree Robotic, A1, <https://www.unitree.com/products/a1/> (accessed on 15th November 2020)

⁶ Jennifer Chu, MIT News Office, *Mini cheetah is the first four-legged robot to do a backflip*, March 4th, 2019, <https://news.mit.edu/2019/mit-mini-cheetah-first-four-legged-robot-to-backflip-0304> (accessed on November 15th 2020)

SQ3U is quadruped platform by Lynxmotion⁷ that using symmetric quadruped design and has been available as a hobby kit. Thanks to its symmetric design with 3 degrees of freedom (DOF) per each leg provides ability to walk in any direction.

The design is using standard hobby grade servos paired with Arduino board and Bluetooth allowing for connecting with wireless controller or external applications for low level processing. It can also be programmed using visual interface with custom PC software supplied by the manufacturer.

SQ3U does not appear to contain any orientation, pressure or computer vision systems which limits the capabilities of the device. Usage of hobby-grade components makes it an affordable option at around \$500 and with easy connectivity with optional external controller and simple software but requires additional investments in battery and controller. Due to its limitations has no practical implementation it is aimed towards hobbyists and students.

Table 3: SQ3U - Key Characteristics Score



SQ3U by Lynxmotion	
Dexterity	***
Capabilities	*
Mobility	**
Autonomy	*
Accessibility	**
Cost	*****

2.1.4 D’Kitty – Trossen Robotics

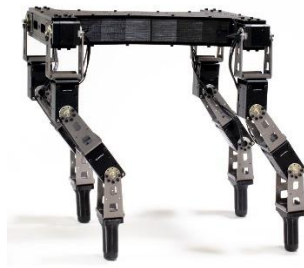
D’Kitty is another dog-like design from Trossen Robotics⁸ that has been based on work for Robel⁹ project. It consists of 12 Dynamixel robotic actuators with 3 degrees of freedom per each limb attached to 3D printed base. The kit does not come with batteries and requires access to 3D printer or laser cutter for non-assembled version. The platform does not come with any sensors, computer vision or on-board processing and its circuitry is limited into Dynamixel U2D2 controller and power hub making it very barebone platform. The platform can be expanded and as an example of Robel project use custom modules and sensor for more advanced task, i.e. HTC Vive sensor for orientation. The platform limited electronics and sensing capabilities as well as and high price at around \$4,500.00 for assembled version makes it option that it can be justified for only high-level hobbyists or researchers.

⁷ Lynxmotion 2020, <http://www.lynxmotion.com/c-26-quadrupods.aspx>

⁸ Trossen Robotics 2020, <https://www.trossenrobotics.com/d-kitty.aspx>

⁹ M. Ahn, H. Zhu, K. Hartikainen, H. Ponte, A. Gupta, S Levine, V Kumar, *ROBEL: Robotics Benchmarks for Learning with Low-Cost Robots*, September 25th, 2019, Cornell University (accessed on November 15th 2020)

Table 4: D’Kitty - Key Characteristics Score



D’Kitty by Trossen Robotics

Dexterity	***
Capabilities	*
Mobility	**
Autonomy	*
Accessibility	*
Cost	**

2.2 Products comparison and identifying gaps.

When comparing available options, we can clearly see the differences between the solutions, and we can also identify potential value of each product. In Figure 1. we can see the comparison of key characteristics represented in radar chart and in Figure 2. we can view the capabilities vs cost ratio. By looking at both figures we can clearly state couple of observations:

- The high costs products are more capable and thus offer more benefits than the cheaper options as they use higher-grade components, more sophisticated software and hardware. This is also making the products more suitable for real world applications.
- The second observation is the price differentiation, i.e. high price of Spot or A1 makes it inaccessible for individuals or even small-medium size organisations where their capabilities could be utilised. The cheaper options are more affordable for individuals and small organisations such as education institutions, however, are fairly limited in capabilities making it often hard to use and with no real-life application.

Based on the above assumption’s we can deduct that market gaps are sitting within the balance between affordability and usability for the proposed of the type of the devices.

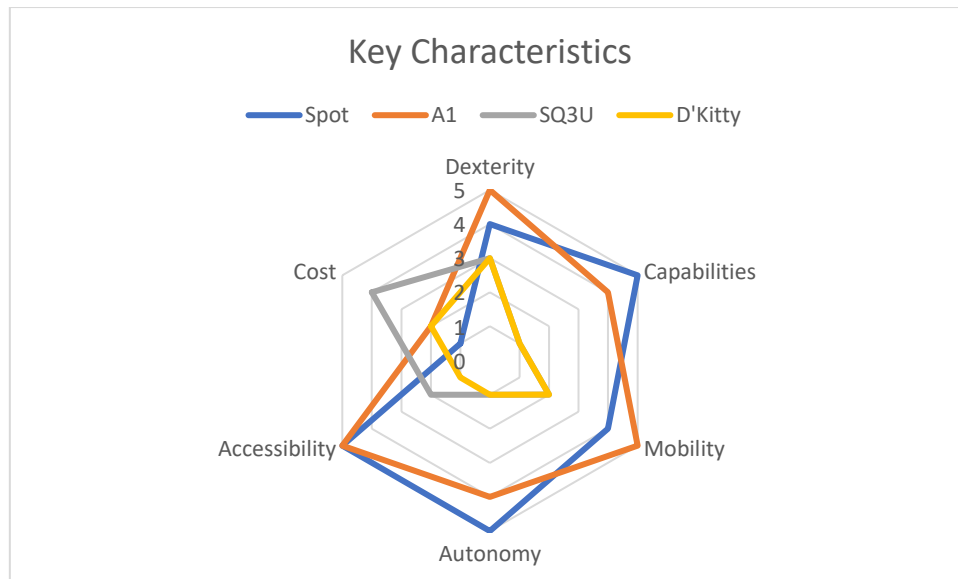


Figure 1: Key Characteristics

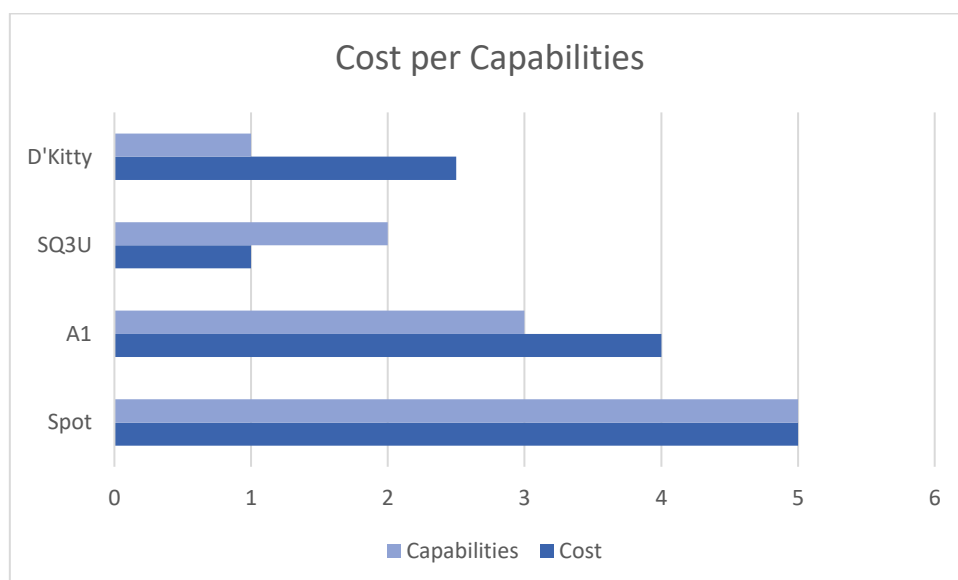


Figure 2: Cost per Capabilities

2.3 Value Engineering and Values Analysis

To better understand the base for the set of requirements, Value Engineering and Values Analysis methodology by Save¹⁰ can be applied to QPR project as it emphasizes capabilities, benefits and costs which are important for this project.

Value Engineering is primarily concerned with value as integral part of overall quality of the product and can be represented as division of functionality that can be quantified by desired

¹⁰ About Value Engineering, Save, 2020, <https://www.value-eng.org/page/AboutVE> (accessed on November 19th, 2020)

capabilities such as ability to climb objects vs cost of resources, i.e. actuators cost and number that can facilitate the climbing functionality.

In context of this project, the aim is to expand functionality that lower end solutions do not offer and at the same time selecting right components that will still allow the design to be affordable thus accessible for individual consumers.

Following high level functionalities can be identified:

- 1. Desired Functionality:** unit to be able to operate untethered.
Solution Proposed: Self-contained unit that combines batteries, computation and sensing in the same package and does not require tether for power or data links.
Cost Expected: Initial higher costs but usage of off-shelf components, and new technologies such as 3D printing should lower the development expenses.
- 2. Desired Functionality:** ability to navigate over uneven terrain and climb obstacles while sustaining balance and movement.
Solution Proposed: environmental sensing throughout IMU sensor (Gyroscope, Accelerometer and Magnetometer), pressure feedback through sensors installed in each of the limb and actuator feedback that can enhance and support the movement and stability of the platform.
Cost Expected: usage of robotic actuators will be most major element of the design costs. The aim is to use mid-grade elements that will provide good balance between price and performance.
- 3. Desired Functionality:** unit to be operated remotely without need for additional controller, i.e. using tablet or smartphone and able to be controlled using HTTP protocol.
Solution Proposed: Using WIFI capable electronic microcontroller that can facilitate both HTTP communication as well as on-board processing.
Cost Expected: usage of available WIFI modules with good HTTP implementation while remaining cost efficient i.e. ESP8266¹¹ or ESP32¹² by Espressif.

Value Engineering has also been supported by Value Analysis in order to improve value of the QPR project. These evaluations have been spaced into 6 separate phases.

- 1. Introduction:** Identified following risks of the projects:
 - Delivery time of the components and tooling heavily affected by Covid pandemic.
 - Costs, additional tools, and components need to be procured for project completion such as Spot Welder for battery pack assembly.
 - Time constrains as the projects require significant hardware and software development and testing time and limited due to full time employment and other activities.
- 2. Functional Analysis:** defined primary and secondary function

¹¹ ESP8266: A cost-effective and highly integrated Wi-Fi MCU for IoT applications, <https://www.espressif.com/en/products/socs/esp8266>

¹² ESP32: A feature-rich MCU with integrated Wi-Fi and Bluetooth connectivity for a wide-range of applications, <https://www.espressif.com/en/products/socs/esp32>

- Primary: compact robotic platform that can be transported and operated by an individual, with easy user interface, minimal setup time and expendability in place as well as will not require additional components such as batteries or controllers.
 - Secondary: Real life application through expanded and upgraded hardware software as well as external modules allowing to perform variety of tasks such as security monitoring, personal assistance, carrying payloads etc.
3. **Creative:** different options for overall design and component design has been considered, selected the one that would meet both functional and cost criteria.
 4. **Evolution:** number of different design prototypes have been constructed before selecting the right options:
 - 2 different battery prototypes build.
 - 3 different controller boards build.
 - 4 different leg either fully or partially build
 - 2 different sets of actuators tested with the design.
 5. **Development:** spaced in number of sprints with incremental functionality added at each cycle, incremental improvements made along the design.
 6. **Presentation:** to the stakeholders, through comparison with similar industry (see section 2.4.)

2.4 Drone industry as example of possible product future.

We can take as an example the commercial drone industry that has emerged in last 10 years. The currently most common drone design is the multirotor designs such as quadcopters, and these were initially developed by hobbyists that started combining new different technologies such improved batteries and electric motors, more powerful and affordable microcontrollers, and environmental sensors. This led to development of new RC toys however with continues improvements and capabilities it led to raise of commercial drone industry and today products from [DJI](#) such as Mavic or Phantom are not only an affordable for individual consumers but often used for commercial applications (Giones, Grem, 2017)¹³.

The same technologies can be applied for development land-based robotic platform, the 'legged' design provide the best mobility and capabilities as they can mimic animal-like movement and current state of this technologies are similar to drone industry from early 2000.

By following the trends QPR can be one step closer to affordable, consumer robots.

¹³ F. Giones, A. Brem, December 2017, *From toys to tools: The co-evolution of technological and entrepreneurial developments in the drone industry*, Business Horizons, <https://www.sciencedirect.com/science/article/pii/S0007681317301210> (accessed on November 15th)

3. Requirements and Risks

Base on the findings from the Chapter 2, a proposal for the device was drafted outlining high-level customer and product requirements as well as to identify any limiting factors and stakeholder involved in the process that can affect the overall outcome.

By applying CMMI Requirement Development¹⁴ framework, the data gathered was refined into set of Functional and Non-Functional requirements that cover customer needs, provide better value proposition while maintaining low cost. Reliance on external partners such as component suppliers, available technologies and own skillset was also taken into consideration as limiting factors.

Finally, a risks assessment was completed and evaluated using risk matrix with appropriate prevention strategies.

3.1 Functional Requirements

Functional requirements are mainly concerned with functional features that can improve customer experience, satisfaction and perceived value while staying within established time, costs, and technology limitations.

Table 5. below presents the selected key functional requirements with their description, justification, and prioritisation levels.

FR01	User must be able to control device from web interface without need for special controller.	Reason: simplifies the device and costs associated, allows for more friendly UI access.	Must
FR02	User should not need to download any additional app's, drivers, and any companion software.	Reason: simplify the deployment process and improving user experience.	Should
FR03	User must be able to use on-screens controls such as joysticks, buttons to operate the device	Reason: familiar UI objects allowing for intuitive operation.	Must
FR04	Robot must provide at least 15 min. of operation time before being re-charged.	Reason: user should be able to complete the full presentation of the platform.	Must
FR05	Robot must be able to move in any direction on even surface.	Reason: key capability	Must
FR06	Robot body should be able to maintain orientation in any deflections within min. 30 degrees horizontal or vertical deflection.	Reason: better stability of the platform and increase in capabilities	Should
FR07	User should be presented with live data from actuators such as Voltage, Current Position, Temperature.	Reason: user able to monitor robot operation and step-in if a fault occurs.	Should

¹⁴CMMI Limited, March 2002, *Requirements Development (RD)*, <http://www.cmmi.co.uk/cmmi/RD.htm> (accessed on November 23rd)

FR08	User must be able to set limits on position, load or torque for each actuator using web interface.	Reason: user able to configure and optimise the operation of the device and avoid accidental damage.	Must
FR09	User should be provided with live environmental data such as orientation from MEMES sensor (gyroscope, acceleration, magnetometer) and pressure per each limb.	Reason: user can better assess the situation and monitor the operation avoiding any accidental damage.	Should
FR10	Robot must be able to move without any tethers or harness	Reason: key capability, device should be fully self-contained.	Must
FR11	Robot should be able to climb over small obstacles such as stairs.	Reason: key capability, allowing for future navigation development	Should
FR12	Robot should be able to move any direction within 30 degree horizontal and / or vertical deflection.	Reason: key capability, allowing for future navigation development	Should
FR13	Robot should have ability to interface with external devices.	Reason: key capability, allowing for future expansion with external sensors.	Could
FR14	Robot could be integrated with LIDAR Sensor	Reason: To provide SLAM navigation	Could
FR15	Robot could be integrated using API with external systems i.e. ROS	Reason: Improving capabilities via advanced controls and mission building.	Could

Table 5: Functional Requirements

3.2 Non-Functional Requirements

NFR01	The project should remain price competitive with competition and affordable for individual consumer.	Reason: the device is intended for individual customers as well as small organisations.	Must
NFR02	The UI should be compatible with all common Smartphones and Tablets screens.	Reason: allow ability to be compatible with majority of mobile devices.	Should
NFR03	The access to connect with the device should be secured with password.	Reason: to protect access to the device.	Must
NFR04	The UI should be clean and intuitive.	Reason: improve user experience.	Must
NFR05	Robot controls and feedback should be responsive.	Reason: improves user experience.	Should
NFR06	Robot should be safe to operate to avoid injuries or accidental damage.	Reason: safety concerns	Must

Table 6: Non-Functional Requirements

3.3 Identified Risks

Risk assessment has been completed using ISO 9001 Ranking Risk¹⁵ framework and based on limiting factors such as costs, technologies, skills, time, and suppliers. Table 7., describes identified risks with severity and probability of occurrence as well as mitigation strategy.

Risk	Description	Mitigation	Likelihood and Consequences	
RS01: Accidental Damage of Key Components	This has moderate level of occurrence during prototyping or assembly phase and can have high consequence level if the affected part is actuator.	Safety procreation in use to ensure that all connections are properly wired before powering any system. Stock replacement if required.	Without Mitigation Strategy	
			Likelihood: Moderate	Consequences: Catastrophic
			With Mitigation Strategy	
			Likelihood: Unlikely	Consequences: Moderate
RS02: Stock Delays	Many items need to be procured from overseas that require long lead time and can be even delayed further due to outgoing pandemic	Making procurement decisions ahead of time based on components availability and importance. Resulting to faster shipping of items from within the UK	Without Mitigation Strategy	
			Likelihood: Moderate	Consequences: Moderate
			With Mitigation Strategy	
			Likelihood: Moderate	Consequences: Minor
RS03: Hardware Design Errors	Custom controller board being built for the project that requires external PCB manufacturer.	An extensive prototyping phase in use to make sure the design is 100% operational before finalising. Building in-house controller board using off-shelf modules.	Without Mitigation Strategy	
			Likelihood: Moderate	Consequences: Major
			With Mitigation Strategy	
			Likelihood: Unlikely	Consequences: Minor
RS04: New Technologies	New Technologies being utilised and developed that can cause unforeseen consequences and delays.	Re-use available off-shelf hardware and software solutions. Make the changes to simplify the design to core functionalities.	Without Mitigation Strategy	
			Likelihood: Likely	Consequences: Major
			With Mitigation Strategy	
			Likelihood: Possible	Consequences: Moderate
RS05: Scope Creep	Missing expected deadlines for each sprint completion.	Work divided into flexible Sprints, with an overlap and ability to return to previous sprint at later time. Saving time by utilising ready solutions as well as simplifying build.	Without Mitigation Strategy	
			Likelihood: Likely	Consequences: Major
			With Mitigation Strategy	
			Likelihood: Possible	Consequences: Minor

Table 7: Risks

¹⁵ Keen, R., 2021. *Actions To Address Risks And Opportunities Explained [with procedure]*. [online] ISO 9001 Checklist. Available at: <<https://www.iso-9001-checklist.co.uk/6.1-actions-to-address-risks-and-opportunities-gbp.htm>> [Accessed 8 April 2021].

3.4 Risk Evaluation

The risk from Table.7 have been evaluated and ranked using probability and severity matrix and assigned score value as per Table 8. The higher the likelihood and more severe consequences, the higher the risk score and the higher risk of project failure.

Consequences	Catastrophic	5	15	19	22	24	25
	Major	4	10	14	18	21	23
	Moderate	3	6	9	13	17	20
	Minor	2	3	5	8	12	16
	Insignificant	1	1	2	4	7	11
Low Risk - 1 to 6			1	2	3	4	5
Moderate Risk - 7 to 18			Rare	Unlikely	Possible	Likely	Certain
High Risk - 19 to 25			Likelihood				

Table 8: Risk Matrix - Score Representation

Table 9. representing the risks and its positioning within the Risk Matrix without alternative strategy in place and Table 10. shows the same risks after applying the mitigation plans.

Consequences	Catastrophic	5		RS01 (19)			
	Major	4				RS02, RS03 (21)	RS05 (23)
	Moderate	3				RS04 (17)	
	Minor	2					
	Insignificant	1					
Low Risk - 1 to 6			1	2	3	4	5
Moderate Risk - 7 to 18			Rare	Unlikely	Possible	Likely	Certain
High Risk - 19 to 25			Likelihood				

Table 9: Risk Matrix - Risk Assessment without Mitigation Strategy

Consequences	Catastrophic	5					
	Major	4					
	Moderate	3		RS01, RS03 (9)			
	Minor	2	RS02 (3)	RS05 (5)	RS04 (8)		
	Insignificant	1					
Low Risk - 1 to 6			1	2	3	4	5
Moderate Risk - 7 to 18			Rare	Unlikely	Possible	Likely	Certain
High Risk - 19 to 25			Likelihood				

Table 10: Risk Matrix - Risk Assessment with Mitigation Strategy

As per findings, by implementing alternative actions, the risks have been significantly lowered in terms of both likelihood as well as severity resulting in much lower overall risk score across all listed items.

4. Design

The QRP design consists of several hardware and software elements that are interacting with each other to allow the user to control the device remotely. Figure 3. demonstrates the high-level design in which user will be able access the QRP with a mobile device using Wi-Fi connectivity. User then will be presented with a web browser UI that would provide both live feedback of the robot parameters and states as well as control inputs for different range of movements and gaits execution.

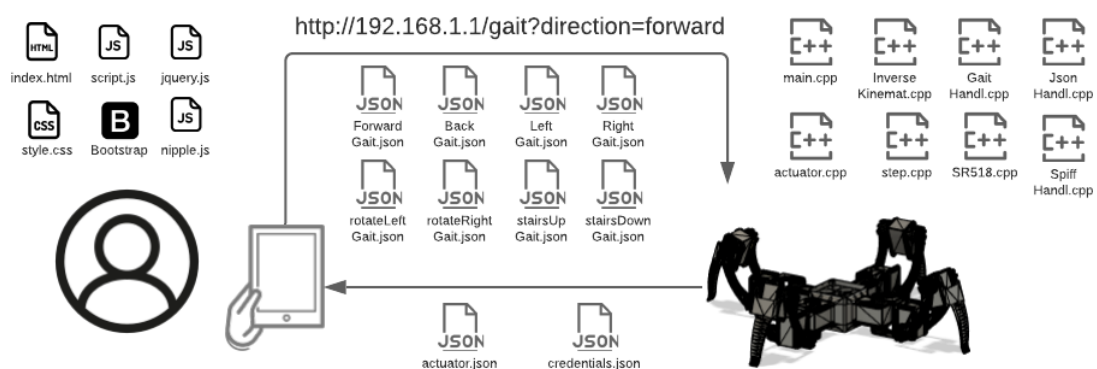


Figure 3: QRP - Design in Principles

The UI will be able to receive updated stats using Ajax and Json data and send HTTP – POST requests that will be processed by the back-end C++ code libraries.

Following chapter will focus on the hardware and UI designs while code architecture will be reviewed in Chapter 5. Implementation.

4.1. Hardware

Hardware consists of number both structural and electronic components that are assembled to satisfy the key requirement i.e. provide sturdy and strong frame that can withstand movements, battery that allows for minimum of 15 minutes of autonomy, processing power to complete necessary calculation in-flight and wireless communication to provide live data feedback and controls.

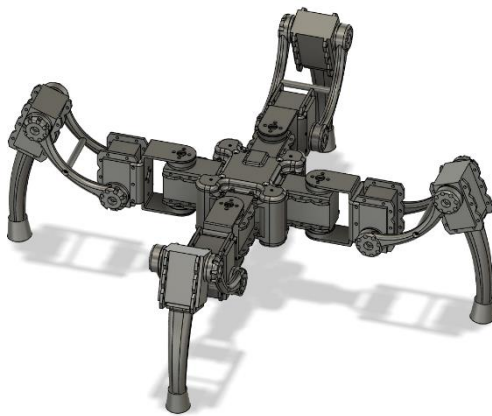


Figure 4: Frame Design vs Completed Build

4.1.1. Frame

Frame is divided into core unit and four separate leg assemblies with total of 39 main parts, 12 actuators and number of small fasteners such M2 nuts and bolts. Majority of the elements have been custom designed on Fusion 360¹⁶, parametric modelling software by Autodesk and 3D printed using black 405 nm UV resin with Elegoo Mars¹⁷, MSLA 3D printer. As per Figure 4. frame has a symmetric design with centre of gravity located as close as possible to the centre of the device (review Appendix 1: Frame Design).

¹⁶ <https://www.autodesk.com/products/fusion-360/overview?term=1-YEAR>

¹⁷ <https://www.elegoo.com/collections/3d-printing/products/elegoo-mars-lcd-3d-printer>

4.1.1.1 Core Unit

Core unit is the central element of the robot frame and it houses battery pack compartment, first stage actuators and standoffs for main Controller Board (CB) and Power Distribution Board (PDB). The part was designed in a way to provide maximum space for all necessary components while maintain the smallest as possible footprint to limit the load on the actuators. As per figure. 3 the part has been split into three individual elements

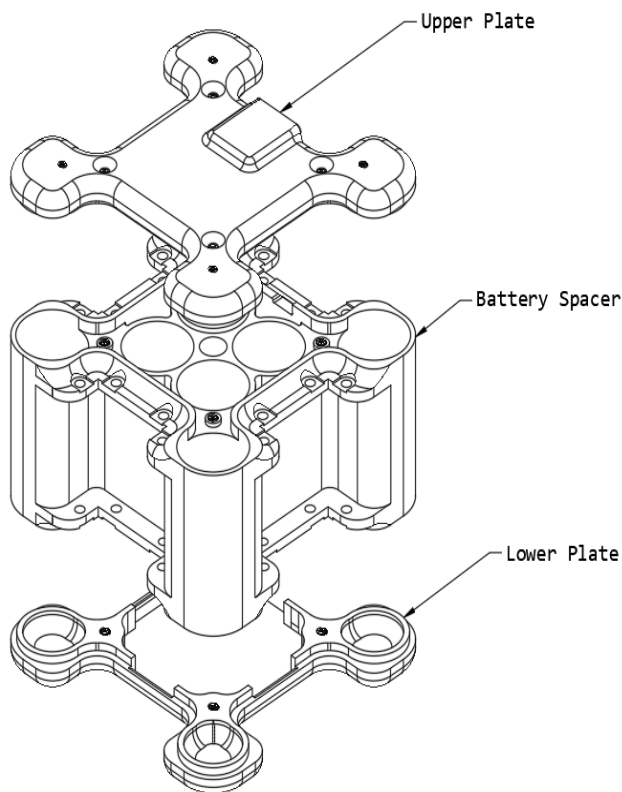


Figure 5: Core Unit Assembly

Upper Plate – contains opening for T-Plug, main power connector, threaded holes for M2.5 standoffs for mounting CB and PDB as well as tempered openings with holes for M2.5, flat head countersink bolts that are attaching the plate directly to Battery Spacer element.

Battery Spacer – has been modelled in a way to accept eight 18350 li-ion batteries in symmetric arrangement with cluster of four in the centre and four quadrants containing single cell. The spacer has also protruding edges with openings and holes for installing SR518 actuators from each side of the component. The middle section of the spacer contains empty cavities between the cells allowing for installing any necessary wiring and connectors as well as M2.5 threaded holes for attaching both upper and lower plates.

Lower Plate - contains opening for the Molex, balance charger connector, tempered openings with holes for M2.5 flat head countersink bolts and four small silicon 8mm protectors to prevent scratching the surface.

4.1.1.2. Leg Assembly

The key considerations when designing the leg assembly is to provide wide range of movement allowing for x, y, z or twist, reach and height coordinates to be plotted while maintaining compact size and appropriate dimensions of each sub-assemblies to ensure that the stall values of the actuators are not exceeded.

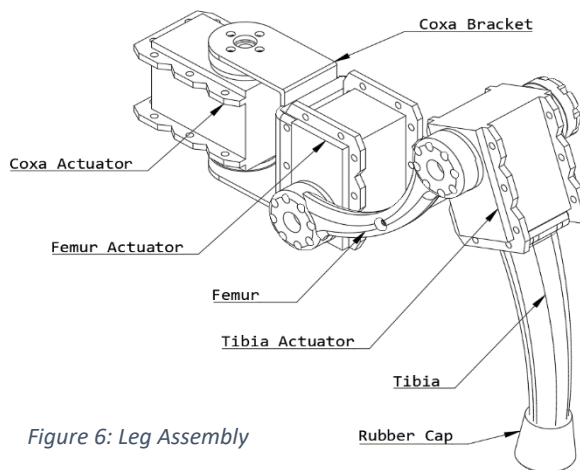


Figure 6: Leg Assembly

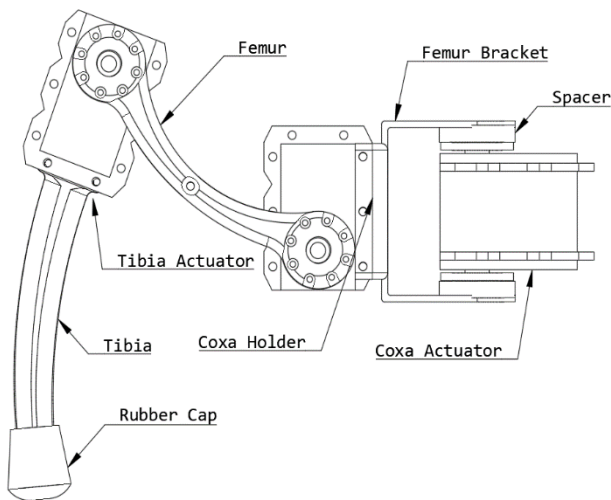


Figure 7: Leg Assembly - side view

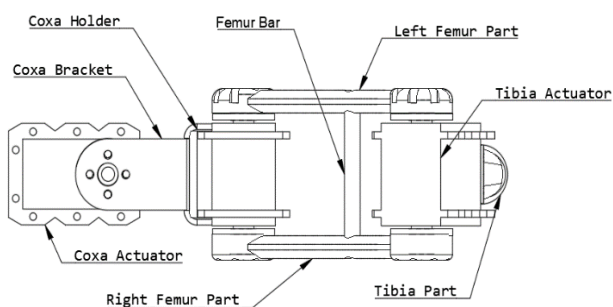


Figure 8: Leg Assembly - top view

There are four identical leg assemblies (Figure 4.) placed symmetrically from each side of the core unit, and divided into three anatomical sub-assemblies for coxa, femur, and tibia.

Coxa Assembly – provides horizontal movement of the leg assembly to desired twist angle and is directly connected to core unit through first stage servo. The coxa actuator is connected to metal U brackets using 3D printed spacers (Figure 7.) which connects to holder element that attaches the second stage, femur actuator.

Femur Assembly – provides initial vertical lift movement and together with tibia assembly allows for extending the leg assembly to desired height and reach values. The assembly consists of identical left and right femur parts separated with bar for increase of strength and stability (Figure 8.). The femur assembly is connecting on each side femur and tibia actuators.

Tibia Assembly – together with femur assembly, the tibia provides vertical lift movement and extending the leg to desired height and reach and consists of extended, angled part connected to tibia actuator at one end and ended with rubber cap to provide better grip to the leg.

4.1.2. Battery Pack

Battery pack consists of eight cells arranged in two serial, four parallel configuration (2S4P) (Figure 9.) and is connected with 7mm X 0.5mm nickel strips spot welded to the battery terminals.

Battery pack contains T-plug connector attached to positive and negative terminals providing battery output that is connected to PDB and the battery input with 3 pin JST XH male connector attached between positive, common, and negative terminals allowing for balance charging.

The cells used in the pack are ICR 18350 lithium-ion cells by LiitoKala¹⁸ brand with nominal voltage of 3.7 volts and indicated capacity of 950 mAh. The cells are also capable of constant discharge rate of 8 a (amps) and charging current of 800 mAh.

As per configuration (2S4P), the battery pack can deliver anywhere between 8.4 to 7.0 volts (nominal 7.4 volts) and combined capacity of 3,800 mAh (4 x 950 mAh) or 28.12 Wh of energy which provides over 15 minutes of mixed operation before needs recharging.

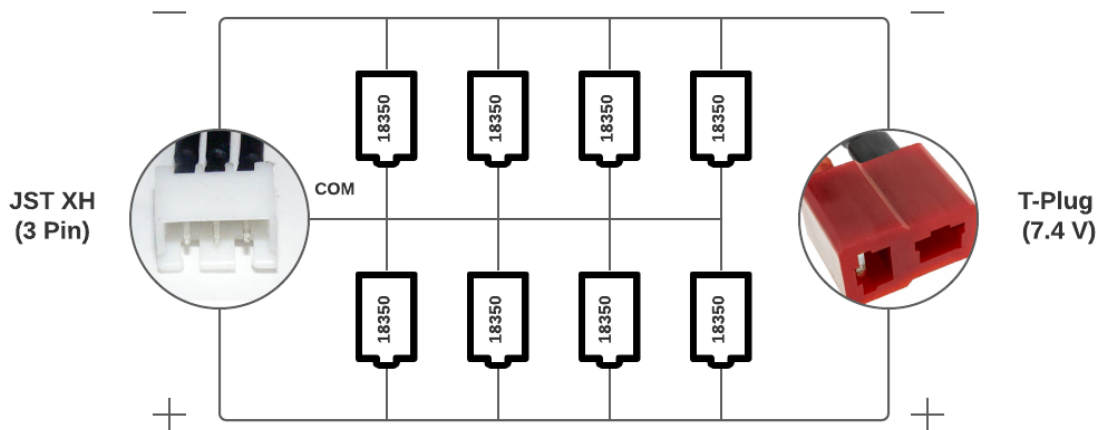


Figure 9: Battery Pack Schematics

4.1.3. Actuators

The project is utilising twelve robotic actuators with three units per each leg in coxa, femur and tibia arrangement (Figures 6-8.). The design allows the legs to extend with x, y, z coordinates thus providing three degrees of freedom (3DOF) each.

The actuators used are SpringRC SR518D¹⁹ models with RS485, half-duplex asynchronous serial communication that allow for serial chaining (daisy chaining) of each device throughout output and input connectors. Each node can be addressed and controlled individually, and provide full position, speed, voltage, temperature, and other feedback functions.

¹⁸ [https://lygte-info.dk/review/batteries2012/LiitoKala%20%2018350%20900mAh%20\(Pink\)%20UK.html](https://lygte-info.dk/review/batteries2012/LiitoKala%20%2018350%20900mAh%20(Pink)%20UK.html)

¹⁹ <http://www.springrc.com/en/nd.jsp?id=16>

The servos have 300° rotation angle with resolution of 0.32° (8 bit) and can also rotate continuously with adjusted speed and torque levels providing precision controls (feel free to review Appendix 2: Control Tables). The actuators can also be powered by wide range of voltages from 6.8V to 14V making it suitable for directly powering from battery pack without need for any additional voltage regulation.

Specification at 7.4v Supply Voltage	
Torque	16.5 kg-cm
Speed	0.19s /60 °
Position Resolution	0.32°
Idle Current	200 mA
Stall Current	1500 mA
Baud Rate	500 Kbps (max 1Mbps)

Table 11: SR518 - default specification



Figure 10: SR518 Actuator

4.1.4 Power Distribution Board (PDB)

Power Distribution Board is one of two custom designed PCB boards for the project and the main role is to distribute both power and RS485 serial data to each of the legs by series of the input and output, Molex Mini-Spox connectors. The board also contains 15 amps fuse to prevent from accidental short circuiting as well as over discharge current that could potentially damage the batteries, actuators as well as controller board.

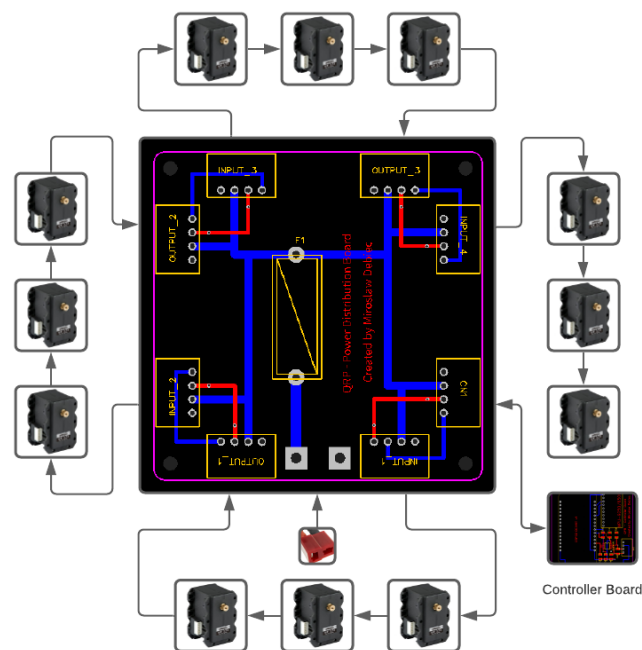


Figure 11: Power Distribution Board

As per Figure 11. PDB has input cable terminated with male, T-Plug connector that attaches directly to the battery pack and its positive line is being feed through the fuse to split power for each input and output leg connectors as well as controller board. The data lines RS485_A and RS485_B are provided from controller board and MAX485²⁰ serial converter and feed into leg connector input_1 and returned on the output_1 which is wired to input_2 connector. The process of wiring output to input connectors is being repeated for legs two, three and four while there is no need for the output connector for the leg four as it is the final in chain configuration (please review Appendix 3: Power Distribution Board).

4.1.5 Controller Board

Controller board has been custom designed PCB board to host main electronics for connectivity, processing, and sensing functionality. The board design (Figure 14.) has been simplified to save time and development effort and consists of ESP32 DEVKIT v1 development board, GY9250 IMP module and MAX485 serial communication transceiver accompanied with passive components. Controller Board also connects with PDB to transmit both data for RS485 controller as well as to receive power from PDB using Molex MiniSpox connector (please review Appendix 3: Controller Board Schematics).

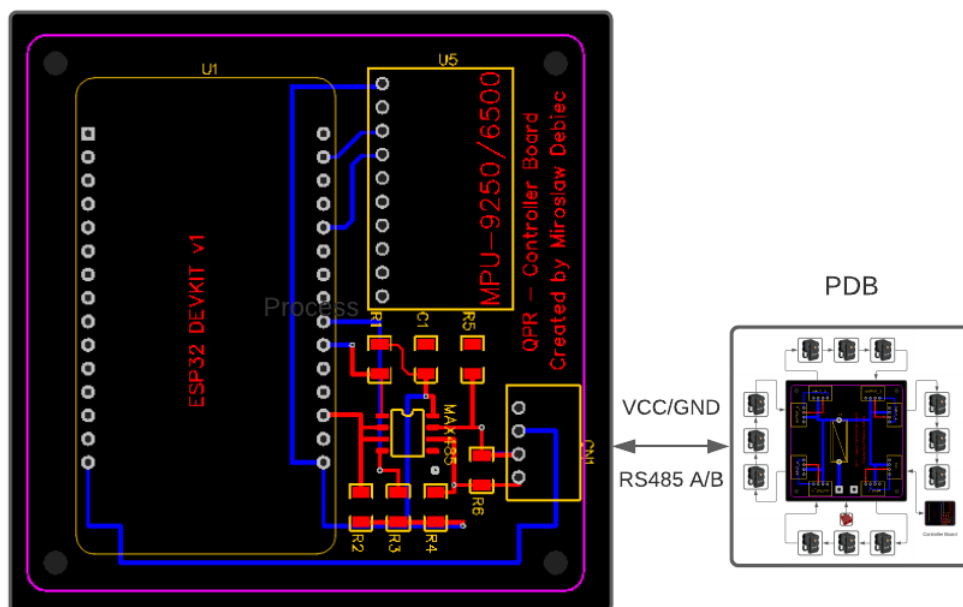


Figure 12: Controller Board

²⁰ <https://www.maximintegrated.com/en/products/interface/transceivers/MAX485.html>

4.1.5.1. ESP32 DEVKIT v1



Figure 13: ESP32 DEVKIT v1

ESP32 DEVKIT v1 is development board designed by Espressif and manufactured by several 3rd party developers. The board contains ESP32²¹ system-on-the-chip (SoC) Arm based microcontroller with integrated WIFI connectivity and 4MB of flash storage. In addition to the MCU, the kit also contains the CP2102, USB to TTL converter with micro-USB connector allowing for easy programming and is equipped with AMS1117 voltage regulator that allows to power MCU and additional low-power electronic IC's and modules reducing any need for additional power management block within the controller board design. Another benefit of using the ESP32 DEVKIT is its low cost of under £5.00 and breadboard friendly pinouts allowing for easy prototyping process.

4.1.5.2. GY9250 (Inertial measurement unit IMU)



Figure 14: GY9250 - IMU

GY9250 is a 9-axis module that provide 3-axis gyroscope, accelerometer and magnetometer data allowing for mapping device orientation and positioning in real time. The module uses MPU9250²² Digital Motion Processor (DMP) by TDK and can be interfaced with microcontroller via I2C and SPI communication protocol and can directly interface with ESP32 MCU.

The module has 16-bit accuracy for accelerometer, gyroscope and magnetometer data and wide range applications in motion controls.

²¹ <https://www.espressif.com/en/products/socs/esp32>

²² <https://invensense.tdk.com/products/motion-tracking/9-axis/mpu-9250/>

4.1.5.3. MAX485

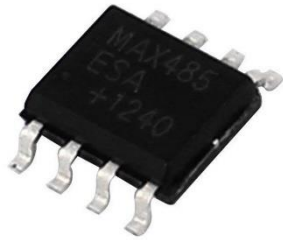


Figure 15: MAX485 IC

MAX485 is serial transceiver IC that allows for conversion of standard serial communication from MCU into RS485 standard. The IC is directly soldered onto controller board PCB with accompanying passive SMD components and interfaces with MCU's UART serial receiver (RX) and transmitter (TX) pins on the input and outputs RS485_A and RS485_B signal lines that through the Molex Mini-Spox connector are connected to PDB board and then wired into actuator input ports.

RS485 standard is common serial communication protocol used in industrial electronics and robotics. It uses differential based voltage system and half-duplex mode allowing for greater transmission speeds and distance, lower interference, and multipoint communication over standard RS232 protocol.

4.2. User Interface

User Interface for the project has been designed with several considerations according with Norman's 6 Principles of Design²³;

- **Visibility:** the interface is simple with clearly marked each section to indicate its functionality and purpose and without any deep-hiding functionality
- **Feedback:** the device will provide instant changes to the displayed actions based on user inputs and movement of the robot.
- **Affordance:** using intuitive on-screen switches and controls that users are familiar from other mobile games and programs, simulating physical game-controllers.
- **Mapping:** the tables with data and modals will resize as per screen size with scrolling options.
- **Constraints:** the responsive web design allows to have good visibility of all UI elements regardless of screen size.
- **Consistency:** the consistent fonts and formatting is utilised throughout the interface and all objects are aligned using grids.

²³ <https://www.engineess.io/insights/6-principles-design-la-donald-norman>

4.2.1. Accessing UI

Before user can access QRP's UI, a connection needs to be established between client device and the robot, this can be completed by connecting with QRP's SSID using standard Wi-Fi connectivity as the device is set by default into an Access Point (AP). In order to connect, client will need to provide the AP's password that is stored within the "*credentials.json*" file located on the flash memory of the robot. In addition, alternative credentials can be added to "*credentials.json*" in order to connect with the existing network.

As per Figure 16. once the connection has been established, the user can access UI by either typing IP address of the device ("*192.168.1.119*") or by using local DNS address "*http://qrp.local*".

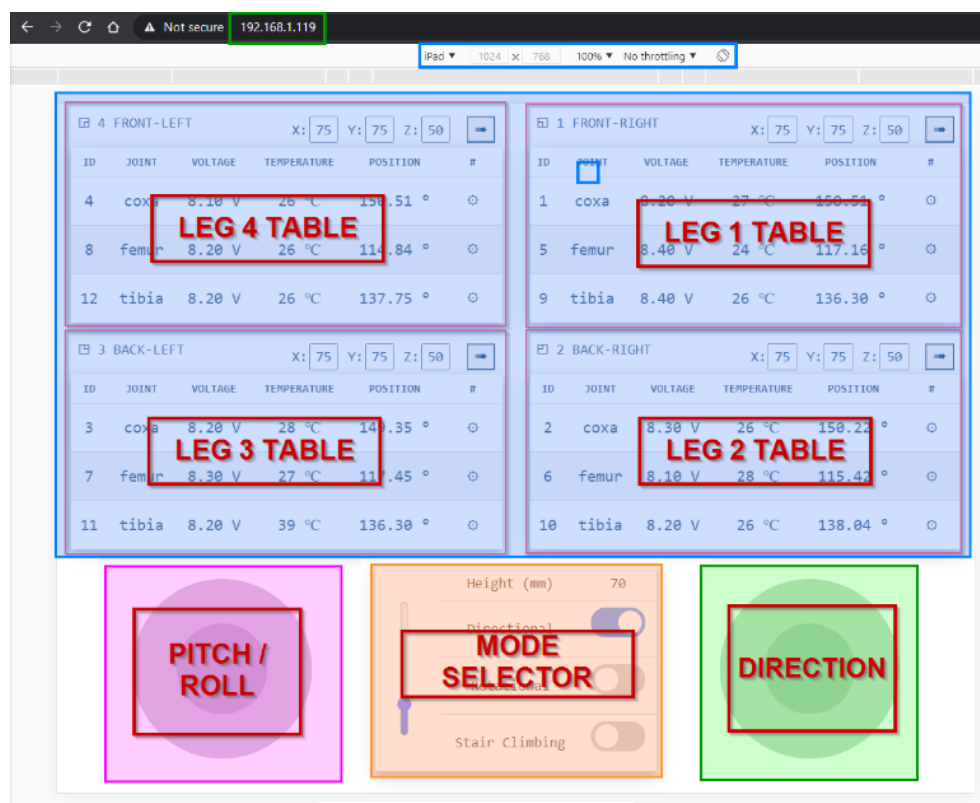


Figure 16: UI – general layout

4.2.2 General Layout

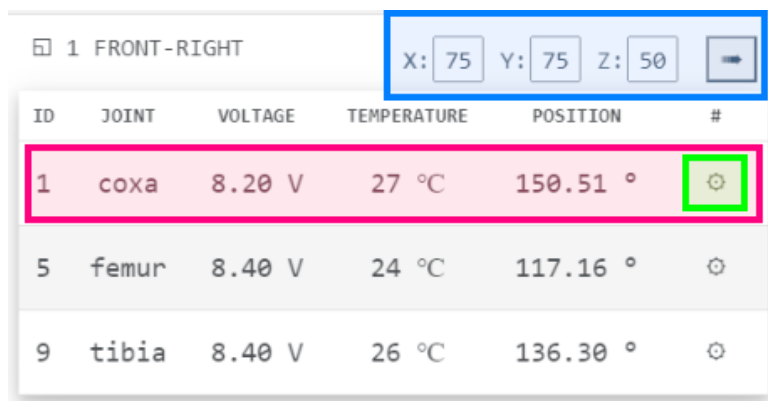
Figure 16: presents the general layout of the User Interface, optimised for tablet size devices with screen resolution of 1024x768 such as iPads, but it can also be used with larger Full HD screens. The UI is single page screen with several sub-sections that provide separate functionality.

- **Leg Tables:** this section takes majority of the screen real-estate and it is divided into four tables, representing each limb of the device in its default orientation. Each of the tables provides feedback and manual settings of the operating actuators.

- **Mode Selector:** located in central-lower section of the screen, allowing user to set elevation and select from different operation modes.
- **Pitch / Roll Controller:** located in lower-left section of the screen allowing to adjust pitch and roll orientation of the robot.
- **Directional Controller:** located in lower-right section of the screen and allows for the directional movement of the device in combination with selected mode of operation, elevation and pitch / roll inputs.

4.2.3 Leg Table

As previously mentioned, there are four identical tables that provide live data feedback and settings of the actuators in any given limb.






ID	JOINT	VOLTAGE	TEMPERATURE	POSITION	#
1	coxa	8.20 V	27 °C	150.51 °	
5	femur	8.40 V	24 °C	117.16 °	
9	tibia	8.40 V	26 °C	136.30 °	

Figure 17: Leg Table

The tables are clearly marked with the limb number and its location, i.e. in Figure 17. the table refers to Leg 1, located on front-right side of the robot.

Right to the leg description, there are X, Y and Z manual inputs where user can specify coordinates of the leg position to which they would like to move it to (highlighted in blue). Once the user presses the button with an arrow, an instruction will be sent to QPR to move the leg into the desired position in space. The robot will calculate the X,Y,Z coordinates and with use of Inverse Kinematics, into angular values for each of the respective servos, the leg will execute “Lift” to desired position and the field within the table will be adjusted with new values as the operation is being executed.

The table consists of three rows (highlighted in red) with each containing the information of the actuator ID, joint name that indicates its location within the leg, current voltage and temperature value for monitoring battery levels and ensures proper operation of the actuator, position field of the current position values and finally Settings Button with a “cog -wheel”

icon (highlighted in green) that opens modal windows with the additional configuration settings for the selected servo.

4.2.4 Actuator Settings

Once the user clicks on the settings button next to the actuator, modal window will be displayed (Figure 18.) with ability to modify upper and lower position limits as well as manually move the actuator to desired position within the specified limits.

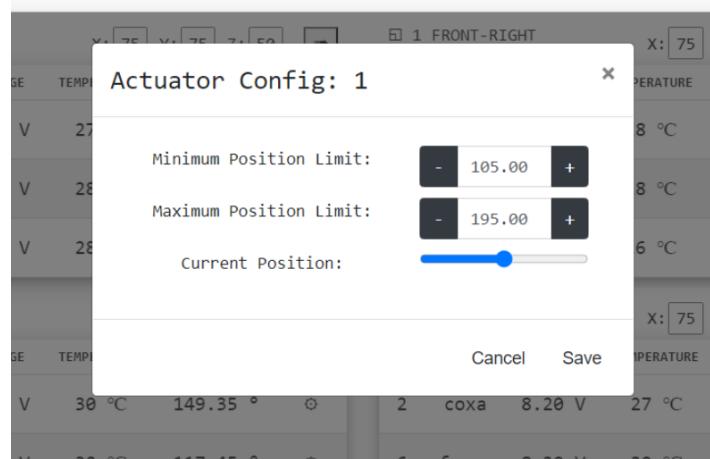


Figure 18: Configuration Modal

The limits are critical to ensure proper actuator operations and avoid accidental damage as the operation range of the joints are typically smaller than the range of the actuator. Once user saves the new values, this will be updated in *"actuator.json"* file that is stored on device SPIFFS memory and the actuator registers. This will ensure that the values are always correctly updated even after device is powered-off.

Additional setting can be implemented to allow for Voltage, Torque, Load and Temperature limits and functionality to change Baud-Rate speeds and ID assignments. .

4.2.5 Mode Selector

Mode Selector (MS) console located in central portion of the lower screen, provides elevation adjustment and mode selection functionalities that can be accessed by simple extending thumb from the controllers located at either ends.

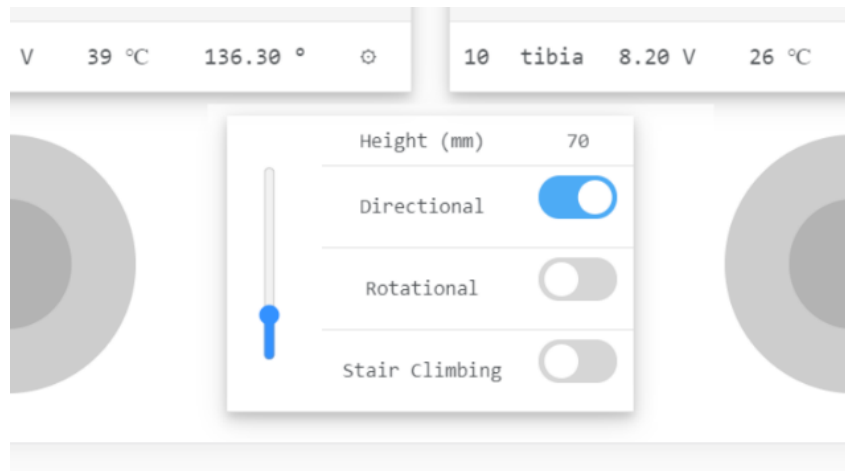


Figure 19: Model Selector Console

As per Figure 19, the elevation of the robot can be changed at any time by moving vertical slider located on left side of the MS console and the height value will be issued to the device as the offset to Z parameter which will ensure that its physical elevation is also updated. The height field will be also adjusted to indicate current level in millimetres. The elevation range is set from 50mm to 150mm in order to ensure the stability of the device without modifying the reach distance (vector of X and Y values).

Below the height indicator there are three mode selectors that user can select from;

- **Directional:** executes lateral movement when directional controller is being moved consisting of forward, back, left, and right gaits (Figure. 21).
- **Rotational:** executes lateral movement when directional controller is being moved up or down and rotational left and right gaits when moved left or right respectively (Figure. 20).
- **Stair Climbing:** a custom mode that when engaged the robot will perform climbing gait when controller moved up.

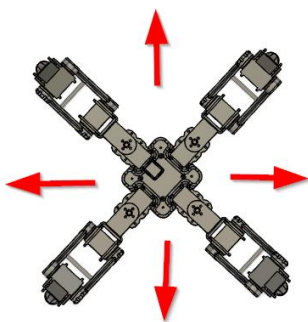


Figure 21: Directional Gait

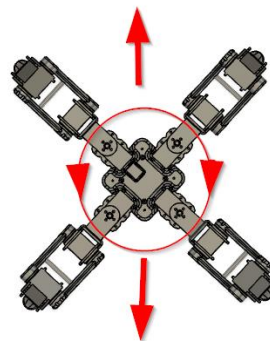


Figure 20: Rotational Gait

4.2.6 On-Screen Controllers

On-screen, controllers are provided by “nippleJS”²⁴ library and are touch capable making it ideal for the mobile device application. UI (Figure 22.) is using two of these controllers located at each side of MS console.

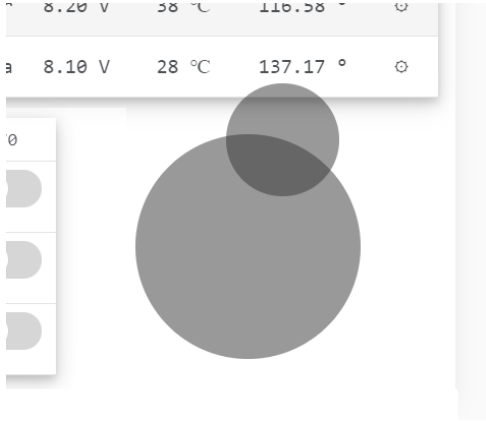


Figure 22: Direction Controller while engaged



Figure 23: QPR with adjusted pitch and roll values

Left controller, changes orientation of the robot with pitch and roll adjustments resulting in changes to actual device orientation (as per Figure 23.) while right joystick, can execute gaits, and move the robot as per selected mode of operations (Figure 20 and 21). Both joysticks can be used at the same time in combination, mixing their inputs and creating deflected gait movement.

5. Software Implementation

Software design of the QPR is using Layered Architecture²⁵ pattern with code split between Presentation, Business, Persistence and Data layers. Code files within each layer have specific roles and responsibilities within the application and relate to each other using data transfer objects (DTO) and dependencies.

²⁴ <https://yoannmoi.net/nipplejs/>

²⁵ <https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch01.html>

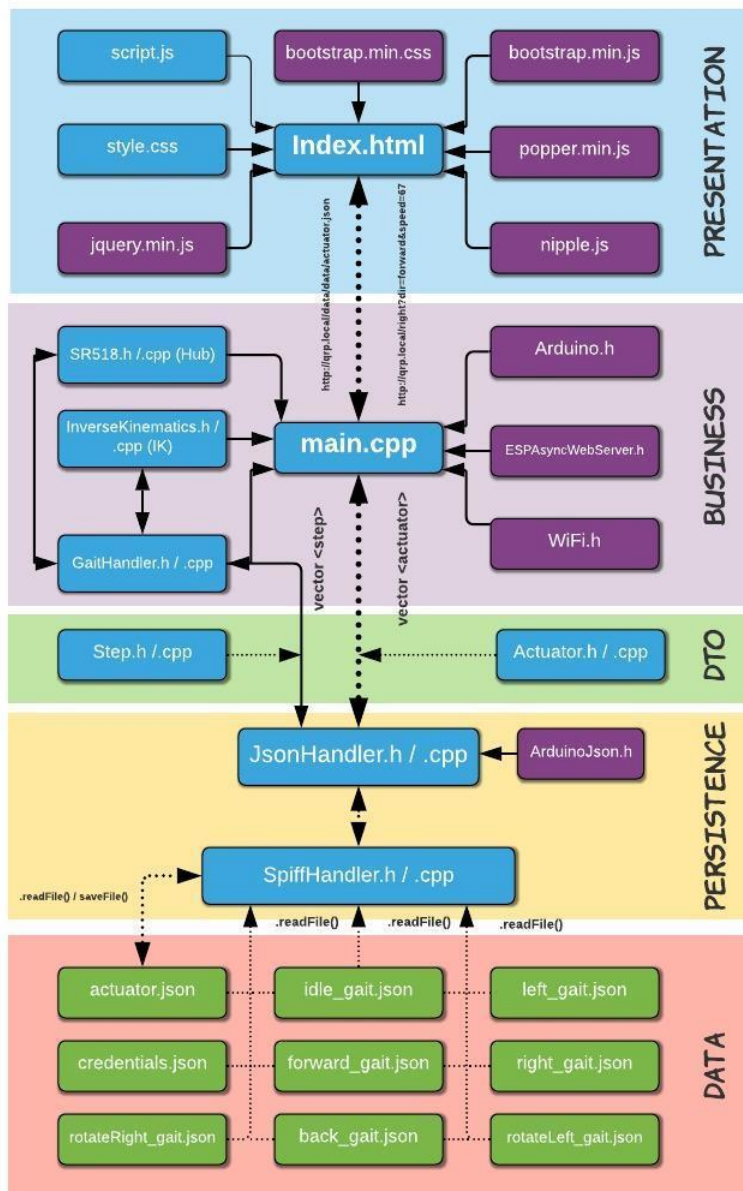


Figure 24: QPR – Software Architecture

Figure 24. demonstrates the code structure with outlined layers and dependencies between each code files. Only the custom code libraries marked in blue, and “.json” files highlighted in green will be covered in this chapter. Any 3rd party libraries (highlighted in purple) will be ignored. In addition, full code snippets are available as appendixes at the end of the report and will be link to paragraphs when applicable.

5.1. Presentation Layer

The UI aspect of the presentation layer will, be omitted as it was covered in chapter 4.2, this will be mainly focused on “index.html”, “script.js” and “style.css” code files.

Presentation Layer:

responsible for displaying UI, consists of “index.html” file linked to “styles.css” and “script.js” as well as 3rd party libraries such as “Bootstrap”, “jQuery” and “nipple.js”.

Business Layer:

responsible for processing inputs from UI, actuator and sensors and communicating the data between Presentation and Persistence layers. It uses multiply C++ classes linked to “main.cpp”

Data Transfer Object (DTO):

object classes that allow for creating dynamic collections of objects that can be modified in flight and passed between Persistence, Business and Presentation layers.

Persistence Layer:

classes that main purpose is to access, manipulate, save, and convert the data from the Data layer into DTO

Data Layer:

consists of number of “.json” objects stored within the device flash memory, these consist of the gaits, actuator and config data.

5.1.1 Index.html

The index.html is main element of Web UI and following key code sections are described below.

- **Lines 4 -13;** <head> section links the document with required “.css” and “.js” files as well as defines the meta data and title of the page.
- **Lines 15 – 193;** <body> section, inside <div class=“grid-container”> that defines the container to host the grid-layout structure with aligned grid items.
- **Lines 19 – 40** (Figure 24.); <div class =“grid-item”>, is first of the four leg-tables grid-item elements and contains few sections, leg-header (lines 21-28) in which the header name as well as inputs for X, Y and Z values can be inserted. Lines 29 – 39 consists for <table> element with header values for id, joint, voltage, temperature, position and the table body in line 38 that is auto populate by JS / AJAX script based on the data from incoming “actuator.json” file.

```
19 <!-- LEG 1 TABLE -->
20 <div class="grid-item" id="leg-1-card">
21   <div class="leg-header">&#9713; 1 FRONT-RIGHT
22     <div class="direction_group">
23       <span class="dir_input">X:<input type="text" id="x-data-1" placeholder="X" value="75"></span>
24       <span class="dir_input">Y:<input type="text" id="y-data-1" placeholder="Y" value="75"></span>
25       <span class="dir_input">Z:<input type="text" id="z-data-1" placeholder="Z" value="50"></span>
26       <span class="dir_input"><button class="dir_btn" onclick="setPos(1)">&#10144;</button></span>
27     </div>
28   </div>
29   <table>
30     <thead>
31       <th>ID</th>
32       <th>JOINT</th>
33       <th>VOLTAGE</th>
34       <th>TEMPERATURE</th>
35       <th>POSITION</th>
36       <th>#</th>
37     </thead>
38     <tbody id="leg_1_table"></tbody>
39   </table>
40 </div>
```

Figure 25: Index.html - leg table

- **Lines 111-148;** define Mode Selector (MS) console with vertical slider for height using “rowspan=‘4’” attribute to allow to fit from the table left side (line 114). Also four table rows are created with each containing nested label with input checkbox that utilises custom .css class=“slider round” allowing for animated toggle switch (lines 123- 126) (Figure 25.)


```

111 <!-- MODE SELECTOR -->
112 <div class="grid-item" id="selector_table">
113   <table id="selector_table">
114     <tr>
115       <td rowspan="4"><input type="range" class="form-control-range" id="height_slider" oninput="setHeight()" min="0" max="100" value="0"></input>
116       </td>
117       <td>Height (mm)</td>
118       <td class="align-middle" id="height_value"></td>
119     </tr>
120     <tr>
121       <td>Directional</td>
122       <td>
123         <label class="switch">
124           <input id="directional_mode" type="checkbox" mode="directional">
125           <div class="slider round"></div>
126         </label>
127       </td>
128     </tr>
129     <tr>
130       <td>Rotational</td>
131       <td>
132         <label class="switch">
133           <input id="rotational_mode" type="checkbox" mode="rotational">
134           <div class="slider round"></div>
135         </label>
136       </td>
137     </tr>
138     <tr>
139       <td>Stair Climbing</td>
140       <td>
141         <label class="switch">
142           <input id="stairclimbing_mode" type="checkbox" mode="stairclimbing">
143           <div class="slider round"></div>
144         </label>
145       </td>
146     </tr>
147   </table>
148 </div>

```

Figure 26: Index.html - mode selector and height adjustment

- **Lines 150 – 154;** are just holder elements for left and right controller.
- **Lines 158 – 188;** define the Actuator Settings Modal (Figure 26.) that appears when the actuator is selected, its content is dynamically generated by JavaScript base on the selection of the actuator and it is using Bootstrap functionality with attributes; “data-toggle”, “data-target” and “data-dismiss”. The Modal contains header with option to close it (line 165), body with three rows for inputs against Minimal Position, Maximum Position and Position Slider (lines 167-182) and footer with buttons to save or cancel changes (lines 183- 187).

```

157 <!-- MODAL -->
158 <div class="modal fade" id="configWindow" role="dialog">
159   <div class="modal-dialog">
160     <!-- Modal content-->
161     <div class="modal-content">
162       <div class="modal-header">
163         <h4 class="modal-title">Modal Header</h4>
164         <button type="button" class="close" data-dismiss="modal">&times;</button>
165       </div>
166       <div class="modal-body">
167         <div class="form-group">
168           <div class="row" style="margin: 10px;">
169             <div class="col-7"><label>Minimum Position Limit:</label></div>
170             <div class="col-5" class="input-values" id="config_min_pos"></div>
171           </div>
172           <div class="row" style="margin: 10px;">
173             <div class="col-7"><label>Maximum Position Limit:</label></div>
174             <div class="col-5" class="input-values" id="config_max_pos"></div>
175           </div>
176           <div class="row" style="margin: 10px;">
177             <div class="col-7"><label>Current Position:</label></div>
178             <div class="col-5" class="input-slider" id="current_pos"></div>
179           </div>
180         </div>
181       </div>
182       <div class="modal-footer">
183         <button type="button" class="btn btn-default" data-dismiss="modal">Cancel</button>
184         <button type="button" id="save_button" class="btn btn-default" data-dismiss="modal">Save</button>
185       </div>
186     </div>
187   </div>
188 </div>

```

Figure 27: Index.html - actuator settings modal

- **Lines 192-192;** contain links to additional JavaScript libraries.

[Full code snippet of “index.html” is available in Appendix 4:]

5.1.2. Script.js

Script.js is the main JavaScript code section that is responsible for all UI functionality such as table data generation, modal behaviour, controllers, inputs processing, AJAX and XMLHttpRequests handling. It mixes both JS and jQuery functionalities to handle most of the code.

Following key code sections are described in detail below:

- **Lines 1-8;** define global variables that are being utilised throughout the code.
- **Lines 10-21;** defines the initiating actions when page starts which include triggering buildActuatorTable() function every 200 milliseconds with setInterval() function (lines 11-13) as well as control the operation of the mode switches in MS console (lines 17-20 Figure 27.).

```
10 $(document).ready(function() {
11     var refresh = setInterval(function() {
12         buildActuatorTable();
13     }, 200);
14
15     setHeight();
16
17     $('input:checkbox').click(function() {
18         mode = $(this).attr("mode");
19         $('input:checkbox').not(this).prop('checked', false);
20     });
21 });
```

Figure 28: Script.js - initiating actions

- **Lines 24-86;** invoke buildActuatorTable() function, this is using AJAX call to fetch data from served "actuator.json" file that is being constantly updated. The data value is being assigned to array of actuators on line 27 and followed by iteration through lists of the elements and through set of "if" statements to filter the actuators based on its leg assignment. The data for each actuator are being formatted and put into table string variable with critical actuator data such as id, joint, temperature, voltage position etc. (Figure 28.). Finally, the strings for table data are being assigned to table bodies for each leg (lines 80-84).

```
24 function buildActuatorTable() {
25     $.getJSON("data/actuator.json", function(data) {
26
27         actuators = data;
28         var leg_1_table_data = "";
29         var leg_2_table_data = "";
30         var leg_3_table_data = "";
31         var leg_4_table_data = "";
32
33
34         $.each(data, function(key, value) {
35             if (value.leg === "1") {
36                 leg_1_table_data += "<tr>";
37                 leg_1_table_data += "<td>" + value.id + "</td>";
38                 leg_1_table_data += "<td>" + value.joint_name + "</td>";
39                 leg_1_table_data += "<td>" + value.voltage + " V</td>";
40                 leg_1_table_data += "<td>" + value.temperature + " &#8451;</td>";
41                 leg_1_table_data += "<td>" + value.position + " &#176;</td>";
42                 leg_1_table_data += "<td><button class='btn' id='settings_" + value.id + "' da";
43                 leg_1_table_data += "</td></tr>";
44             }
45         });
46     });
47 }
```

Figure 29: Script.js - buildActuatorTable()

- **Lines 89-102;** jQuery script that triggers modal when the setting button is being clicked next to the listed actuator, “id” attribute is being retrieved and used to fetch the details from array of actuators. On lines 97 to 101, the sections of the modal are being populated with Max, Min position values and slider for position movement (Figure 29.).

```

88 //MODAL JS
89 $('#configWindow').on('show.bs.modal', function(event) {
90     var button = $(event.relatedTarget);
91     var id = button.data('id');
92     var modal = $(this);
93
94     var max_pos = actuators[id - 1].max_position;
95     var min_pos = actuators[id - 1].min_position;
96
97     modal.find('.modal-title').text("Actuator Config: " + id);
98     modal.find('#config_min_pos').html("<div class='input-group'><div class='input
99     modal.find('#config_max_pos').html("<div class='input-group'><div class='input
100     modal.find('#current_pos').html("<input type='range' class='form-control-range
101     modal.find('#save_button').attr("onclick", "saveConfig(" + id + ")");
102 })
103

```

Figure 30: Script.js - populating modal

- **Lines 104-149;** define additional modal functions for increasing and decreasing Max and Min position limits within the modal as well as the range of position slider with “adjustSliderRange(id)” function.
- **Lines 151-155;** define *move(id)* function used to change position of the selected actuator, and xmlhttp, post request is being build on line 153 that takes adhocURL, actuator ID and desired position value and it is being issued to the server for execution (Figure 30.).
- **Lines 157-162;** define *saveConfig(id)* function that allows to issue another xmlhttp request to save the Max and Min values specified within the modal (Figure 30.).

```

151 function move(id) {
152     var pos = document.getElementById("slider_" + id);
153     xmlhttp.open('POST', adhocURL + "move?id=" + id + "&pos=" + pos.value);
154     xmlhttp.send();
155 }
156
157 function saveConfig(id) {
158     var min_pos = parseInt(document.getElementById('min_pos_' + id).value, 10);
159     var max_pos = parseInt(document.getElementById('max_pos_' + id).value, 10);
160     xmlhttp.open('POST', adhocURL + "save?id=" + id + "&min_pos=" + min_pos + "&ma
161     xmlhttp.send();
162 }

```

Figure 31: Script.js - move() and saveConfig()

- **Lines 165-179;** define the left and right controllers using nipple.js library and by assigning them to containers within index.html file. In addition, size and position, color and mode is being defined.
- **Lines 181-2017;** define the “start”, “move” and “end” events for both controllers. Base on the user inputs, such as event type, vector and angle of the deflection is being obtained and converted into roll, pitch and direction with getRoll() (lines 226-231), getPitch() (lines 233-238) and getDirection() (lines 219-224) functions respectively. The values are then issued to server over xmlhttp post request.

```

200 joystickR.on('start', function(evt, data) {
201     r_event = evt.type;
202 }).on('end', function(evt, data) {
203     r_event = evt.type;
204     r_distance = 0;
205     r_direction = "idle";
206     xhttp.open('POST', adhocURL + "gait?direction=" + r_direction + "&mode=" + mode);
207     xhttp.send();
208 }).on('move', function(evt, data) {
209     r_event = evt.type;
210     r_distance = Math.round(data.distance);
211     r_angle = Math.round(data.angle.degree);
212     r_direction = getDirection(r_angle);
213     if (r_distance > 50) {
214         xhttp.open('POST', adhocURL + "gait?direction=" + r_direction + "&mode=" + mode);
215         xhttp.send();
216     }
217 });

```

Figure 32: Script.js - right controller event handling

- **Lines 241-249;** define setPos() function that based on the X,Y,Z values will issue another xhttp request to execute manual leg movement to desired coordinates from within the leg table.
- **Lines 264-270;** define the setHeight() function that is being triggered when the height of the elevation in MS console is being adjusted, it also uses xhttp request to issue change to Z value offset.

[Full code snippet of “script.js” is available in Appendix 5:]

5.1.3. Style.css

Style.css is responsible for layout, visual appearance, and behaviours of the elements and is utilising CSS Grid Layout Module that allows to arrange the UI’s element into rows and columns making it easier to design and more robust for changing screen sizes.

- **Lines 1-5;** define default formatting for the document by setting margin and padding values to 0;
- **Lines 9-13;** define the “grid-container” setting the “display” attribute to “grid” and defining layout with six columns and three rows (Figure 32.)
- **Lines 15-21;** define the “grid-item” objects with default padding and font size (Figure 32.).

```

8  /* LAYOUT */
9  .grid-container {
10     display: grid;
11     grid-template-columns: auto auto auto auto auto auto;
12     grid-template-rows: auto auto auto;
13 }
14
15 .grid-item {
16     background-color: rgba(255, 255, 255, 0.8);
17     opacity: 0.8;
18     text-align: center;
19     padding: 0.3em 0.5em;
20     font-size: 30px;
21 }

```

Figure 33: Style.css -CSS Grid Layout Model

- **Lines 23-51;** assign position of each UI's element within the grid to their respective column and row values (Figure 33.)

```

23  #leg-1-card {
24      grid-column: 4 / span 3;
25      grid-row: 1;
26  }
27
28  #leg-2-card {
29      grid-column: 4 / span 3;
30      grid-row: 2;
31  }
32
33  #leg-3-card {
34      grid-column: 1 / span 3;
35      grid-row: 2;
36  }

```

Figure 34: Style.css - layout arrangement

- **Lines 55-71;** define the placement and formatting of Mode Selector (MS) table (Figure 34.), line 62 is using “webkit” extension to change height slider from horizontal into vertical orientation.

```

54  /* CENTRAL TABLE */
55  #selector_table {
56      font-family: Consolas;
57      grid-column: 3/span 2;
58      grid-row: 3;
59  }
60
61  #height_slider {
62      -webkit-appearance: slider-vertical;
63      width: 50px;
64      height: 150px;
65  }

```

Figure 35: Style.cc - mode selector (MS) formatting

- **Lines 77-136;** define the formatting of the tables including the headers, rows, columns, and X,Y,Z inputs.
- **Lines 143-162;** define modal formatting mainly by setting the font type, size and aligning the element. Majority of the modal formatting is being handled by Bootstrap styling sheet.
- **Lines 169-213;** define formatting and transition effect of the mode switches located within the MS console of the UI, these are designed based on the Maangatech²⁶ article.

[Full code snippet of “style.css” is available in Appendix 6:]

²⁶ <https://maangatech.com/how-to-create-a-toggle-switch-with-html-and-css/>

5.2 Business Layer

5.2.1.1 Main.cpp

Main.cpp is the core class of the Business logic and it's derived from Arduino library. Same as Arduino ".ino" code, the main class consists of two methods, setup() and loop() that must be present within the code in order to ensure compatibility with Arduino, Platform.IO frameworks and allow for the ESP32 microcontroller to be fully programmable from Visual Studio IDE.

Purpose of the main.cpp is to initiate all communication protocols and required services, handle the web requests, routing, and manage global variables. This class is similar in nature to Controller and Services classes within an MVC framework.

- **Lines 1-6;** importing core libraries from Arduino and Espressif for compatibility.
- **Lines 7-11;** importing the custom, local libraries for handling the actuator controls and access to the Presentation layer.
- **Lines 13-15;** defining the pinouts to handle Serial2 port that will be used for RS485 communication.
- **Lines 22-30;** define constant variables for gait controls.
- **Lines 32-45;** setting up the necessary variable and assigning initial values, these will be re-used throughout the main.cpp executions and passed as an argument to further classes.
- **Lines 47-54;** we initiate setup method by setting the Serial_1 and Serial_2 connections.
- **Lines 56-62;** initiating SPI Flash (SPIFFS) memory allowing for accessing our program and data files
- **Lines 64-89;** we initiate WiFi services, by retrieving credentials stored within "credentials.json" such as SSID and Password, connecting to existing network or setting its own WiFi network and initiating DNS services to allow access via predefined URL (i.e. <http://grp.local>).

```

64 //Getting Network Credentials
65 String _ssid = JsonHandler::getJSONElement("ssid", "credentials.json");
66 String _password = JsonHandler::getJSONElement("password", "credentials.json");
67 const char *_ssid = _ssid.c_str();
68 const char *_password = _password.c_str();
69 delay(100);
70
71 //Initiating WIFI
72 WiFi.begin(ssid, password);
73 Serial.print("Connecting to WiFi");
74 while (WiFi.status() != WL_CONNECTED)
75 {
76     delay(100);
77     Serial.print(".");
78 }
79 Serial.println(" ");
80 Serial.println(WiFi.localIP());
81 delay(100);

```

Figure 36 Main.cpp - accessing the credentials and setting WiFi

- **Lines 92-105**- retrieving the actuator data from “actuator.json” file into a vectors for all, and each leg actuators.
- **Lines 108-134**; initiating WebServer by mapping all UI files within SPIFFS memory such as “index.html”, “script.js”, “bootstrap.min.css” etc.

```

107 //Starting Server
108 server.on("/", HTTP_GET, [] (AsyncWebServerRequest *request) {
109     request->send(SPIFFS, "/index.html", "text/html");
110 });
111
112 server.on("/js/popper.min.js", HTTP_GET, [] (AsyncWebServerRequest *request) {
113     request->send(SPIFFS, "/js/popper.min.js", "text/javascript");
114 });
115
116 server.on("/js/bootstrap.min.js", HTTP_GET, [] (AsyncWebServerRequest *request) {
117     request->send(SPIFFS, "/js/bootstrap.min.js", "text/javascript");
118 });

```

Figure 37: Main.cpp - mapping UI with SPIFFS files

- **Lines 136-138**; initiates and handles Ajax dynamic changes to the actuators table by passing the vector of actuators into “JsonHandler” class that updates the values and returns to the server. This is being performed every time the call comes from UI and it happens every 200ms.
- **Lines 140-154**; handle the incoming “/save?id=5&min_pos=75&max_pos=100” requests, these are coming from UI’s config modal when user adjust the Min and Max position values from the query string are being updated against vector<actuator> variable and sent directly to SR518.cpp (Hub) for updating servo registers as well as “JsonHanlder.cpp” for saving in the “actuator.json” file within SPIFFS memory.

```

140 server.on("/save", HTTP_POST, [] (AsyncWebServerRequest *request) {
141     AsyncWebParameter *p_id = request->getParam("id");
142     AsyncWebParameter *p_min_pos = request->getParam("min_pos");
143     AsyncWebParameter *p_max_pos = request->getParam("max_pos");
144     request->send(200);
145
146     int id = p_id->value().toInt();
147     int min_pos = p_min_pos->value().toInt();
148     int max_pos = p_max_pos->value().toInt();
149     actuators[id - 1].setMinPosition(min_pos);
150     actuators[id - 1].setMaxPosition(max_pos);
151     Hub.setPosLimitMin(id, min_pos);
152     Hub.setPosLimitMax(id, max_pos);
153     JsonHandler::saveActuators(actuators);
154 });

```

Figure 38: Main.cpp – updating actuator config

- **Lines 156-163;** executing incoming “/move?id=2&pos=200” requests from UI’s config modal, that is passed directly to Hub.
- **Lines 165-188;** executing incoming “/pos?id=1&x=75&y=125&z=54” requests from UI’s leg table’s header, base on the “id” value the the actuators related to same leg_Id are being pushed into separate vector and moved to “GaitHandler” class for processing and execution.
- **Lines 190-262;** executing incoming “/gait?direction=forward&mode=rotational” requests for UI’s directional controller as well Mode Selector (MS) console. Base on the “direction” and “mode” parameters, the appropriate gait values are being assigned to the gait, global variable.

```

190 server.on("/gait", HTTP_POST, [] (AsyncWebServerRequest *request) {
191     request->send(200);
192
193     AsyncWebParameter *p_direction = request->getParam("direction");
194     AsyncWebParameter *p_mode = request->getParam("mode");
195     direction = p_direction->value();
196     mode = p_mode->value();
197
198     if (direction == "idle")
199     {
200         gait = IDLE_GAIT;
201     }
202     else if (direction == "up")
203     {
204         if (mode == "directional")

```

Figure 39: Main.cpp - handling gait and mode selector

- **Lines 264-274;** executing incoming “/orientation?roll=-25&pitch=17&height=52” requests from UI’s left controller and elevation slider, it also assigns the values into respective global variables.
- **Line 276;** initiates the Web Server as per define routing values above.
- **Lines 280-285;** Executes loop() method by constantly executing gait with defined global variables while mode values is not set to “manual”.


```

279 void loop()
280 {
281     if (mode != "manual")
282     {
283         GaitHandler::ExecuteGait(actuators, gait, height, pitch, roll);
284     }
285 }

```

Figure 40: Main.cpp - loop()

[Full code snippet of “main.cpp” is available in Appendix 7:]

5.2.2 GaitHandler.h / GaitHandler.cpp

“GaitHandler.cpp” is a static class that purpose is to receive inputs from main.cpp, fetch required gait data from stored “.json” files within the SPIFFS memory in format of vector of step sequences or “vector<vector<Step>>”. Adjust the data the step data based on the pasted inputs such as height, roll and pitch and with help of “InverseKinematics.cpp” (IK) class, process the inputs into angular values for each actuator that can be transmitted into “SR518.cpp” (Hub) class.

“GaiHandler.cpp” consists of only three methods.

GaitHandler::ExecuteGait() – this is a public method that is invoked from the main.cpp and takes as input arguments vector of actuators, string value of requested gait and integer values for pitch, roll and height offsets.

- **Lines 62-100;** define the local variables and sets initial values for x, y, z, coxa, femur, tibia and type for each of the legs.
- **Line 102;** a gait as vector<vector<Step>> is being retrieved with usage of the “JsonHandler.cpp” class.
- **Lines 104-119;** vector<Actuator> is being split into individual vectors for each leg;
- **Lines 121-173;** nested iteration process begins to filter through each step in each sequence and base on provided leg_Id, values x,y,z and type are being assigned to each leg. In addition an “if” statements are at each step of iteration (i.e. lines 144-147) to pick the type value of 2 that indicates the step requires execution of lift movement that is forward into “LiftLeg()” method. If the type is equal 1, the step expects a simple shift which must be accomplished in sync movement of all actuators.

```

120
121     for (vector<Step> sequence : gait)
122     {
123         for (Step step : sequence)
124         {
125             if (step.getLegId() == 1)
126             {
127                 x_1 = step.getX();
128                 y_1 = step.getY();
129                 z_1 = step.getZ();
130                 type_1 = step.getType();
131
132                 if (type_1 == 2)
133                 {
134                     LiftLeg(leg_1_actuators, z_offset, x_1, y_1, z_1 + pitch + roll);
135                 }
136             }
137             else if (step.getLegId() == 2)

```

Figure 41: GaitHandler.cpp - Sequence Iteration

- **Line 174;** executes the ShiftLeg() method for all legs coordinates.

GaitHandler::LiftLeg() – executes a lift operation that consists of three phases, lifting, transfer and put-down.

- **Lines 7-15;** split the vector<Actuator> (leg Actuators) to determine the ID's of coxa, femur and tibia servos.
- **Lines 17-23;** lift phase, the x,y,z values are being translated with IK class into angular values for each joint in order to achieve changes to femur and tibia actuators but keep coxa joint in same place.

```

17     //Lift-UP Leg
18     double femur_angle_up = IK.getFemurAngle(x, y, 1);
19     double tibia_angle_up = IK.getTibiaAngle(x, y, 1);
20     double coxa_angle_current = Hub.getPosition(coxaId);
21
22     Hub.moveLeg(leg_actuators, coxa_angle_current, femur_angle_up, tibia_angle_up);
23     delay(TIMEOUT);
24
25     //Transfer Leg
26     double coxa_angle_down = IK.getCoxaAngle(x, y);
27     Hub.moveLeg(leg_actuators, coxa_angle_down, femur_angle_up, tibia_angle_up);
28     delay(TIMEOUT);
29
30     // Put Down
31     double femur_angle_down = IK.getFemurAngle(x, y, z + z_offset);
32     double tibia_angle_down = IK.getTibiaAngle(x, y, z + z_offset);
33     Hub.moveLeg(leg_actuators, coxa_angle_down, femur_angle_down, tibia_angle_down);
34

```

Figure 42: GaitHandler.cpp - LiftLeg()

- **Lines 26-28;** transfer phase, coxa joint moves to desired final position while femur and tibia remain in same elevated position.
- **Lines 33-34;** put-down phase, femur and tibia joint being moved to desired final position while coxa remains in same position.

GaitHandler::ShiftLeg() – shifts all the legs simultaneously to desired x,y,z coordinates of each leg;

- **Lines 38-41;** adjusting the Z values based on inputs such as pitch, roll, height (z_offset).
- **Lines 43-57;** using IK to convert the coordinate values into angular values for each joint.

- **Line 59;** forwarding the values into Hub class to execute synchronous shift movement.

```

38     int z_1_adj = z_1 + z_offset + pitch + roll;
39     int z_2_adj = z_2 + z_offset - pitch + roll;
40     int z_3_adj = z_3 + z_offset - pitch - roll;
41     int z_4_adj = z_4 + z_offset + pitch - roll;
42
43     double coxa_1_angle = IK.getCoxaAngle(x_1, y_1);
44     double femur_1_angle = IK.getFemurAngle(x_1, y_1, z_1_adj);
45     double tibia_1_angle = IK.getTibiaAngle(x_1, y_1, z_1_adj);
46

```

Figure 43: GaitHandler.cpp - calculating offsets

[Full code snippet of “GaitHandler.cpp / .h” is available in Appendix 8:]

5.2.3. InverseKinematics.cpp / InverseKinematics.h (IK)

InverseKinematics.cpp class is one of the smallest but at the same time the most mathematically complex within the entire project and its purpose is to process X,Y,Z values for each of the leg into angular values for Coxa, Femur and Tibia joint.

In order to obtain required Coxa, Femur and Tibia angles, number of trigonometric equations have to be solved based on provided X,Y,Z coordinates as well as length of Coxa, Femur, and Tibia structural parts. It can also be invoked with “IK” prefix, allowing it for quicker access.

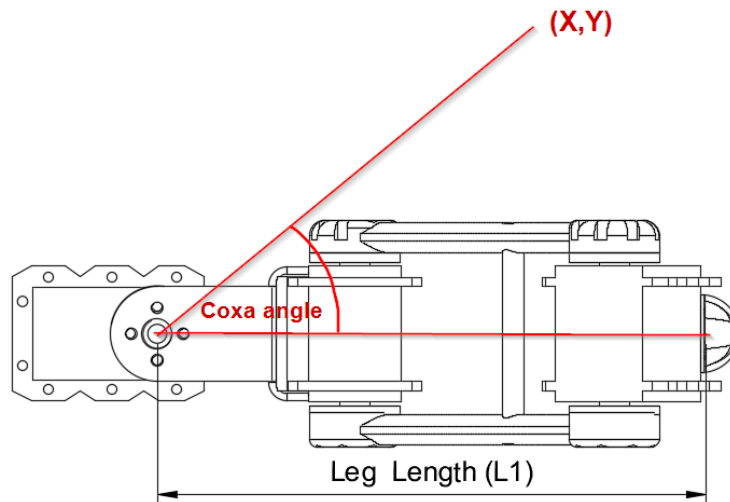


Figure 44: IK - leg top view

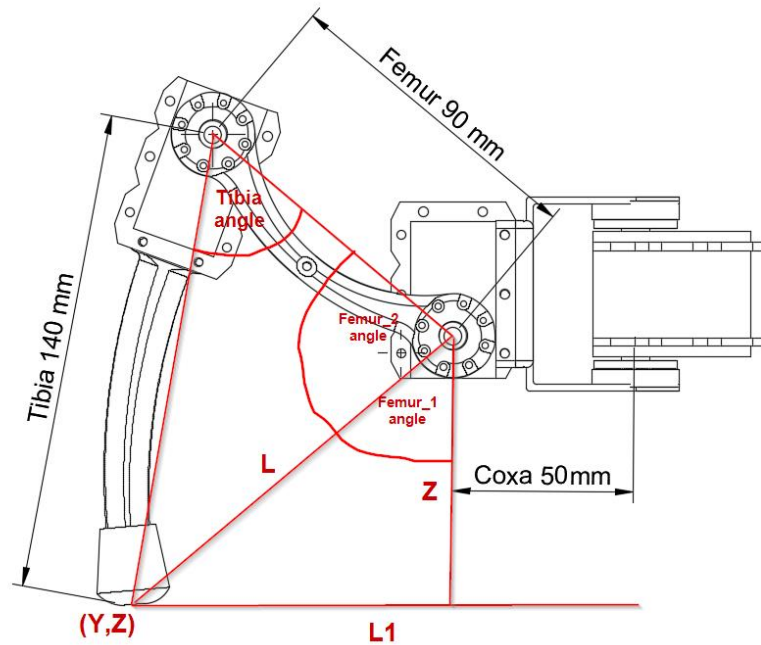


Figure 45: IK - leg side view

As per Figures 44 and 46, we can see that a trigonometric equations for triangle and cosine rules can be applied in order to obtain required angles in following steps:

1. Calculating Coxa Angle with given X, and Y coordinates (Figure. 44) can be accomplished by applying negative cotangent.

$$\frac{x}{y} = \tan(\text{Coxa angle})$$

$$\text{Coxa angle} = \tan^{-1}\left(\frac{x}{y}\right)$$

This is being handled identically in lines 7-14 (Figure. 46) by "getCoxaAngle()" method with only difference for in line 12 where conversion form radians to degrees is completed.

```

7  double InverseKinematics::getCoxaAngle(int _x, int _y)
8  {
9      double x = (double)_x;
10     double y = (double)_y;
11     double coxa_angle = atan(x / y);
12     double result = coxa_angle * 180 / PI;
13     return result;
14 }
```

Figure 46: InverseKinematics.cpp - getCoxaAngle()

2. Calculating Femur angle can be accomplished by splitting the angle into Femur1 and Femur2 and with given X and Z values

- We can calculate distance of L by applying right triangle formula

$$L^2 = (L1 - Coxa Length)^2 + Z^2$$

$$L = \sqrt{(L1 - Coxa Length)^2 + Z^2}$$

- We can now Calculate Femur_1 using negative Cosine.

$$Femur1\ angle = \cos^{-1}\left(\frac{Z}{L}\right)$$

- For Femur2 angle we can apply Cosine Rule for triangle Tibia, Femur, L

$$Tibia\ Length^2 = Femur\ Length^2 + L^2 - 2(Femur\ Length)(L)\cos(Femur2\ angle)$$

$$Femur2\ angle = \cos^{-1}\frac{Tibia\ Length^2 - Femur\ Length^2 - L^2}{-2(Femur\ Length)(L)}$$

- After adding Femur1 and Femur2 equations we should have a working value of Femur angle

$$Femur\ angle = \cos^{-1}\left(\frac{Z}{L}\right) + \cos^{-1}\frac{Tibia\ Length^2 - Femur\ Length^2 - L^2}{-2(Femur\ Length)(L)}$$

These calculations are performed in "getFemurAngle()" method in lines 16-29.

```

16 double InverseKinematics::getFemurAngle(int _x, int _y, int _z)
17 {
18     double x = (double)_x;
19     double y = (double)_y;
20     double z = (double)_z;
21     double l1 = sqrt(pow(x, 2) + pow(y, 2));
22     double l = sqrt(pow(z, 2) + pow(l1 - COXA_LENGTH, 2));
23     double femur_angle_1 = acos(z / l);
24     double femur_angle_2 = acos((pow(TIBIA_LENGTH, 2) - pow(FEMUR_LENGTH, 2) - pow(l, 2)
25     femur_angle = femur_angle_1 + femur_angle_2;
26     double result = femur_angle * 180 / PI;
27     return result;
28 }

```

Figure 47: InverseKinematics.cpp - getFemurAngle()

3. We can now calculate Tibia angle using Cosine Rule again for triangle Tibia, Femur, L

$$L^2 = Tibia\ Length^2 + Femur\ Length^2 - 2(Tibia\ Length)(Femur\ Length)\cos(Tibia\ angle)$$

$$Tibia\ angle = \cos^{-1}\frac{L^2 - Tibia\ Length^2 - Femur\ Length^2}{-2(Femur\ Length)}$$

These calculations are performed in “getTibiaAngle()” method in lines 16-29.

```

30 double InverseKinematics::getTibiaAngle(int _x, int _y, int _z)
31 {
32     double x = (double)_x;
33     double y = (double)_y;
34     double z = (double)_z;
35     double l1 = sqrt(pow(x, 2) + pow(y, 2));
36     double l = sqrt(pow(z, 2) + pow(l1 - COXA_LENGTH, 2));
37     double tibia_angle = acos((pow(l, 2) - pow(TIBIA_LENGTH, 2) - pow(FEMUR_LENGTH, 2)) /
38     double result = tibia_angle * 180 / PI;
39     return result;
40 }

```

Figure 48: InverseKineamtics.cpp - getTibiaAngle()

[Full code snippet of “InverseKinematics.cpp / .h” is available in Appendix 9:]

5.2.4 SR518.h / SR518.cpp (Hub)

SR518.cpp class is responsible for handling communication directly with SR518 actuators and compiles the angular values for Coxa, Femur and Tibia into data packets that can be transmitted using RS485 serial protocol. SR518.cpp can also receive packets directly from the actuators allowing for real time monitoring and feedback.

The code has been based on the SR518²⁷ datasheet by Spring RC as well as DynamixelSerial²⁸ library by Savage Electronics.

Table. present typical structure of the packets with byte values.

Control table for SR518 with list of programable registers within RAM and EEPROM memory can be accessed in Appendix 2.

Instruction Packet								
0xFF	0xFF	ID	Length	Instruction	Parameter_1	...	Parameter_N	Check Sum
Status Packet								
0xFF	0xFF	ID	Length	Error	Parameter_1	...	Parameter_N	Check Sum

Table 12: Instruction and Status Packet Structure

- **0xFF 0xFF** – flag that indicats beginning of the packet
- **ID** – id of the receiving actuator 0x00-0xFD (0xFD / 254 is the broadcast ID)

²⁷ <http://www.springrc.com/en/pd.jsp?id=87>

²⁸ <https://savageelectronics.com/category/dynamixel/>

- **Length** – length of the packet, this can be calculated using $\text{Parameters}(N) + 2$
 - **Instruction Code** – there are seven supported instruction types, however the SR518.cpp is only using READ DATA (0x02), WRITE DATA (0x03) and SYNC WRITE (0x83)
 - **Parameter_1 – N** – specify additional attributes based on the control tables and input values. i.e. Position, Speed, Torque etc. (see Appendix 2)
 - **Check Sum** – low byte of sum of the packet data values $\sim(\text{id} + \text{Length} + \text{Instruction} + \text{Parameter}_1 + \dots + \text{Parameter}_N)$
- **Lines 6-46 / .h;** used to define values based on the Control Tables and supported instruction set so these can be re-used during the packet creation.
 - **Lines 55-73 / .h;** define the public methods for Serial connection setup, movement, configure changes and data feedback.
 - **Lines 74-89 / .h;** define private variables as well as private read_error() method.
 - **Lines 8-31 / .cpp;** provide implementation of the read_error() method that checks for timeouts and retrieves the error code from returned status packet.
 - **Lines 33-43 / .cpp;** provide implementation of setSerial() and begin() methods that instantiate instance of hardware serial port with assigned baud-rate value as well as directional pin value that switches between RX and TX operation modes.
 - **Lines 45-65 / .cpp;** provide implementation of move() method that allows to move an actuator with specified ID to desired position. In order to complete it few steps needs to be completed:
 - Line 47; angular value is being converted into integer by multiplying by resolution converter.
 - Line 48 & 49; position value is being split into position high_byte and low_byte that will be used as parameters.
 - Line 50; checksum is being calculated.
 - Line 52; directional pin is set to High level to enable transmission (sets MAX485 IC into transmission mode)
 - Lines 53-61; Instruction packet is being transmitted byte by byte starting with notification flag (0xFF 0xFF), followed by ID of the destination actuator, Position Length value that is defined in control tables, WRITE instruction value, GOAL_POSITION_L (value from control table) low and high position bytes and ended with checksum value.
 - Line 63; directional pin is set to low enabling receiving of the Status Packet.
 - Line 64; return value of the Status Pack, in case of failed transmission an error bit will be returned.

```

45 int SR518::move(byte id, double position_angle)
46 {
47     int position = position_angle * POSITION_RESOLUTION_H;
48     char pos_h = position >> 8;
49     char pos_l = position & 255;
50     byte checksum = ~lowByte(id + POSITION_LENGTH + WRITE + GOAL_POSITION_L + pos_l + pos_h);
51
52     digitalWrite(Direction_Pin, TX_MODE);
53     serialPort->write(START_FLAG);
54     serialPort->write(START_FLAG);
55     serialPort->write(id);
56     serialPort->write(POSITION_LENGTH); //length
57     serialPort->write(WRITE); //Instruction
58     serialPort->write(GOAL_POSITION_L); //First Address of data write
59     serialPort->write(pos_l); //First data write
60     serialPort->write(pos_h); //Second data write
61     serialPort->write(checksum);
62     serialPort->flush();
63     digitalWrite(Direction_Pin, RX_MODE);
64     return (read_error());
65 }

```

Figure 49: SR518.cpp - move()

- **Lines 67-157 / .cpp;** moveLeg() method is being implemented, it is extension of the move() function that allows for synchronous movement of all actuators within specified leg. The method takes as input argument vector of actuators, as well as angular values for coxa, femur and tibia that are provided from “GaitHandler.cpp”. The key difference is usage of “for loop” on line 83-105 that assigns coxa, femur and tibia correct actuator ID’s and calibrates the positions by using predefined Min and Max position values as well as by applying offsets:
 - Coxa Calibrated Position = Max Position Limit – Coxa Angle.
 - Femur Calibrated Position = Max Position Limit – (Femur Angle – 60)
 - Tibia Calibrated Position = Min Position + Tibia Angle

```

81 for (Actuator leg_servo : leg_actuators)
82 {
83     if (leg_servo.getJoint() == 1)
84     {
85         if (leg_servo.getInverted() == 1)
86         {
87             coxa_position_calibrated = leg_servo.getMaxPosition() - _coxa_pos_angle;
88         }
89         else
90         {
91             coxa_position_calibrated = _coxa_pos_angle + leg_servo.getMinPosition();
92         }
93         servo_coxa_id = leg_servo.getId();
94     }
95     if (leg_servo.getJoint() == 2)
96     {
97         femur_position_calibrated = leg_servo.getMaxPosition() - (_femur_pos_angle - 60);
98         servo_femur_id = leg_servo.getId();
99     }
100     if (leg_servo.getJoint() == 3)
101     {
102         tibia_position_calibrated = _tibia_pos_angle + leg_servo.getMinPosition();
103         servo_tibia_id = leg_servo.getId();
104     }
105 }

```

Figure 50: SR518.cpp - calibration loop

- **Lines 160-461 / .cpp;** movaAll() method is responsible for movement of all twelve actuators at the same time, it is identical to moveLeg() method.

- **Lines 463-483/.cpp;** setPosLimitMin() method, updates minimum position limit register (MIN_POSITION_L) for selected actuator, this will make sure that the actuator does not go below the value.
- **Lines 485-505/.cpp;** setPosLimitMax() method, updates maximum position limit register (MAX_POSITION_L) for selected actuator, this will make sure that the actuator does not go above the value.

```

485 int SR518::setPosLimitMax(byte id, double position_angle)
486 {
487     int position = position_angle * POSITION_RESOLUTION_H;
488     char pos_h = position >> 8;
489     char pos_l = position % 256;
490     byte checksum = ~lowByte(id + POSITION_LENGTH + WRITE + MAX_POSITION_L + pos_l + pos_h);
491
492     digitalWrite(Direction_Pin, TX_MODE);
493     serialPort->write(START_FLAG);
494     serialPort->write(START_FLAG);
495     serialPort->write(id);
496     serialPort->write(POSITION_LENGTH);
497     serialPort->write(WRITE);
498     serialPort->write(MAX_POSITION_L);
499     serialPort->write(pos_l);
500     serialPort->write(pos_h);
501     serialPort->write(checksum);
502     serialPort->flush();
503     digitalWrite(Direction_Pin, RX_MODE);
504     return (read_error());
505 }

```

Figure 51: SR518.cpp - setPosLimitMax()

- **Lines 507-544 /.cpp;** getTemperature() method, requests value of the temperature from specified actuator by addressing servo with it's ID and "PRESENT_TEMPERATURE" instruction. It is followed by Status Packet from the actuator in lines 531-541 and the temperature byte is being retrieved in line 540 and returned from the method in line 543.

```

531 while (serialPort->available() > 0)
532 {
533     if ((serialPort->read() == START_FLAG) & (serialPort->peek() == START_FLAG))
534     {
535         serialPort->read(); // Start Bytes
536         serialPort->read(); // Ax-12 ID
537         serialPort->read(); // Length
538         if ((Error_Byte = serialPort->read()) != 0) // Error
539             return (Error_Byte * (-1));
540         Temperature_Byte = serialPort->read(); // Temperature
541     }
542 }
543 return (Temperature_Byte); // Returns the read temperature
544 }

```

Figure 52: SR518.cpp getTemperature()

- **Lines 546-583/.cpp;** getVoltage() method, is identical to getTemperature() but it uses PRESENT_VOLTAGE instruction on Instruction Packet and retrieves Voltage byte from Status Packet.
- **Lines 587-626/.cpp;** getPosition() method, is identical to getTemperature() and getVoltage() but it uses PRESENT_POSITIONE instruction on Instruction Packet and retrieves Position byte from Status Packet.

[Full code snippet of "SR518.cpp / .h" is available in Appendix 10:]

5.3 Data Transfer Objects (DTO)

The purpose of DTOs is to add level of data separation between Data and Business layers but also allow for quicker updates to the objects and better transcoding between “.json” files from SPIFFS and “.json” for Presentation layer.

The DTO consists of two object classes that are usually organised in lists or vector<>

5.3.1 Actuator.h/.cpp

This class defines properties required for the operation of actuator such as ID, Joint, Voltage, Temperature, Position, Minimal and Maximal angle limits and can be accessed through set of public getters and setter methods as well as through Constructor. The actuators are typically passed into Business logic in form of “vector<Actuator>”

[Full code snippet of “actuator.cpp / .h” is available in Appendix 11:]

5.3.2 Step.h/.cpp

This class defines properties required for creating walking gaits throughout of sequences of steps. The properties include, Leg_ID, X, Y, Z coordinates and Type which indicates if the step is expected to Shift Leg (value = 1) or Lift Leg (value = 2). The steps are typically passed into Business logic in form of “vector<vector<Step>”

[Full code snippet of “step.cpp / .h” is available in Appendix 11:]

5.4 Persistence Layer

Purpose of the persistence layer is to access stored data within the Data Layer and convert these into more manipulatable Data Transfer Objects (DTO), mainly Steps and Actuators. The layer consists of two separated classes that supplement its functionality as well as 3rd party “ArduinoJson” library allowing for serialization and deserialization.

5.4.1 JsonHandler.h / JsonHandler.cpp

Its is main class within the layer, its purpose is to manage conversion process of data from object level into primitive data level such as String.

- **Lines 3-13;** getJsonElement() the class allows for retrieving of any element within any stored “.json” file within SPIFFS memory by invoking “SpiffHandler” class to retrieve object

of type File and then using “ArduinoJson” library to deserialize the element’s value from the file as a String.

```
3   String JsonHandler::getJSONElement(String element, String file_name)
4   {
5       File jsonFile = SpiFFHandler::readFile(file_name);
6       std::unique_ptr<char[]> buf(new char[SIZE]);
7       jsonFile.readBytes(buf.get(), SIZE);
8       DynamicJsonDocument doc(SIZE);
9       deserializeJson(doc, buf.get());
10
11       String value = doc[element];
12       return value;
13   }
```

Figure 53: JsonHandler.cpp - getJsonElement()

- **Lines 15-31;** getJsonActuators() similar to the getElement() method but the deserialized object is being iterated through and each element is being put into vector<Actuator> that is returned from the method.

```
21   deserializeJson(doc, buf.get());
22   JsonArray root = doc.as<JsonArray>();
23
24   vector<Actuator> actuators;
25
26   for (JsonObject item : root){
27       Actuator actuator = Actuator(item["id"], item["leg"], item["inverted"]);
28       actuators.push_back(actuator);
29   }
30   return actuators;
31 }
```

Figure 54: JsonHandler.cpp - getJsonActuators()

- **Lines 33-54;** updateActuators() this method is reverse of getJsonActuators() , it takes as the input argument vector<Actuator> and using “ArduinoJson” library serializes the data into one String value allowing for future saving.

```

33 String JsonHandler::updateActuators(vector<Actuator> actuators)
34 {
35     DynamicJsonDocument outputDoc(SIZE);
36     for (Actuator actuator : actuators)
37     {
38         int id = actuator.getId();
39         JsonObject obj = outputDoc.createNestedObject();
40         obj["id"] = String(id);
41         obj["leg"] = String(actuator.getLeg());
42         obj["inverted"] = String(actuator.getInverted());
43         obj["joint"] = String(actuator.getJoint());
44         obj["joint_name"] = String(actuator.getJointName().c_str());
45         obj["max_position"] = String(actuator.getMaxPosition());
46         obj["min_position"] = String(actuator.getMinPosition());
47         obj["temperature"] = String(Hub.getTemperature(id));
48         obj["voltage"] = String(Hub.getVoltage(id));
49         obj["position"] = String(Hub.getPosition(id));
50     }
51     String outputDocString = "";
52     serializeJson(outputDoc, outputDocString);
53     return outputDocString;
54 }

```

Figure 55: JsonHandler.cpp - updateActuators();

- **Lines 56-60;** saveActuators() is just simple function that passes the retrieved String value from updateActuators() into SpiffHandler class for saving.
- **Lines 63-101;** readGait() method is very similar to getJsonActuators() method but it uses nested iteration process in order to obtain sequences of Steps or “vector<vector<Step>>”.

```

74 for (JsonObject sequence : root)
75 {
76     vector<Step> sequence_vector;
77     int sequence_name = sequence["sequence"];
78     JsonArray steps = sequence["steps"];
79
80     for (JsonObject step : steps)
81     {
82         Step step_object = Step(step["id"], step["type"], step["x"], step["y"], step["z"]);
83         sequence_vector.push_back(step_object);
84
85         Serial.print("ID: ");
86         const char *id = step["id"];
87         Serial.print(id);
88         Serial.print(" - X: ");
89         const char *x = step["x"];
90         Serial.print(x);
91         Serial.print(" - Y: ");
92         const char *y = step["y"];
93         Serial.print(y);
94         const char *z = step["z"];
95         Serial.print(" - Z: ");
96         Serial.println(z);
97     }
98     gait_vector.push_back(sequence_vector);
99 }
100 return gait_vector;

```

Figure 56: JsonHandler.cpp - readGait()

[Full code snippet of “JsonHandler.cpp / .h” is available in Appendix 12:]

5.4.2 SpiffHandler.cpp / SpiffHandler.h

SpiffHandler.cpp is a very simple class that uses “SPIFFS.h” library by Espresiffs to access the serial flash memory located on the ESP32 microcontroller and save or read data.

- **Lines 3-7;** saveFile() this method takes as argument input String and name of the destination file, in line 4 an instance of “File” object is being opened with FILE_WRITE permissions, line 5 modifies the file with the input string and in line 6 the file is being closed. The purpose of the method is to update .json files.
- **Lines 9-13;** saveFile() overloaded version of saveFile() method that uses instead of input String a “File” object and is used when file content needs to be replaced.
- **Lines 15-19;** readFile() method similar to saveFile() method but instead of saving a file, it returns the File object from the SPIFFS memory by using FILE_READ permissions.

```
1  #include "SpiffHandler.h"
2
3  void SpiffHandler::saveFile(String output_string, String file_name){
4      File file = SPIFFS.open(SPIFF_PATH + file_name, FILE_WRITE);
5      file.print(output_string);
6      file.close();
7  }
8
9  void SpiffHandler::saveFile(File input_file, String file_name){
10     File file_opened = SPIFFS.open(SPIFF_PATH + file_name, FILE_WRITE);
11     file_opened.print(input_file);
12     file_opened.close();
13 }
14
15 File SpiffHandler::readFile(String file_name){
16     File file = SPIFFS.open(SPIFF_PATH + file_name, FILE_READ);
17     return file;
18     file.close();
19 }
```

Figure 57: SpiffHandler.cpp

[Full code snippet of “SpiffHandler.cpp / .h” is available in Appendix 13:]

5.5 Data Layer

Data layer consists of number of the “.json” files that store the configuration, actuator data and gait values and through Persistence layers can be converted into DTO objects and then moved for processing and displaying through Business and Presentation layers.

We have two types “.json” files are stored within the SPIFFS memory

5.5.1 Config files.

- **Actuator.json** – is responsible for configuration of the actuators within the system, it stores id, leg, joint, joint name, temperature, voltage, position, max and min position levels. Some of the values such as Voltage or Temperature are unassigned as these are being updated dynamically during the executions.

```
1  [
2  {
3      "id": "1",
4      "leg": "1",
5      "joint": "1",
6      "joint_name": "coxa",
7      "temperature": "0.0",
8      "voltage": "0.0",
9      "position": "150.0",
10     "max_position": "195",
11     "min_position": "105"
12 },
13 {
14     "id": "2",
15     "leg": "2",
16     "joint": "1",
17     "joint_name": "coxa",
18     "temperature": "0.0",
19     "voltage": "0.0",
20     "position": "150.0",
21     "max_position": "195",
22     "min_position": "105"
23 },
24 ]
```

Figure 58: Actuator.json

- **Credentials.json** – allows for safely storing SSID and Password values for the setup of Wi-Fi connectivity, it also provides a security level as the credentials are not hard coded and cannot be easily accessed by users within the SPIFFS memory

```
1  {
2      "ssid": "ssid",
3      "password": "password"
4  }
```

Figure 59: Credentials.json

5.5.2 Gaits

We also have series of “.json” files within SPIFFS that store the Gaits for execution during movements, these can be divided based on the operational mode:

Directional gaits - “forward_gait.json”, “back_gait.json”, “left_gait.json” and “right_gait.json”

Directional Gaits are variations of the “Creep Gait”, facilitate latter movement forward, back, left and right.

The gaits consist of six sequences that combine both lift and shift leg movements. As per Figure 60., sequences 1,2,4,5 are lift sequences during which only one leg lifts while remaining are in contact with the ground creating tripod stand. Sequences 3 and 6 are performing shift movement where all legs are moving to new coordinates while remaining in contact with the ground.

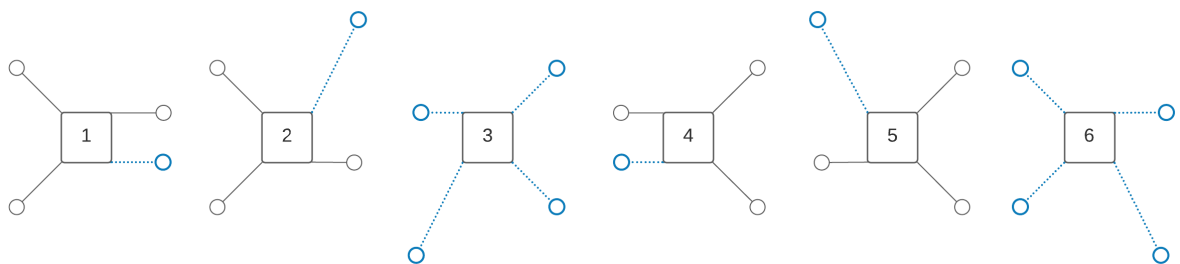


Figure 60: Creep Gait – forward

```
1  [
2  {
3      "sequence": "1",
4      "steps": [{ "id": "1", "x": "50", "y": "10", "z": "50", "type": "1"},
5                  { "id": "2", "x": "50", "y": "10", "z": "50", "type": "2"},
6                  { "id": "3", "x": "75", "y": "75", "z": "50", "type": "1"},
7                  { "id": "4", "x": "75", "y": "75", "z": "50", "type": "1"}]
8  },
9  {
10     "sequence": "2",
11     "steps": [{ "id": "1", "x": "75", "y": "150", "z": "50", "type": "2"},
12               { "id": "2", "x": "50", "y": "10", "z": "50", "type": "1"},
```

Figure 61: Forward_gait.json

Rotational gaits – “rotateLeft_gait.json” and “rotateRight_gait.json”

Rotational gaits consist of five sequences during which the body of the robot moves along its Z axis (rotates). As per Figure.62, the sequences 1 to 4, each leg lift and moves into new

coordinates one by one and at the sequence 5 all legs shift to its neutral positions completing the rotational movement.

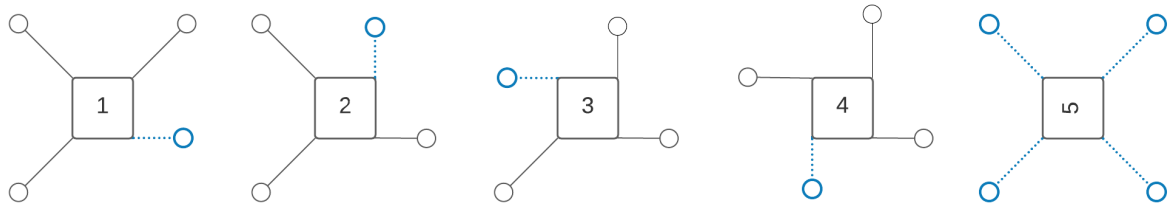


Figure 62: Rotational Gait – left

6. Project Planning

A several Software Development Life Cycle methodologies have been considered for the project:

Waterfall Model: offered initially a logical way of planning through clearly designed steps, however it was quickly dismissed as did not provide enough flexibility with adjusting the scope and requirements for the project that due to its nature was providing to be technologically challenging.

Evolutionary Prototyping Model: was another consideration as it allowed put more focus on creating semi-functional prototypes which were extensive part of the project development process, however it was also dismissed as the technological challenges could quickly overwhelm the progress and result in delays.

Eventually, it was decided that a modifies version o Agile / Scrum model will be applied to the project as it provided high level of flexibility to move between scope and the implementation stages and would still allow for organising the work in a manner that each Sprint would result in the visible progress toward the project goal.

The key difference between standard Scrum and adopted model was that due to hardware heavy nature, no functioning prototype could be delivered by the end of each sprint. This was mainly caused by following factors.

- **Procurement Lead Time** – components had to be ordered from overseas suppliers and due to outgoing pandemic, long delivery periods were expected.
- **Component Costs** – each structural element was adding to the cost of the project that had to be closely monitored to not to exceed established budget.
- **Non-Homogenous Work** – number of different skillsets and had to be used to design and complete the hardware such as parametric modelling, PCB and electronic circuit design, UI and Software design and implementation.
- **Extensive Prototyping Effort** - to avoid any accidental damage to the hardware components, more time has been devoted to this step as any errors or mistakes could result in significant cost and time overruns that would jeopardise the success of the project.

The workload has been split into six separated Sprints that consisted of both Hardware and Software components, Table. 13 displays each Sprint overview and Figure 63. represents the timelines of each Sprint completions in Gant Chart.

Sprint	Due Date	Goals	Activities
Sprint 1: Design and Assembly of the Core Unit with Battery Pack	31st December 2020	Complete main structural component with ability to power the electronics	<ol style="list-style-type: none"> 1. Designing the Core Module using Fusion 360 and parametric modelling. 2. Manufacture Core Unit using Elegoo Mars 3D Printer. 3. Build battery pack in 2S4P configuration within the Core Unit parts using spot welding tool and provide both balanced charging as well as power output connectors. 3. Attach Coxa actuators to the core and test its functionality with usage of breadboard prototype controller. 4. Create initial web-UI allowing for basic actuator control and feedback.
Sprint 2: Design and Assembly of the Electronics and UI expansion	15th January 2021	Complete the design of Controller and Power Distribution Boards and expand UI's functionality.	<ol style="list-style-type: none"> 1. Designing and testing of Power Distribution board and Controller Board and submitting an order with JLCPCB manufacturer and Mauser, electronics (component wholesaler). 2. Expanding the UI with Configuration Modal allowing for Setting Max and Min position limits as well as manual servo movement.
Sprint 3: Design and Assembly of the Legs	31st January 2021	Complete leg design and assemblies, finalise hardware portion of the project	<ol style="list-style-type: none"> 1. Design and manufacture Coxa part of the leg assembly using Fusion 360 and Elegoo Mars 2. Design and manufacture Femur parts of the leg assembly using Fusion 360 and Elegoo Mars. 3. Design and manufacture Tibia part of the leg assembly using Fusion 360 and Elegoo Mars.
Sprint 4: Initial Code Structure.	28th February 2021	Design all code layers with initial classes to perform "functionality spike"	<ol style="list-style-type: none"> 1. Initial design of Presentation layer with "Actuator Table" 2. Initial design of Domain layer with Main.cpp, GaitHandler.cpp. 3. Initial design of the Persistence layer with JsonHandler.cpp and SpiffHandler.cpp
Sprint 5: Inverse Kinematics and Gait movement	31st March 2021	Implement movement	<ol style="list-style-type: none"> 1. Create InverseKinematics.cpp library. 2. Create Directional and Rotational Gaits. 3. Extend UI with "on-screen" controls and mode selectors. 4. Implement pitch and roll controls.
Sprint 6: Final Steps and Documentation	26th April 2021	Finalise UI , movements and Documentation	<ol style="list-style-type: none"> 1. Complete Documentation. 2. Complete Custom Gait (Climbing Obstacles) 3. Finalise UI by applying styling and elevation controls.

Table 13: Sprints Break-Down

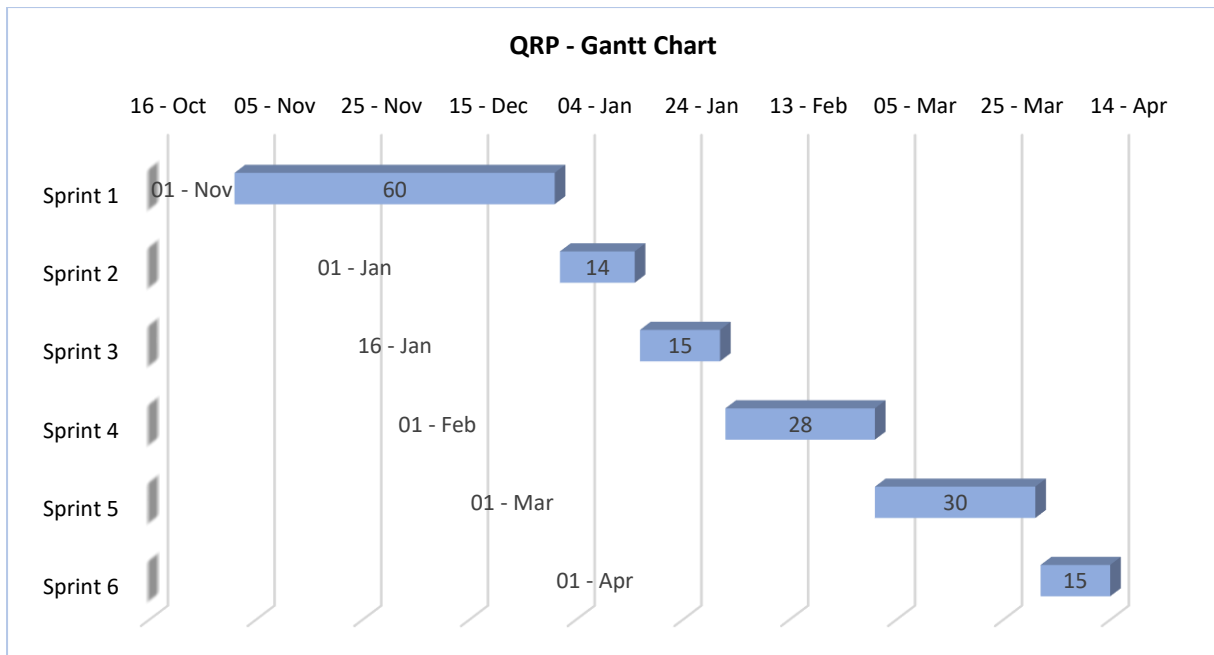


Figure 63: Gantt Chart

7. Testing and Evaluation

The project has been tested according to a set of scenarios that should address all aspects of functional and non-functional requirement defined in Chapter 2. Based on the test results, an evaluation of the requirements can be completed and provide indication on how the project performed.

7.1. Testing

Table. defines the tests scenarios with their description, expected functionality, actual functionality, issue detected, severity and additional comments.

Test Case	Description	Expected Result	Actual Result	Issue	Severity	Corrective Actions
TC_01	Password Protection	Password required before establishing connection with the device.	User had to enter password before connecting to QRP SSID	None	None	None
TC_02	Accessing Interface	User can access QRP from Wi-Fi enabled device after entering provided URL or IP	User was able to load UI by entering "http://qrp.local" or "192.168.1.119" in Web browser after establishing connection with QRP	None	None	None

TC_03	Responsive UI	All UI's element should be aligned and fully visible without need for scrolling or zooming on devices with resulting 1024x768 or greater.	User can view all the element without any issues on full HD and larger screens, in case of tablet screens, an extra refresh was required when the orientation changed.	Bug	Minor	It has been determined that this is related with usage "nipple.js" library and would require supplementing it with custom one. This would take too much effort to implement before the release therefore it was decided to not to take any corrective actions for now.
TC_04	Live Actuator Feedback	User receives live feedback of actuators data, such as voltage, temperature, position, id and location joint.	User is presented with table displaying id, joint, voltage, position and temperature, changes to voltage and position values during the movement indicates that values are properly refreshed during operation.	None	None	None
TC_05	Setting Position Limits	Pressing setting button next to the actuator within the leg table will open modal with inputs for Minimum, Maximum position limits, manual position slider and options to cancel or save configuration	User can access settings button that opens the modal with the configs, slider, and option to save or cancel the changes.	None	None	None
TC_06	Adjusting Minimum and Maximum Position Limits	Adjusting the Minimum and Maximum limits will result in adjustments to slider moving range as well as leg movement range.	Adjustments to Minimum and Maximum values correctly affected the slider and leg movement, user was also unable to exceed Minimum limit above Maximum limit value and vice-verse.	None	None	None
TC_08	Saving Minimum and Maximum Position Limits	After adjusting Minimum and Maximum limit values, user can select on "save" button that will updates the values within the system. The new values should remain after device is powered down.	User adjusted Minimum position limit from 40 degrees to 90 degrees and Maximum limit from 200 degrees to 150 degrees and saved the config. Device was switched off completely and after 30 seconds rebooted, the values remained in place and leg movement was restricted as per new values.	None	None	None
TC_09	Manual Leg Movement to Specified X, Y, Z Coordinates	User should be able to enter X,Y,Z values into any of the leg table header inputs and execute lift movement to specified coordinates.	User was able to specify the new coordinates however the leg did not move to correct positions and executed shift rather than lift movement.	Bug	Major	It was established that the issue was with Inverse Kinematics taking into calculations only Y values for reach rather than vector of X and Y. The instruction was also incorrectly mapped with shiftLeg() method instead with liftLeg() resulting in incorrect movement. Both issues were rectified and fixed before the final release.
TC_10	Switching Between Operation Modes	User should be able to select and switch between available operational modes.	User was able to engage directional mode and switch to rotational, custom and to manual without any issues.	None	None	None

TC_11	Directional Mode	User selects "Directional" mode, it should allow user to move the right controller in forward, back, left and right directions while robot executes corresponding gaits and move in same directions. When controller is released, robot should return to "idle" state.	User was able to move robot forward, left, right and back directions after pushing the right controller to corresponding positions.	None	None	None
TC_12	Rotational Mode	User selects "Rotational" mode, it should allow user to move the right controller in forward, back to execute same gaits as for "Directional" mode, but left and right directions should be substituted with rotate left and rotate right movement along its Z axis. When controller is released, robot should return to "idle" state.	User was able to move robot forward, back and execute rotation left and right, when controller was released, the robot returned to "idle" position. User noticed that rotation left is less stable, and some loss of coordination occurs.	Bug	Minor	It was established that there is small offset in centre of gravity due to manner the wiring has been completed resulting in small pull when the left rotation is executed. It was not resolved as it was not preventing from completing of the gait and it would require significant efforts to fully rectify.
TC_13	Custom Mode	User selects "Custom" mode, it should allow user to move the right controller forward in order to complete object climbing gait and when controller released, return to "idle" state.	User was able to complete the gait with some coordination issues.	Expected	Minor	It is due to complexity of the gait and high elevation required for the gait, this requires further improvements and integration of IMU sensor to improve stability, out of scope of the project
TC_14	Adjusting Orientation: Elevation	When user moves the slider in Mode Selector (MS) console, the robot should increase the elevation to desired position with 50 and 150 millimetres as well as update height indicator within MS console with relative values.	User was able to successfully update the elevation values, however, as per manual measurement the clearance was slightly below expected value.	Bug	Minor	It was due to incorrect offset limit being applied, it was corrected before the release.
TC_15	Adjusting Orientation: Pitch & Roll	When user moves left controller forward or back, QPR should adjust its pitch values accordingly and when left controller is moved to left or right, QPR will adjust its roll values.	User was able to move the slider forward and back and left and right resulting in pitch and roll adjustments, respectively. User also noticed that moving controller at an angle position, it results in adjustments of both pitch and roll simultaneously.	Expected	None	It was anticipated behaviour.
TC_16	Combining Orientation with Directional Mode	User should be able to combine inputs from both left and right controllers while executing directional gait.	User was able to perform forward gait while pitch, roll and elevation values were increased.	None	None	None

TC_17	Combining Orientation with Rotational Mode	User should be able to combine inputs from both left and right controllers while executing rotational gait.	User was able to perform rotate right and rotate left gait while pitch, roll and elevation values were increased.	None	None	None
-------	--	---	---	------	------	------

Table 14: Testing Scenarios

7.2 Evaluation of the Requirements

As per the Chapter 2, the set of the Functional and Non-Functional Requirements have been evaluated, Table 15. present the results with added columns on accomplishment status and comments section with justification of the results.

Functional Requirements				
#	Description	Priority	Status	Comments
FR01	User must be able to control device from web interface without need for special controller.	Must	Fully Satisfied	QRP can be accessed from PC or Tablet and does not require additional hardware.
FR02	User should not need to download any additional app's, drivers, and any companion software.	Should	Fully Satisfied	QRP can be accessed from PC or Tablet and does not require additional software, can be accessed with any web browser.
FR03	User must be able to use on-screens controls such as joysticks, buttons to operate the device	Must	Fully Satisfied	UI is using "on-screen" controllers to operate direction and orientation of the robot as well as switch the gaits and access settings with standard UI elements
FR04	Robot must provide at least 15 min. of operation time before being re-charged.	Must	Fully Satisfied	Robot successfully operated for > 15 minutes with mixed use.
FR05	Robot must be able to move in any direction on even surface.	Must	Partially Satisfied	Robot can move in left, right, forward, back directions and rotate left and right however is missing omni-directional movement.
FR06	Robot body should be able to maintain orientation in any deflections within min. 30 degrees horizontal or vertical deflection.	Should	Not Satisfied	Issues with software implementation of IMU sensor which resulted in lack of ability to auto stabilise its orientation
FR07	User should be presented with live data from actuators such as Voltage, Current Position, Temperature.	Should	Fully Satisfied	Leg tables provide live update of Temperature, Voltage and Current Position.
FR08	User must be able to set limits on position, load or torque for each actuator using web interface.	Must	Partially Satisfied	Robot has ability to set Maximum and Minimum position limits within UI, however implementation of Torque and Load is missing due to time constrains but it also proved to be not essential for the success of the project.
FR09	User should be provided with live environmental data such as orientation from MEMES sensor (gyroscope, acceleration, magnetometer) and pressure per each limb.	Should	Not Satisfied	Issues with software implementation of IMU sensor which resulted in lack of ability to provide orientation feedback
FR10	Robot must be able to move without any tethers or harness	Must	Fully Satisfied	Robot can operate wirelessly using built in battery pack and Wi-Fi connectivity.

FR11	Robot should be able to climb over small obstacles such as stairs.	Should	Partially Satisfied	Robot can climb obstacle of up to 10 cm height but requires preparation and correct placement before the execution of the custom gait.
FR12	Robot should be able to move any direction within 30 degree horizontal and / or vertical deflection.	Should	Partially Satisfied	Robot can move with the deflection limits of 30 degrees in pitch and roll however it shows instability at higher elevation levels resulting in lowering values to 20 degrees only.
FR13	Robot should have ability to interface with external devices.	Could	Partially Satisfied	Controller board attaches to Power Distribution board using external connector wit RS485 bus, this can be reused for connecting to an external module.
FR14	Robot could be integrated with LIDAR Sensor	Could	Not Satisfied	No dedicated connector to integrate with Lidar module
FR15	Robot could be integrated using API with external systems i.e. ROS	Could	Partially Satisfied	ROS can be integrated by using HTTP request with valid query strings to operate the robot, however, dedicated API functionality not included.

Non-Functional Requirements

#	Description	Priority	Status	Comments
NFR01	The project should remain price competitive with competition and affordable for individual consumer.	Must	Fully Satisfied	The final cost estimate for parts approximate £350, making it affordable for individual consumer
NFR02	The UI should be compatible with all common Smartphones and Tablets screens.	Should	Partially Satisfied	UI is compatible with Screen of 1024 x 768 and higher, on smaller screens the element will not fit properly and require additional scrolling or zooming.
NFR03	The access to connect with the device should be secured with password.	Must	Fully Satisfied	User is required to enter password before establishing connection, the password is also stored in configuration file within the device making it difficult to access for not authorised users.
NFR04	The UI should be clean and intuitive.	Must	Fully Satisfied	UI uses simply and clean layout with application UX design principles.
NFR05	Robot controls and feedback should be responsive.	Should	Partially Satisfied	Controls are sufficiently responsive, however, dynamic gait implementation is missing that would allow for interruption of gait execution at any time rather than after its completion
NFR06	Robot should be safe to operate to avoid injuries or accidental damage.	Must	Fully Satisfied	QPR has been designed with position limits to avoid accidental damage to the actuators as well as temperature and battery voltage feedback allowing to monitor the operation of the actuators and act if required. The battery pack comes with separated output and balance charging terminals allowing for calibrated charging process.

Table 15: Functional and Non-Functional Requirements Evaluation

8. Conclusion

The project proved to be very challenging but also highly rewarding as it required number non-homogenies activities such 3D and CAD design, manual assemblies of both frame and electronics and extensive coding with usage of trigonometry. This allowed me to learn and develop number of skills and re-use the knowledge gained throughout the course.

8.1 Complexity

Figure 64. demonstrates complexity of code execution when user pushes left and right controller to complete forward movement. The processing of inputs is split into several steps such as, accessing and storing data, converting inputs into set of X,Y,Z coordinates, applying offsets to coordinates, conversion of coordinates into angular values for each leg separately and finally compiling packets for driving network of twelve RS518 actuators to move synchronously.

All these operations are executed every 200ms, or 5 times each second and all files, services and processing is being handled by £1.49 module (ESP32 SoC.)

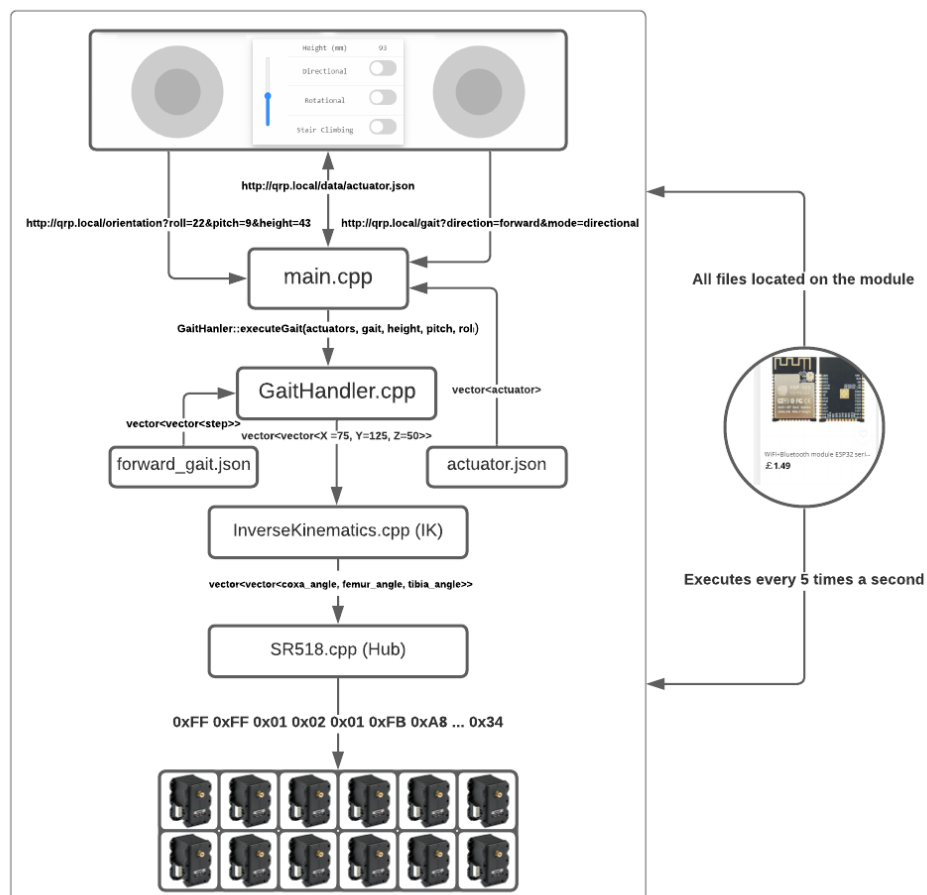


Figure 64: Processing of forward gait movement

8.2. Challenges

There were number of daily and weekly challenges that had to be solved throughout the project development cycle, following few might be useful for similar projects in future.

1. *Frame Design – prototyping and iterations.*

- **Frame footprint vs weight?** – footprint proved to be more important than weight as the larger the footprint will act as lever and put unnecessary stress on all actuators that can easily exceed the stall values.
- **Tibia vs Femur: how long?** – the ratio between the lengths will influence how the load is distributed between Femur and Tibia actuators, Femur actuator tend to be under more stress as they need to not only move its assigned load but also lift the weight of Tibia assembly, and only way to equalise the loads is to design leg in way that tibia is always longer than femur part.
- **Tibia vs Femur: how thick?** – materials used were also an important factor that required an adjustment, my initial design had to be changed as It was made from PLA material that was stronger than the UV Resin used in final design. Resin parts tend to be more brittle and less flexible resulting in part shattering at weak points, many element of leg assemblies had to be re-modelled and re-enforced with additional material.

2. *Controller Board and Power Distribution Board design.*

- **What components to use?** – selection of the components such as MCU is important, initially the design was going to use 8-bit microcontroller with external WIFI module and memory expansions, however the ESP32 SoC proved to be better solution as it was combining all the required functionality in a single module while being still very affordable and fully compatible with existing Arduino libraries.
- **How to connect the parts?** – the initial design was assuming that the ESP32 would drive 4 separate RS485 serial buses for each leg, however after the testing, the serial software port that was used for one of the buses was not fully compatible and had issues with reading the incoming status packets. This resulted in abandoning the idea and single serial bus was selected for driving actuators in “daisy-chain” setup.
- **How to fit everything on PCB?** – initially all the components were going to be included on a single PCB design, however due to size limitation, the board had to be redesign and split between two PCB designs, CB and PDB board.

3. *Code issues and limitations*

- The code had to be design with memory consumption and processing in mind as all UI and program files had to be stored directly into microcontroller with limited processing power and size of available memory (520 KB RAM, 448KB ROM and 4MB SPI Flash, CPU clocked at 160 MHz)
- The code had to be structured in way that was compatible with manufacturer frameworks and provided 3rd party libraries.

- Board performance and stability – this was the main issue behind missing implementation behind the IMU sensor, the implementation initially worked and QPR was receiving correct pitch and roll values that could be used for stabilisation but due to issues with the 3rd party library, a frequent reset of the board occurred resulting in losing control of the robot for certain periods of the time. This was still investigated at the time of writing the report and a future improvement will be required.
- Combining Inverse Kinematics with Gaits – this was one of the most challenging tasks during the development process as it required two very complex classes to cooperate before any issues could be investigated. This often was causing a great difficulty in identifying the culprits.
- Conversion of data types from int into float values resulted in rounding values error making it not viable for trigonometric equations in IK class.
- Finding optimal values for given IK scenarios, i.e. increasing X,Y,Z values beyond scope would result in actuators resetting to 0 position (NaN issue).
- Combining and calculating offsets coming from various sources such actuator's physical orientation, user inputs (Roll, Pitch, Height), and actuators Min and Max position limits, this can become confusing very quickly.
- Timing the gait executions, providing insufficient delay time was resulting in gait sequences not fully completing, increasing the delay values too much resulted in imbalance and coordination issues.

8.3 Future Improvements

There are countless improvements that can be applied to hardware and software and I am intending to continue working and expanding the project in 2021 to include following functionality.

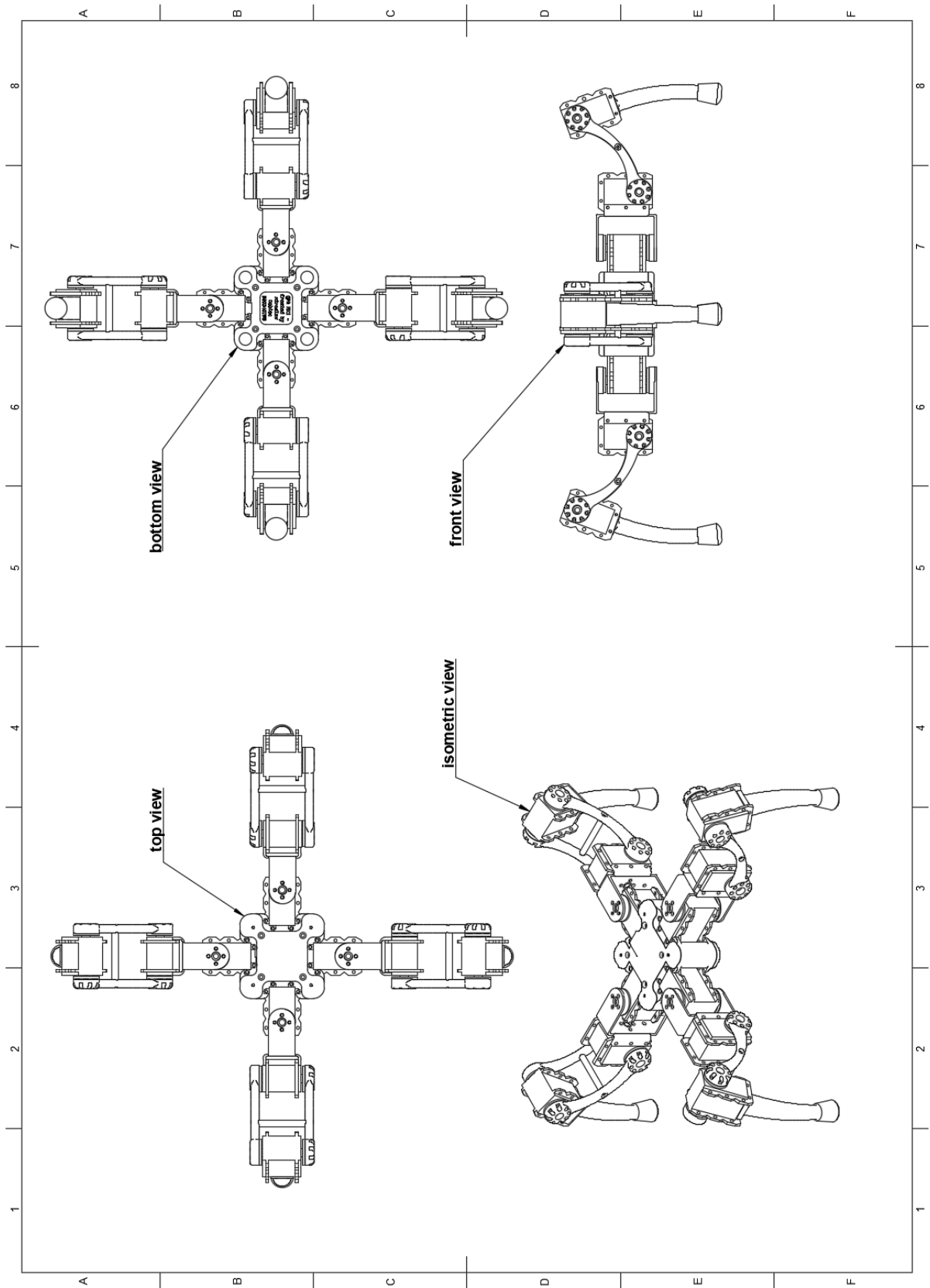
- Implementing IMU sensor (gyroscope stabilization)
- Implementing Dynamic Gaits through expanding the Inverse Kinematics model i.e. X,Y,Z coordinates would be generated in-flight and based on the changes to centre of gravity.
- Improving responsiveness by redesigning code to handle interrupts as well as moving into faster, UDP protocol and increasing the baud-rate values for actuator communication.
- Implementing dynamic position limits that change based on the location of the joints at any given time.
- Improving UI to be fully compatible with smartphone size screens.

With beginning of 2022, I am also hoping to start work on improved design with more accurate and power efficient actuators and even more movement capabilities.

References:

1. Rachel Lerman, Jay Greene, Big Tech was first to send workers home. Now it's in no rush to bring them back. The Washington Post, May 18, 2020, <https://www.washingtonpost.com/technology/2020/05/18/facebook-google-work-from-home> (accessed on October 20th, 2020)
2. Will Knight, Robot dexterity, MIT Technology Review, April 2nd, 2020, <https://www.technologyreview.com/technology/robot-dexterity/> (accessed on November 10th, 2020)
3. E. Cha, A. D. Dragan and S. S. Srinivasa, Perceived robot capability, 2015 24th IEEE International Symposium on Robot and Human Interactive Communication, <https://ieeexplore.ieee.org/document/7333656> (accessed on November 8th, 2020)
4. Spot, Boston Dynamics 2020, <https://www.bostondynamics.com/spot> (accessed on 15th November 2020)
5. Unitree Robotic, A1, <https://www.unitree.com/products/a1/> (accessed on 15th November 2020)
6. Jennifer Chu, MIT News Office, Mini cheetah is the first four-legged robot to do a backflip, March 4th, 2019, <https://news.mit.edu/2019/mit-mini-cheetah-first-four-legged-robot-to-backflip-0304> (accessed on November 15th 2020)
7. Lynxmotion 2020, <http://www.lynxmotion.com/c-26-quadrupods.aspx>
8. Trossen Robotics 2020, <https://www.trossenrobotics.com/d-kitty.aspx>
9. M. Ahn, H. Zhu, K. Hartikainen, H. Ponte, A. Gupta, S. Levine, V. Kumar, ROBEL: Robotics Benchmarks for Learning with Low-Cost Robots, September 25th, 2019, Cornell University (accessed on November 15th 2020)
10. About Value Engineering, Save, 2020, <https://www.value-eng.org/page/AboutVE> (accessed on November 19th, 2020)
11. ESP8266, A cost-effective and highly integrated Wi-Fi MCU for IoT applications, <https://www.espressif.com/en/products/socs/esp8266> (accessed April 10th, 2021)
12. ESP32, A feature-rich MCU with integrated Wi-Fi and Bluetooth connectivity for a wide-range of applications, <https://www.espressif.com/en/products/socs/esp32> (accessed April 10th, 2021)
13. F. Giones, A. Brem, December 2017, From toys to tools: The co-evolution of technological and entrepreneurial developments in the drone industry, Business Horizons, <https://www.sciencedirect.com/science/article/pii/S0007681317301210> (accessed on November 15th)
14. CMMI Limited, March 2002, Requirements Development (RD), <http://www.cmmi.co.uk/cmmi/RD.htm> (accessed on November 23rd)
15. Keen, R., 2021. Actions To Address Risks And Opportunities Explained [with procedure]. [online] ISO 9001 Checklist. Available at: <<https://www.iso-9001-checklist.co.uk/6.1-actions-to-address-risks-and-opportunities-gbp.htm>> (Accessed 8 April 2021).
16. Enginess.io. 2021. *The 6 Principles Of Design, a la Donald Norman | Enginess Insights*. [online] Available at: <<https://www.enginess.io/insights/6-principles-design-la-donald-norman>> [Accessed 24 April 2021].
17. O'Reilly Online Learning. 2021. *Software Architecture Patterns*. [online] Available at: <<https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch01.html>> [Accessed 24 April 2021].

Appendix 1: Frame Design

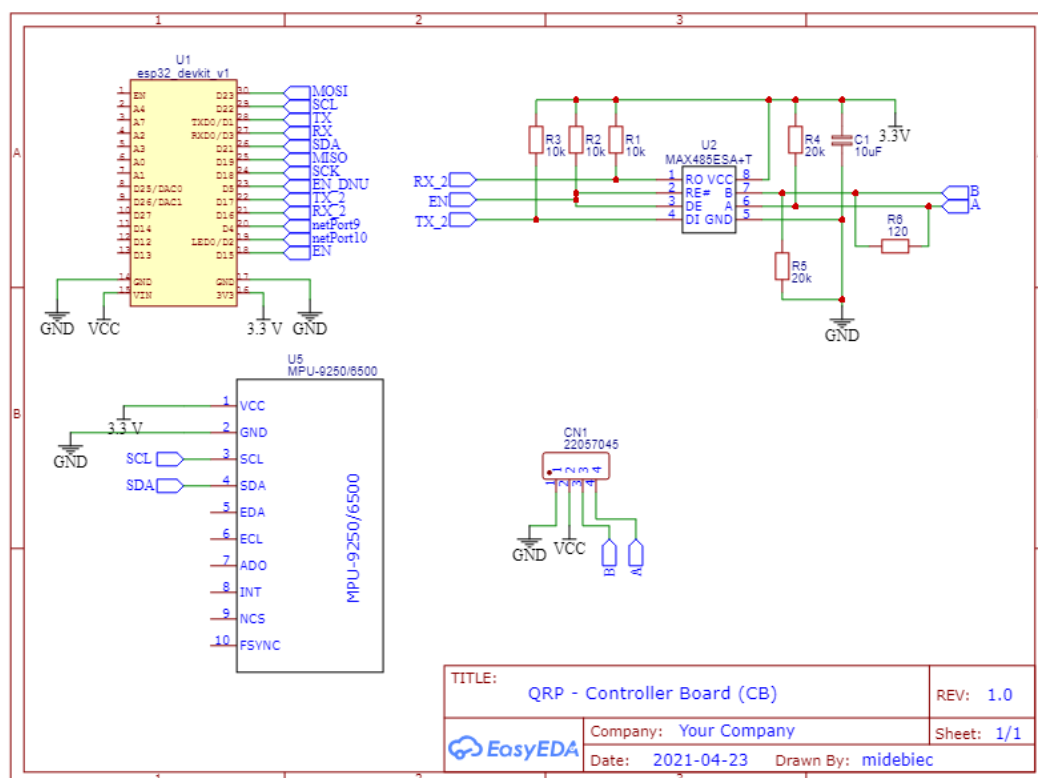
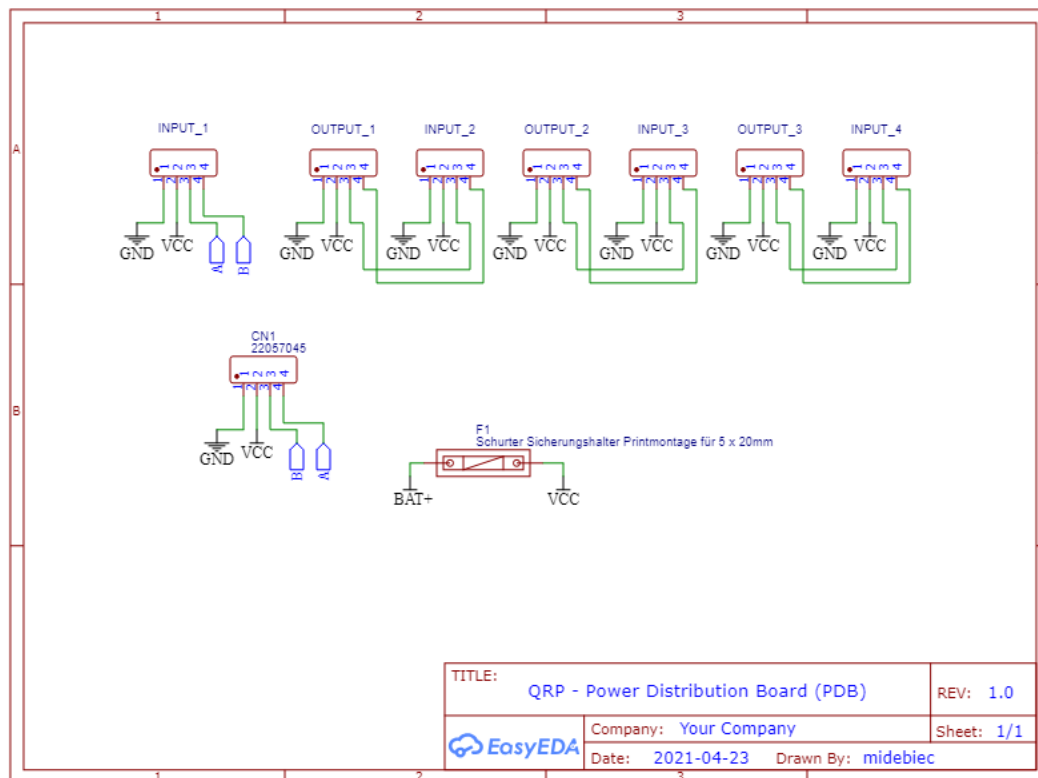


Appendix 2: SR518 - Control Table

Address hexadecima	Description	Read write	Initial Value (Hexadecimal)	Store area
0 (0x00)	--	--	--	EEPROM
1 (0x01)	--	--	--	
2 (0x02)	Version of Firmware	read	--	
3 (0x03)	ID	read/write	1 (0x01)	
4 (0x04)	Baud Rate	read/write	1 (0x01)	
5 (0x05)	Return Delay Time	read/write	0 (0x00)	
6 (0x06)	Lowest byte of clockwise Angle Limit (L)	read/write	0 (0x00)	
7 (0x07)	Highest byte of clockwise Angle Limit (H)	read/write	0 (0x00)	
8 (0x08)	Lowest byte of counterclockwise Angle Limit (L)	read/write	255 (0xFF)	
9 (0x09)	Highest byte of counterclockwise Angle Limit (H)	read/write	3 (0x03)	
10 (0x0A)	--			
11 (0x0B)	the Highest Limit Temperature	read/write	80 (0x50)	
12 (0x0C)	Lowest Limit Voltage	read/write	?	
13 (0x0D)	Highest Limit Voltage	read/write	?	
14 (0x0E)	Max Torque(L)	read/write	255 (0xFF)	
15 (0x0F)	Max Torque(H)	read/write	3 (0x03)	
16 (0x10)	Status Return Leve	read/write	2 (0x02)	
17 (0x11)	Alarm LED	read/write	37 (0x25)	
18 (0x12)	Unload condition	read/write	4 (0x04)	
19 (0x13)	--	--	--	
20 (0x14)	Potentiometer correction	--	--	
21 (0x15)	Potentiometer correction	--	--	
22 (0x16)	Potentiometer correction	--	--	
23 (0x17)	Potentiometer correction	--	--	
24 (0x18)	Torque On/Off	read/write	0 (0x00)	RAM
25 (0x19)	LED On/Off(switch)	read/write	0 (0x00)	
26 (0x1A)	CW Compliance margin	read/write	2 (0x02)	
27 (0x1B)	CCW Compliance margin	read/write	2 (0x02)	

28 (0x1C)	Clock wise ratio	read/write	32 (0x20)	
29 (0x1D)	Counter clock wise ratio	read/write	32 (0x20)	
30 (0x1E)	Goal Position(L)	read/write	[Addr36]value	
31 (0x1F)	Goal Position(H)	read/write	[Addr37]value	
32 (0x20)	Moving Speed (L)	read/write	0	
33 (0x21)	Moving Speed (H)	read/write	0	
34 (0x22)	Accelerating speed	read/write	32	
35 (0x23)	Decelerating speed	read/write	32	
36 (0x24)	Present Position(L)	read	?	
37 (0x25)	Present Position(H)	read	?	
38 (0x26)	Present Speed (L)	read	?	
39 (0x27)	Present Speed (H)	read	?	
40 (0x28)	Present Load	read	?	
41 (0x29)	Present Load	read	?	
42 (0x2A)	Present Voltage	read	?	
43 (0x2B)	Present Temperature	read	?	
44 (0x2C)	REG WRITE mark	read	0 (0x00)	
45 (0x2D)	--		0 (0x00)	
46 (0x2E)	In operation	read	0 (0x00)	
47 (0x2F)	Lock mark	read/write	0 (0x00)	
48 (0x30)	minimumPWM(L)	read/write	90 (0x5A)	
49 (0x31)	minmumPWM(H)	read/write	00 (0x00)	

Appendix 3: Power Distribution Board (PDB) and Controller Board (CB)



Appendix 4: Index.html

```
[1] <!DOCTYPE html>
[2] <html lang="en">
[3]
[4] <head>
[5]   <meta charset="UTF-8">
[6]   <meta name="viewport" content="width=device-width, initial-scale=1.0">
[7]   <link rel="stylesheet" href="css/bootstrap.min.css">
[8]   <link rel="stylesheet" href="css/style.css">
[9]   <script src="js/jquery.min.js"></script>
[10]  <script src="js/popper.min.js"></script>
[11]  <script src="js/bootstrap.min.js"></script>
[12]
[13]   <title>Quadruped</title>
[14] </head>
[15]
[16] <body>
[17]   <div class="grid-container">
[18]     <!-- LIMB TABLES -->
[19]
[20]     <!-- LEG 1 TABLE -->
[21]     <div class="grid-item" id="leg-1-card">
[22]       <div class="leg-header">&#9713; 1 FRONT-RIGHT
[23]       <div class="direction_group">
[24]         <span class="dir_input">X:<input type="text" id="x-data-1" placeholder="X" value="75"></span>
[25]         <span class="dir_input">Y:<input type="text" id="y-data-1" placeholder="Y" value="75"></span>
[26]         <span class="dir_input">Z:<input type="text" id="z-data-1" placeholder="Z" value="50"></span>
[27]         <span class="dir_input"><button class="dir_btn" onclick="setPos(1)">&#10144;</button></span>
[28]       </div>
[29]     </div>
[30]     <table>
[31]       <thead>
[32]         <th>ID</th>
[33]         <th>JOINT</th>
[34]         <th>VOLTAGE</th>
[35]         <th>TEMPERATURE</th>
[36]         <th>POSITION</th>
[37]         <th>#</th>
[38]       </thead>
[39]       <tbody id="leg_1_table"></tbody>
[40]     </table>
[41]   </div>
[42]
[43]   <!-- LEG 2 TABLE -->
[44]   <div class="grid-item" id="leg-2-card">
[45]     <div class="leg-header">&#9712; 2 BACK-RIGHT
[46]     <div class="direction_group">
[47]       <span class="dir_input">X:<input type="text" id="x-data-2" placeholder="X" value="75"></span>
[48]       <span class="dir_input">Y:<input type="text" id="y-data-2" placeholder="Y" value="75"></span>
[49]       <span class="dir_input">Z:<input type="text" id="z-data-2" placeholder="Z" value="50"></span>
[50]       <span class="dir_input"><button class="dir_btn" onclick="setPos(2)">&#10144;</button></span>
[51]     </div>
[52]   </div>
[53]   <table>
[54]     <thead>
[55]       <th>ID</th>
[56]       <th>JOINT</th>
[57]       <th>VOLTAGE</th>
[58]       <th>TEMPERATURE</th>
[59]       <th>POSITION</th>
[60]       <th>#</th>
[61]     </thead>
[62]     <tbody id="leg_2_table"></tbody>
[63]   </table>
[64] </div>
[65]
[66]   <!-- LEG 3 TABLE -->
[67]   <div class="grid-item" id="leg-3-card">
[68]     <div class="leg-header">&#9715; 3 BACK-LEFT
[69]     <div class="direction_group">
[70]       <span class="dir_input">X:<input type="text" id="x-data-3" placeholder="X" value="75"></span>
[71]       <span class="dir_input">Y:<input type="text" id="y-data-3" placeholder="Y" value="75"></span>
[72]       <span class="dir_input">Z:<input type="text" id="z-data-3" placeholder="Z" value="50"></span>
[73]       <span class="dir_input"><button class="dir_btn" onclick="setPos(3)">&#10144;</button></span>
[74]     </div>
[75]   </div>
[76]   <table>
[77]     <thead>
[78]       <th>ID</th>
[79]       <th>JOINT</th>
[80]       <th>VOLTAGE</th>
[81]       <th>TEMPERATURE</th>
[82]       <th>POSITION</th>
[83]       <th>#</th>
[84]     </thead>
[85]     <tbody id="leg_3_table"></tbody>
[86]   </table>
[87] </div>
[88]
[89]   <!-- LEG 4 TABLE -->
[90]   <div class="grid-item" id="leg-4-card">
[91]     <div class="leg-header">&#9714; 4 FRONT-LEFT
[92]     <div class="direction_group">
[93]       <span class="dir_input">X:<input type="text" id="x-data-4" placeholder="X" value="75"></span>
[94]       <span class="dir_input">Y:<input type="text" id="y-data-4" placeholder="Y" value="75"></span>
[95]       <span class="dir_input">Z:<input type="text" id="z-data-4" placeholder="Z" value="50"></span>
[96]       <span class="dir_input"><button class="dir_btn" onclick="setPos(4)">&#10144;</button></span>
[97]     </div>
[98]   </div>
[99]   <table>
[100]    <thead>
[101]      <th>ID</th>
[102]      <th>JOINT</th>
[103]      <th>VOLTAGE</th>
[104]      <th>TEMPERATURE</th>
[105]      <th>POSITION</th>
[106]      <th>#</th>
[107]    </thead>
[108]    <tbody id="leg_4_table"></tbody>
[109]  </table>
[110] </div>
[111]
[112]   <!-- MODE SELECTOR -->
[113]   <div class="grid-item" id="selector_table">
```

```

[114]         <table id="selector_table">
[115]             <tr>
[116]                 <td rowspan="4"><input type="range" class="form-control-range" id="height_slider" oninput="setHeight()" min="0" max="100"
value="0"></input>
[117]                 </td>
[118]                 <td>Height (mm)</td>
[119]                 <td class="align-middle" id="height_value"></td>
[120]             </tr>
[121]             <tr>
[122]                 <td>Directional</td>
[123]                 <td>
[124]                     <label class="switch">
[125]                         <input id="directional_mode" type="checkbox" mode="directional">
[126]                         <div class="slider round"></div>
[127]                     </label>
[128]                 </td>
[129]             </tr>
[130]             <tr>
[131]                 <td>Rotational</td>
[132]                 <td>
[133]                     <label class="switch">
[134]                         <input id="rotational_mode" type="checkbox" mode="rotational">
[135]                         <div class="slider round"></div>
[136]                     </label>
[137]                 </td>
[138]             </tr>
[139]             <tr>
[140]                 <td>Stair Climbing</td>
[141]                 <td>
[142]                     <label class="switch">
[143]                         <input id="stairclimbing_mode" type="checkbox" mode="stairclimbing">
[144]                         <div class="slider round"></div>
[145]                     </label>
[146]                 </td>
[147]             </tr>
[148]         </table>
[149]     </div>
[150]
[151]     <!-- LEFT CONTROLLER -->
[152]     <div class="grid-item" id="left-controls"></div>
[153]
[154]     <!-- RIGHT CONTROLLER -->
[155]     <div class="grid-item" id="right-controls"></div>
[156] </div>
[157]
[158] <!-- MODAL -->
[159] <div class="modal fade" id="configWindow" role="dialog">
[160]     <div class="modal-dialog">
[161]
[162]         <!-- Modal content-->
[163]         <div class="modal-content">
[164]             <div class="modal-header">
[165]                 <h4 class="modal-title">Modal Header</h4>
[166]                 <button type="button" class="close" data-dismiss="modal">&times;</button>
[167]             </div>
[168]             <div class="modal-body">
[169]                 <div class="form-group">
[170]                     <div class="row" style="margin: 10px;">
[171]                         <div class="col-7"><label>Minimum Position Limit:</label></div>
[172]                         <div class="col-5" class="input-values" id="config_min_pos"></div>
[173]                     </div>
[174]                     <div class="row" style="margin: 10px;">
[175]                         <div class="col-7"><label>Maximum Position Limit:</label></div>
[176]                         <div class="col-5" class="input-values" id="config_max_pos"></div>
[177]                     </div>
[178]                     <div class="row" style="margin: 10px;">
[179]                         <div class="col-7"><label>Current Position:</label></div>
[180]                         <div class="col-5" class="input-slider" id="current_pos"></div>
[181]                     </div>
[182]                 </div>
[183]             </div>
[184]             <div class="modal-footer">
[185]                 <button type="button" class="btn btn-default" data-dismiss="modal">Cancel</button>
[186]                 <button type="button" id="save_button" class="btn btn-default" data-dismiss="modal">Save</button>
[187]             </div>
[188]         </div>
[189]     </div>
[190] </div>
[191] </div>
[192] <!-- END of MODAL-->
[193] <script src="js/nipplejs.js"></script>
[194] <script src="js/script.js"></script>
[195] </body>
[196] </html>

```

Appendix 5: Script.js

```

[1] var adhocURL = "http://qrp.local/";
[2] var l_event, l_distance, l_angle, l_direction, r_event, r_distance, r_angle, r_direction;
[3] var actuators;
[4] var xhttp = new XMLHttpRequest();
[5] var mode = "manual";
[6] var height = 0;
[7] var pitch = 0;
[8] var roll = 0;
[9]
[10] $(document).ready(function() {
[11]     var refresh = setInterval(function() {
[12]         buildActuatorTable();
[13]     }, 200);
[14]
[15]     setHeight();
[16]
[17]     $('input:checkbox').click(function() {
[18]         mode = $(this).attr("mode");
[19]         $('input:checkbox').not(this).prop('checked', false);
[20]     });
[21] });
[22]
[23]
[24] function buildActuatorTable() {

```



```

[25] $.getJSON("data/actuator.json", function(data) {
[26]
[27]     actuators = data;
[28]     var leg_1_table_data = "";
[29]     var leg_2_table_data = "";
[30]     var leg_3_table_data = "";
[31]     var leg_4_table_data = "";
[32]
[33]
[34]     $.each(data, function(key, value) {
[35]         if (value.leg === "1") {
[36]             leg_1_table_data += "<tr>";
[37]             leg_1_table_data += "<td>" + value.id + "</td>";
[38]             leg_1_table_data += "<td>" + value.joint_name + "</td>";
[39]             leg_1_table_data += "<td>" + value.voltage + " V</td>";
[40]             leg_1_table_data += "<td>" + value.temperature + " &#8451;</td>";
[41]             leg_1_table_data += "<td>" + value.position + " &#176;</td>";
[42]             leg_1_table_data += "<td><button class='btn' id='settings_" + value.id + "' data-toggle='modal' data-target='#configWindow' data-id='"
+ value.id + "'>&#9881;</button></td>";
[43]             leg_1_table_data += "</tr>";
[44]         }
[45]
[46]         if (value.leg === "2") {
[47]             leg_2_table_data += "<tr>";
[48]             leg_2_table_data += "<td>" + value.id + "</td>";
[49]             leg_2_table_data += "<td>" + value.joint_name + "</td>";
[50]             leg_2_table_data += "<td>" + value.voltage + " V</td>";
[51]             leg_2_table_data += "<td>" + value.temperature + " &#8451;</td>";
[52]             leg_2_table_data += "<td>" + value.position + " &#176;</td>";
[53]             leg_2_table_data += "<td><button class='btn' id='settings_" + value.id + "' data-toggle='modal' data-target='#configWindow' data-id='"
+ value.id + "'>&#9881;</button></td>";
[54]             leg_2_table_data += "</tr>";
[55]         }
[56]
[57]         if (value.leg === "3") {
[58]             leg_3_table_data += "<tr>";
[59]             leg_3_table_data += "<td class='align-middle'>" + value.id + "</td>";
[60]             leg_3_table_data += "<td class='align-middle'>" + value.joint_name + "</td>";
[61]             leg_3_table_data += "<td class='align-middle'>" + value.voltage + " V</td>";
[62]             leg_3_table_data += "<td class='align-middle'>" + value.temperature + " &#8451;</td>";
[63]             leg_3_table_data += "<td class='align-middle'>" + value.position + " &#176;</td>";
[64]             leg_3_table_data += "<td class='align-middle'><button class='btn' id='settings_" + value.id + "' data-toggle='modal' data-
target='#configWindow' data-id='" + value.id + "'>&#9881;</button></td>";
[65]             leg_3_table_data += "</tr>";
[66]         }
[67]
[68]         if (value.leg === "4") {
[69]             leg_4_table_data += "<tr>";
[70]             leg_4_table_data += "<td class='align-middle'>" + value.id + "</td>";
[71]             leg_4_table_data += "<td class='align-middle'>" + value.joint_name + "</td>";
[72]             leg_4_table_data += "<td class='align-middle'>" + value.voltage + " V</td>";
[73]             leg_4_table_data += "<td class='align-middle'>" + value.temperature + " &#8451;</td>";
[74]             leg_4_table_data += "<td class='align-middle'>" + value.position + " &#176;</td>";
[75]             leg_4_table_data += "<td class='align-middle'><button class='btn' id='settings_" + value.id + "' data-toggle='modal' data-
target='#configWindow' data-id='" + value.id + "'>&#9881;</button></td>";
[76]             leg_4_table_data += "</tr>";
[77]         }
[78]     });
[79]
[80]     $("#leg_1_table").html(leg_1_table_data);
[81]     $("#leg_2_table").html(leg_2_table_data);
[82]     $("#leg_3_table").html(leg_3_table_data);
[83]     $("#leg_4_table").html(leg_4_table_data);
[84]
[85]     }.bind(this));
[86] };
[87]
[88] //MODAL JS
[89] $('#configWindow').on('show.bs.modal', function(event) {
[90]     var button = $(event.relatedTarget);
[91]     var id = button.data('id');
[92]     var modal = $(this);
[93]
[94]     var max_pos = actuators[id - 1].max_position;
[95]     var min_pos = actuators[id - 1].min_position;
[96]
[97]     modal.find('.modal-title').text("Actuator Config: " + id);
[98]     modal.find("#config_min_pos").html("<div class='input-group'><div class='input-group-prepend'><button class='btn btn-dark'
onclick='decrease_min_pos(\" + id + \")'></button></div><input onpropertychange='adjustSliderRange(\" + id + \")' type='number' id='min_pos_" + id + "'
type='text' class='form-control' value=\"" + min_pos + "' min='0' max='1024'><div class='input-group-append'><button class='btn btn-dark'
onclick='increase_min_pos(\" + id + \")'></button></div></div>");
[99]     modal.find("#config_max_pos").html("<div class='input-group'><div class='input-group-prepend'><button class='btn btn-dark'
onclick='decrease_max_pos(\" + id + \")'></button></div><input onchange='adjustSliderRange(\" + id + \")' type='number' id='max_pos_" + id + "' type='text'
class='form-control' value=\"" + max_pos + "' min='0' max='1024'><div class='input-group-append'><button class='btn btn-dark'
onclick='increase_max_pos(\" + id + \")'></button></div></div>");
[100]     modal.find("#current_pos").html("<input type='range' class='form-control-range' id='slider_" + id + "' oninput='move(\" + id + \")' min=\"" + min_pos
+ "\" max=\"" + max_pos + "\"></input>");
[101]     modal.find("#save_button").attr("onclick", "saveConfig(\" + id + \")");
[102] });
[103]
[104] function decrease_min_pos(id) {
[105]     var value = parseInt(document.getElementById('min_pos_' + id).value, 10);
[106]     var min_value = parseInt(document.getElementById('min_pos_' + id).min, 10);
[107]     value = isNaN(value) ? 0 : value;
[108]     value <= min_value ? value = min_value + 1 : '';
[109]     --value;
[110]     document.getElementById('min_pos_' + id).value = value;
[111]     adjustSliderRange(id);
[112] }
[113]
[114] function increase_min_pos(id) {
[115]     var value = parseInt(document.getElementById('min_pos_' + id).value, 10);
[116]     var max_value = parseInt(document.getElementById('max_pos_' + id).value, 10);
[117]     value = isNaN(value) ? 0 : value;
[118]     value >= max_value ? value = max_value - 1 : '';
[119]     ++value;
[120]     document.getElementById('min_pos_' + id).value = value;
[121]     adjustSliderRange(id);
[122] }
[123]
[124] function decrease_max_pos(id) {
[125]     var value = parseInt(document.getElementById('max_pos_' + id).value, 10);
[126]     var min_value = parseInt(document.getElementById('min_pos_' + id).value, 10);
[127]     value = isNaN(value) ? 0 : value;
[128]     value <= min_value ? value = min_value + 1 : '';
[129]     --value;
[130]     document.getElementById('max_pos_' + id).value = value;
[131]     adjustSliderRange(id);
[132] }
[133]

```

```

[134] function increase_max_pos(id) {
[135]     var value = parseInt(document.getElementById('max_pos_' + id).value, 10);
[136]     var max_value = parseInt(document.getElementById('max_pos_' + id).max, 10);
[137]     value = isNaN(value) ? 0 : value;
[138]     value >= max_value ? value = max_value - 1 : '';
[139]     ++value;
[140]     document.getElementById('max_pos_' + id).value = value;
[141]     adjustSliderRange(id);
[142] }
[143]
[144] function adjustSliderRange(id) {
[145]     var min_pos = parseInt(document.getElementById('min_pos_' + id).value, 10);
[146]     var max_pos = parseInt(document.getElementById('max_pos_' + id).value, 10);
[147]     document.getElementById('slider_' + id).min = min_pos;
[148]     document.getElementById('slider_' + id).max = max_pos;
[149] }
[150]
[151] function move(id) {
[152]     var pos = document.getElementById('slider_' + id);
[153]     xhttp.open('POST', adhocURL + "move?id=" + id + "&pos=" + pos.value);
[154]     xhttp.send();
[155] }
[156]
[157] function saveConfig(id) {
[158]     var min_pos = parseInt(document.getElementById('min_pos_' + id).value, 10);
[159]     var max_pos = parseInt(document.getElementById('max_pos_' + id).value, 10);
[160]     xhttp.open('POST', adhocURL + "save?id=" + id + "&min_pos=" + min_pos + "&max_pos=" + max_pos);
[161]     xhttp.send();
[162] }
[163]
[164] //CONTROLLERS JS
[165] var joystickL = nipplejs.create({
[166]     zone: document.getElementById("left-controls"),
[167]     mode: "static",
[168]     position: { left: "18%", top: "82%" },
[169]     color: "black",
[170]     size: 200,
[171] });
[172]
[173] var joystickR = nipplejs.create({
[174]     zone: document.getElementById("right-controls"),
[175]     mode: "static",
[176]     position: { left: "82%", top: "82%" },
[177]     color: "black",
[178]     size: 200,
[179] });
[180]
[181] joystickL.on('start', function(evt, data) {
[182]     l_event = evt.type;
[183] }).on('end', function(evt, data) {
[184]     l_event = evt.type;
[185]     pitch = 0;
[186]     roll = 0;
[187]     xhttp.open('POST', adhocURL + "orientation?roll=" + roll + "&pitch=" + pitch + "&height=" + height);
[188]     xhttp.send();
[189] }).on('move', function(evt, data) {
[190]     l_event = evt.type;
[191]     l_distance = Math.round(data.distance);
[192]     l_angle = Math.round(data.angle.degree);
[193]     roll = getRoll(l_angle, l_distance);
[194]     pitch = getPitch(l_angle, l_distance);
[195]     xhttp.open('POST', adhocURL + "orientation?roll=" + roll + "&pitch=" + pitch + "&height=" + height);
[196]     xhttp.send();
[197] });
[198]
[199]
[200] joystickR.on('start', function(evt, data) {
[201]     r_event = evt.type;
[202] }).on('end', function(evt, data) {
[203]     r_event = evt.type;
[204]     r_distance = 0;
[205]     r_direction = "idle";
[206]     xhttp.open('POST', adhocURL + "gait?direction=" + r_direction + "&mode=" + mode);
[207]     xhttp.send();
[208] }).on('move', function(evt, data) {
[209]     r_event = evt.type;
[210]     r_distance = Math.round(data.distance);
[211]     r_angle = Math.round(data.angle.degree);
[212]     r_direction = getDirection(r_angle);
[213]     if (r_distance > 50) {
[214]         xhttp.open('POST', adhocURL + "gait?direction=" + r_direction + "&mode=" + mode);
[215]         xhttp.send();
[216]     }
[217] });
[218]
[219] function getDirection(angle) {
[220]     if (angle >= 45 && angle < 135) return "up";
[221]     else if (angle >= 135 && angle < 225) return "left";
[222]     else if (angle >= 225 && angle < 315) return "down";
[223]     else return "right";
[224] }
[225]
[226] function getRoll(angle, distance) {
[227]     relativeAngle = getRelativeAngle(angle);
[228]     var x = Math.round(distance * Math.cos(relativeAngle * Math.PI / 180));
[229]     if (angle >= 90 && angle < 270) return parseInt(-x / 4);
[230]     else return parseInt(x / 4);
[231] }
[232]
[233] function getPitch(angle, distance) {
[234]     relativeAngle = getRelativeAngle(angle);
[235]     var y = Math.round(distance * Math.sin(relativeAngle * Math.PI / 180));
[236]     if (angle >= 180 && angle < 360) return parseInt(-y / 4);
[237]     else return parseInt(y / 4);
[238] }
[239]
[240] //XYZ Position
[241] function setPos(leg) {
[242]     var x = document.getElementById("x-data-" + leg).value;
[243]     var y = document.getElementById("y-data-" + leg).value;
[244]     var z = document.getElementById("z-data-" + leg).value;
[245]     $('input:checkbox').prop('checked', false);
[246]     mode = "manual";
[247]     xhttp.open('POST', adhocURL + "pos?id=" + leg + "&x=" + x + "&y=" + y + "&z=" + z);
[248]     xhttp.send();
[249] }
[250]
[251] function getRelativeAngle(angle) {
[252]     var relativeAngle = 0;
[253]     if (angle >= 0 && angle < 90)

```

```

[254]         relativeAngle = angle;
[255]     else if (angle >= 90 && angle < 180)
[256]         relativeAngle = (180 - angle);
[257]     else if (angle >= 180 && angle < 270)
[258]         relativeAngle = -(180 - angle);
[259]     else if (angle >= 270 && angle < 360)
[260]         relativeAngle = (360 - angle);
[261]     return relativeAngle;
[262] }
[263]
[264] function setHeight() {
[265]     height = document.getElementById("height_slider").value;
[266]     var temp = 50 + parseInt(height);
[267]     document.getElementById("height_value").innerHTML = temp;
[268]     xhttp.open("POST", adhocURL + "orientation?roll=" + roll + "&pitch=" + pitch + "&height=" + height);
[269]     xhttp.send();
[270] }

```

Appendix 6: Style.css

```

[1] body {
[2]     font-family: Arial, Helvetica, sans-serif;
[3]     margin: 0px;
[4]     padding: 0px;
[5] }
[6]
[7]
[8] /* LAYOUT */
[9] .grid-container {
[10]     display: grid;
[11]     grid-template-columns: auto auto auto auto auto auto;
[12]     grid-template-rows: auto auto auto;
[13] }
[14]
[15] .grid-item {
[16]     background-color: rgba(255, 255, 255, 0.8);
[17]     opacity: 0.8;
[18]     text-align: center;
[19]     padding: 0.3em 0.5em;
[20]     font-size: 30px;
[21] }
[22]
[23] #leg-1-card {
[24]     grid-column: 4 / span 3;
[25]     grid-row: 1;
[26] }
[27]
[28] #leg-2-card {
[29]     grid-column: 4 / span 3;
[30]     grid-row: 2;
[31] }
[32]
[33] #leg-3-card {
[34]     grid-column: 1 / span 3;
[35]     grid-row: 2;
[36] }
[37]
[38] #leg-4-card {
[39]     grid-column: 1 / span 3;
[40]     grid-row: 1;
[41] }
[42]
[43] #left-controls {
[44]     grid-column: 1;
[45]     grid-row: 3;
[46] }
[47]
[48] #right-controls {
[49]     grid-column: 5;
[50]     grid-row: 3;
[51] }
[52]
[53]
[54] /* CENTRAL TABLE */
[55] #selector_table {
[56]     font-family: Consolas;
[57]     grid-column: 3/span 2;
[58]     grid-row: 3;
[59] }
[60]
[61] #height_slider {
[62]     -webkit-appearance: slider-vertical;
[63]     width: 50px;
[64]     height: 150px;
[65] }
[66]
[67] #selector_table td {
[68]     font-size: medium;
[69]     padding: 0.1;
[70]     border-collapse: collapse;
[71] }
[72]
[73] #selector_table tr:nth-child(even) {
[74]     background-color: #fff;
[75] }
[76]
[77] /* TABLE */
[78] .leg-header {
[79]     font-family: Consolas;
[80]     font-size: medium;
[81]     text-align: left;
[82]     padding: 0 0.5em 0.5em 0.5em;
[83] }
[84]
[85] table {
[86]     box-shadow: 0 4px 8px 0 rgba(0, 0, 0, 0.2), 0 6px 20px 0 rgba(0, 0, 0, 0.19);
[87]     border-collapse: collapse;
[88]     border: 1px solid black;
[89]     border: 2px;
[90]     width: 100%;

```

```

[91] }
[92]
[93] th {
[94]     border-bottom: 1px solid #ddd;
[95]     font-family: Consolas;
[96]     font-size: small;
[97]     padding: 0.5em;
[98] }
[99]
[100] td {
[101]     font-size: large;
[102]     font-family: Consolas;
[103]     padding: 0.5em;
[104]     text-align: center;
[105]     border-bottom: 1px solid #ddd;
[106] }
[107]
[108] tr:nth-child(even) {
[109]     background-color: #f2f2f2;
[110] }
[111]
[112] table {
[113]     width: 100%;
[114]     border-collapse: collapse;
[115]     border: 1px;
[116] }
[117]
[118] .direction_group {
[119]     display: inline;
[120]     font-family: Consolas;
[121]     float: right;
[122] }
[123]
[124] .dir_input {
[125]     float: inline-end;
[126]     width: 2em;
[127]     text-align: center;
[128] }
[129]
[130] .dir_input input {
[131]     width: 2em;
[132]     text-align: center;
[133]     margin: 5px 2px;
[134] }
[135]
[136] .dir_btn {
[137]     margin-left: 0.5em;
[138] }
[139]
[140] /* MODAL */
[141]
[142] .modal-title {
[143]     font-family: Consolas;
[144] }
[145]
[146] .modal-body {
[147]     text-align: center;
[148]     font-family: Consolas;
[149] }
[150]
[151] .input-values {
[152]     text-align: center;
[153] }
[154]
[155] /* TOGGLE */
[156]
[157] .switch {
[158]     position: relative;
[159]     display: inline-block;
[160]     width: 60px;
[161]     height: 34px;
[162] }
[163]
[164] .switch input {
[165]     display: none;
[166] }
[167]
[168] .slider {
[169]     position: absolute;
[170]     cursor: pointer;
[171]     top: 0;
[172]     left: 0;
[173]     right: 0;
[174]     bottom: 0;
[175]     background-color: #ccc;
[176]     -webkit-transition: .4s;
[177]     transition: .4s;
[178] }
[179]
[180] .slider:before {
[181]     position: absolute;
[182]     content: "";
[183]     height: 26px;
[184]     width: 26px;
[185]     left: 4px;
[186]     bottom: 4px;
[187]     background-color: white;
[188]     -webkit-transition: .4s;
[189]     transition: .4s;
[190] }
[191]
[192] input:checked+.slider {
[193]     background-color: #2196F3;
[194] }
[195]
[196] input:focus+.slider {
[197]     box-shadow: 0 0 1px #2196F3;
[198] }
[199]
[200] input:checked+.slider:before {
[201]     -webkit-transform: translateX(26px);
[202]     -ms-transform: translateX(26px);
[203]     transform: translateX(26px);
[204] }
[205]
[206] .slider.round {
[207]     border-radius: 34px;
[208] }
[209]
[210]

```

```

[211] .slider.round:before {
[212]   border-radius: 50%;
[213] }

```

Appendix 7: main.cpp

```

[1]  #include <Arduino.h>
[2]  #include <WiFi.h>
[3]  #include <ESPAsyncWebServer.h>
[4]  #include <vector>
[5]  #include <ESPmDNS.h>
[6]
[7]  #include "Actuator.h"
[8]  #include "JsonHandler.h"
[9]  #include "InverseKinematics.h"
[10] #include "SR518.h"
[11] #include "GaitHandler.h"
[12]
[13] #define RX2 16
[14] #define TX2 17
[15] #define EN2 15
[16]
[17] #define CoxaLength 55
[18] #define FemurLength 90
[19] #define TibialLength 140
[20] #define Start_Gate 8
[21]
[22] #define FORWARD_GAIT "forward"
[23] #define BACK_GAIT "back"
[24] #define LEFT_GAIT "left"
[25] #define RIGHT_GAIT "right"
[26] #define IDLE_GAIT "idle"
[27] #define ROTATE_LEFT_GAIT "rotateLeft"
[28] #define ROTATE_RIGHT_GAIT "rotateRight"
[29] #define STAIRS_UP_GAIT "stairsUp"
[30] #define STAIRS_DOWN_GAIT "stairsDown"
[31]
[32] AsyncWebServer server(80);
[33] vector<Actuator> actuators;
[34] vector<Actuator> leg_1_actuators;
[35] vector<Actuator> leg_2_actuators;
[36] vector<Actuator> leg_3_actuators;
[37] vector<Actuator> leg_4_actuators;
[38] size_t size = 4096;
[39] int height = 0;
[40] int pitch = 0; // offset between front (4,1) and back legs (3,2)
[41] int roll = 0; // offset between left (3,4) and right legs (1,2)
[42]
[43] String gait = IDLE_GAIT;
[44] String mode = "manual";
[45] String direction = "idle";
[46]
[47] void setup()
[48] {
[49]   //Initiate Serial
[50]   Serial.begin(115200);
[51]   Serial2.begin(RX2, TX2);
[52]   Hub.setSerial(&Serial2);
[53]   Hub.begin(500000, EN2);
[54]   delay(100);
[55]
[56]   //Initiate SPIFFS
[57]   if (!SPIFFS.begin(true))
[58]   {
[59]     Serial.println("Error: mounting SPIFFS");
[60]     return;
[61]   }
[62]   delay(100);
[63]
[64]   //Getting Network Credentials
[65]   String _ssid = JsonHandler::getElement("ssid", "credentials.json");
[66]   String _password = JsonHandler::getElement("password", "credentials.json");
[67]   const char *_ssid = _ssid.c_str();
[68]   const char *_password = _password.c_str();
[69]   delay(100);
[70]
[71]   //Initiating WiFi
[72]   WiFi.begin(_ssid, _password);
[73]   Serial.print("Connecting to WiFi");
[74]   while (WiFi.status() != WL_CONNECTED)
[75]   {
[76]     delay(100);
[77]     Serial.print(".");
[78]   }
[79]   Serial.println("");
[80]   Serial.println(WiFi.localIP());
[81]   delay(100);
[82]
[83]   Initialise mDNS
[84]   if (!mDNS.begin("qrp"))
[85]   {
[86]     Serial.println("Error starting mDNS");
[87]     return;
[88]   }
[89]   delay(100);
[90]
[91]   //Setting actuators
[92]   actuators = JsonHandler::getJsonActuators();
[93]
[94]   // Leg Actuators
[95]   for (Actuator servo : actuators)
[96]   {
[97]     if (servo.getLeg() == 1)
[98]       leg_1_actuators.push_back(servo);
[99]     else if (servo.getLeg() == 2)
[100]       leg_2_actuators.push_back(servo);
[101]     else if (servo.getLeg() == 3)
[102]       leg_3_actuators.push_back(servo);
[103]     else if (servo.getLeg() == 4)
[104]       leg_4_actuators.push_back(servo);

```

```

[105] }
[106]
[107] //Starting Server
[108] server.on("/", HTTP_GET, [(AsyncWebServerRequest *request) {
[109]     request->send(SPIFFS, "/index.html", "text/html");
[110] });
[111]
[112] server.on("/js/popper.min.js", HTTP_GET, [(AsyncWebServerRequest *request) {
[113]     request->send(SPIFFS, "/js/popper.min.js", "text/javascript");
[114] });
[115]
[116] server.on("/js/bootstrap.min.js", HTTP_GET, [(AsyncWebServerRequest *request) {
[117]     request->send(SPIFFS, "/js/bootstrap.min.js", "text/javascript");
[118] });
[119]
[120] server.on("/js/jquery.min.js", HTTP_GET, [(AsyncWebServerRequest *request) {
[121]     request->send(SPIFFS, "/js/jquery.min.js", "text/javascript");
[122] });
[123]
[124] server.on("/css/bootstrap.min.css", HTTP_GET, [(AsyncWebServerRequest *request) {
[125]     request->send(SPIFFS, "/css/bootstrap.min.css", "text/css");
[126] });
[127]
[128] server.on("/js/script.js", HTTP_GET, [(AsyncWebServerRequest *request) {
[129]     request->send(SPIFFS, "/js/script.js", "text/javascript");
[130] });
[131]
[132] server.on("/js/nipplejs.js", HTTP_GET, [(AsyncWebServerRequest *request) {
[133]     request->send(SPIFFS, "/js/nipplejs.js", "text/javascript");
[134] });
[135]
[136] server.on("/data/actuator.json", HTTP_GET, [(AsyncWebServerRequest *request) {
[137]     request->send(200, "application/json", JsonHandler::updateActuators(actuators));
[138] });
[139]
[140] server.on("/save", HTTP_POST, [(AsyncWebServerRequest *request) {
[141]     AsyncWebParameter *p_id = request->getParam("id");
[142]     AsyncWebParameter *p_min_pos = request->getParam("min_pos");
[143]     AsyncWebParameter *p_max_pos = request->getParam("max_pos");
[144]     request->send(200);
[145]
[146]     int id = p_id->value().toInt();
[147]     int min_pos = p_min_pos->value().toInt();
[148]     int max_pos = p_max_pos->value().toInt();
[149]     actuators[id - 1].setMinPosition(min_pos);
[150]     actuators[id - 1].setMaxPosition(max_pos);
[151]     Hub.setPosLimitMin(id, min_pos);
[152]     Hub.setPosLimitMax(id, max_pos);
[153]     JsonHandler::saveActuators(actuators);
[154] });
[155]
[156] server.on("/move", HTTP_POST, [(AsyncWebServerRequest *request) {
[157]     AsyncWebParameter *p_id = request->getParam("id");
[158]     AsyncWebParameter *p_pos = request->getParam("pos");
[159]     request->send(200);
[160]     int id = p_id->value().toInt();
[161]     int pos = p_pos->value().toInt();
[162]     Hub.move(id, pos);
[163] });
[164]
[165] server.on("/pos", HTTP_POST, [(AsyncWebServerRequest *request) {
[166]     AsyncWebParameter *p_id = request->getParam("id");
[167]     AsyncWebParameter *p_x = request->getParam("x");
[168]     AsyncWebParameter *p_y = request->getParam("y");
[169]     AsyncWebParameter *p_z = request->getParam("z");
[170]     request->send(200);
[171]
[172]     int id = p_id->value().toInt();
[173]     int x = p_x->value().toInt();
[174]     int y = p_y->value().toInt();
[175]     int z = p_z->value().toInt();
[176]     mode = "manual";
[177]
[178]     vector<Actuator> leg_actuators;
[179]
[180]     for (Actuator servo : actuators)
[181]     {
[182]         if (servo.getLeg() == id)
[183]         {
[184]             leg_actuators.push_back(servo);
[185]         }
[186]     }
[187]     GaithHandler::LiftLeg(leg_actuators, height, x, y, z);
[188] });
[189]
[190] server.on("/gait", HTTP_POST, [(AsyncWebServerRequest *request) {
[191]     request->send(200);
[192]
[193]     AsyncWebParameter *p_direction = request->getParam("direction");
[194]     AsyncWebParameter *p_mode = request->getParam("mode");
[195]     direction = p_direction->value();
[196]     mode = p_mode->value();
[197]
[198]     if (direction == "idle")
[199]     {
[200]         gait = IDLE_GAIT;
[201]     }
[202]     else if (direction == "up")
[203]     {
[204]         if (mode == "directional")
[205]         {
[206]             gait = FORWARD_GAIT;
[207]         }
[208]         else if (mode == "rotational")
[209]         {
[210]             gait = FORWARD_GAIT;
[211]         }
[212]         else if (mode == "stairclimbing")
[213]         {
[214]             gait = STAIRS_UP_GAIT;
[215]         }
[216]     }
[217]     else if (direction == "down")
[218]     {
[219]         if (mode == "directional")
[220]         {
[221]             gait = BACK_GAIT;
[222]         }
[223]         else if (mode == "rotational")
[224]         {

```

```

[225]         gait = BACK_GAIT;
[226]     }
[227]     else if (mode == "stairclimbing")
[228]     {
[229]         gait = STAIRS_DOWN_GAIT;
[230]     }
[231] }
[232] else if (direction == "left")
[233] {
[234]     if (mode == "directional")
[235]     {
[236]         gait = LEFT_GAIT;
[237]     }
[238]     else if (mode == "rotational")
[239]     {
[240]         gait = ROTATE_LEFT_GAIT;
[241]     }
[242]     else if (mode == "stairclimbing")
[243]     {
[244]         gait = IDLE_GAIT;
[245]     }
[246] }
[247] else if (direction == "right")
[248] {
[249]     if (mode == "directional")
[250]     {
[251]         gait = RIGHT_GAIT;
[252]     }
[253]     else if (mode == "rotational")
[254]     {
[255]         gait = ROTATE_RIGHT_GAIT;
[256]     }
[257]     else if (mode == "stairclimbing")
[258]     {
[259]         gait = IDLE_GAIT;
[260]     }
[261] }
[262] });
[263]
[264] server.on("/orientation", HTTP_POST, [(AsyncWebServerRequest *request) {
[265]     request->send(200);
[266]
[267]     AsyncWebParameter *p_roll = request->getParam("roll");
[268]     AsyncWebParameter *p_pitch = request->getParam("pitch");
[269]     AsyncWebParameter *p_height = request->getParam("height");
[270]
[271]     roll = p_roll->value().toInt();
[272]     pitch = p_pitch->value().toInt();
[273]     height = p_height->value().toInt();
[274] });
[275]
[276] server.begin();
[277] }
[278]
[279] void loop()
[280] {
[281]     if (mode != "manual")
[282]     {
[283]         GaitHandler::ExecuteGait(actuators, gait, height, pitch, roll);
[284]     }
[285] }

```

Appendix: 8: GaitHandler.cpp / .h

```

[1] #include "GaitHandler.h"
[2]
[3] void GaitHandler::LiftLeg(vector<Actuator> leg_actuators, int z_offset, int x, int y, int z)
[4] {
[5]     int coxaId, femurId, tibiaId;
[6]
[7]     for (Actuator servo : leg_actuators)
[8]     {
[9]         if (servo.getJoint() == 1)
[10]             coxaId = servo.getId();
[11]         else if (servo.getJoint() == 2)
[12]             femurId = servo.getId();
[13]         else if (servo.getJoint() == 3)
[14]             tibiaId = servo.getId();
[15]     }
[16]
[17]     //Lift-UP Leg
[18]     double femur_angle_up = IK.getFemurAngle(x, y, 1);
[19]     double tibia_angle_up = IK.getTibiaAngle(x, y, 1);
[20]     double coxa_angle_current = Hub.getPosition(coxaId);
[21]
[22]     Hub.moveLeg(leg_actuators, coxa_angle_current, femur_angle_up, tibia_angle_up);
[23]     delay(TIMEOUT);
[24]
[25]     //Transfer Leg
[26]     double coxa_angle_down = IK.getCoxaAngle(x, y);
[27]     Hub.moveLeg(leg_actuators, coxa_angle_down, femur_angle_up, tibia_angle_up);
[28]     delay(TIMEOUT);
[29]
[30]     // Put Down
[31]     double femur_angle_down = IK.getFemurAngle(x, y, z + z_offset);
[32]     double tibia_angle_down = IK.getTibiaAngle(x, y, z + z_offset);
[33]     Hub.moveLeg(leg_actuators, coxa_angle_down, femur_angle_down, tibia_angle_down);
[34] }
[35]
[36] void GaitHandler::ShiftLegs(vector<Actuator> actuators, int z_offset, int pitch, int roll, int x_1, int y_1, int z_1, int x_2, int y_2, int z_2, int
x_3, int y_3, int z_3, int x_4, int y_4, int z_4)
[37] {
[38]     int z_1_adj = z_1 + z_offset + pitch + roll;
[39]     int z_2_adj = z_2 + z_offset - pitch + roll;
[40]     int z_3_adj = z_3 + z_offset - pitch - roll;
[41]     int z_4_adj = z_4 + z_offset + pitch - roll;
[42]
[43]     double coxa_1_angle = IK.getCoxaAngle(x_1, y_1);
[44]     double femur_1_angle = IK.getFemurAngle(x_1, y_1, z_1_adj);
[45]     double tibia_1_angle = IK.getTibiaAngle(x_1, y_1, z_1_adj);
[46]

```

```

[47]     double coxa_2_angle = IK.getCoxaAngle(x_2, y_2);
[48]     double femur_2_angle = IK.getFemurAngle(x_2, y_2, z_2_adj);
[49]     double tibia_2_angle = IK.getTibiaAngle(x_2, y_2, z_2_adj);
[50]
[51]     double coxa_3_angle = IK.getCoxaAngle(x_3, y_3);
[52]     double femur_3_angle = IK.getFemurAngle(x_3, y_3, z_3_adj);
[53]     double tibia_3_angle = IK.getTibiaAngle(x_3, y_3, z_3_adj);
[54]
[55]     double coxa_4_angle = IK.getCoxaAngle(x_4, y_4);
[56]     double femur_4_angle = IK.getFemurAngle(x_4, y_4, z_4_adj);
[57]     double tibia_4_angle = IK.getTibiaAngle(x_4, y_4, z_4_adj);
[58]
[59]     Hub.moveAll(actuators, coxa_1_angle, femur_1_angle, tibia_1_angle, coxa_2_angle, femur_2_angle, tibia_2_angle, coxa_3_angle, femur_3_angle,
[60] tibia_3_angle, coxa_4_angle, femur_4_angle, tibia_4_angle);
[61] }
[62]
[63] void GaitHandler::ExecuteGait(vector<Actuator> actuators, String gait_name, int z_offset, int pitch, int roll)
[64] {
[65]     double coxa_1_angle = 0.0;
[66]     double coxa_2_angle = 0.0;
[67]     double coxa_3_angle = 0.0;
[68]     double coxa_4_angle = 0.0;
[69]
[70]     double femur_1_angle = 0.0;
[71]     double femur_2_angle = 0.0;
[72]     double femur_3_angle = 0.0;
[73]     double femur_4_angle = 0.0;
[74]
[75]     double tibia_1_angle = 0.0;
[76]     double tibia_2_angle = 0.0;
[77]     double tibia_3_angle = 0.0;
[78]     double tibia_4_angle = 0.0;
[79]
[80]     int x_1 = 0;
[81]     int x_2 = 0;
[82]     int x_3 = 0;
[83]     int x_4 = 0;
[84]
[85]     int y_1 = 0;
[86]     int y_2 = 0;
[87]     int y_3 = 0;
[88]     int y_4 = 0;
[89]
[90]     int z_1 = 0;
[91]     int z_2 = 0;
[92]     int z_3 = 0;
[93]     int z_4 = 0;
[94]
[95]     int type_1 = 1;
[96]     int type_2 = 1;
[97]     int type_3 = 1;
[98]     int type_4 = 1;
[99]
[100]     bool shiftLegs = false;
[101]
[102]     vector<vector<Step>> gait = JsonHandler::readGait(gait_name);
[103]
[104]     vector<Actuator> leg_1_actuators;
[105]     vector<Actuator> leg_2_actuators;
[106]     vector<Actuator> leg_3_actuators;
[107]     vector<Actuator> leg_4_actuators;
[108]
[109]     for (Actuator servo : actuators)
[110]     {
[111]         if (servo.getLeg() == 1)
[112]             leg_1_actuators.push_back(servo);
[113]         else if (servo.getLeg() == 2)
[114]             leg_2_actuators.push_back(servo);
[115]         else if (servo.getLeg() == 3)
[116]             leg_3_actuators.push_back(servo);
[117]         else if (servo.getLeg() == 4)
[118]             leg_4_actuators.push_back(servo);
[119]     }
[120]
[121]     for (vector<Step> sequence : gait)
[122]     {
[123]         for (Step step : sequence)
[124]         {
[125]             if (step.getLegId() == 1)
[126]             {
[127]                 x_1 = step.getX();
[128]                 y_1 = step.getY();
[129]                 z_1 = step.getZ();
[130]                 type_1 = step.getType();
[131]
[132]                 if (type_1 == 2)
[133]                 {
[134]                     LiftLeg(leg_1_actuators, z_offset, x_1, y_1, z_1 + pitch + roll);
[135]                 }
[136]                 else if (step.getLegId() == 2)
[137]                 {
[138]                     x_2 = step.getX();
[139]                     y_2 = step.getY();
[140]                     z_2 = step.getZ();
[141]                     type_2 = step.getType();
[142]
[143]                     if (type_2 == 2)
[144]                     {
[145]                         LiftLeg(leg_2_actuators, z_offset, x_2, y_2, z_2 - pitch + roll);
[146]                     }
[147]                 }
[148]                 else if (step.getLegId() == 3)
[149]                 {
[150]                     x_3 = step.getX();
[151]                     y_3 = step.getY();
[152]                     z_3 = step.getZ();
[153]                     type_3 = step.getType();
[154]
[155]                     if (type_3 == 2)
[156]                     {
[157]                         LiftLeg(leg_3_actuators, z_offset, x_3, y_3, z_3 - pitch - roll);
[158]                     }
[159]                 }
[160]                 else if (step.getLegId() == 4)
[161]                 {
[162]                     x_4 = step.getX();
[163]                     y_4 = step.getY();
[164]                     z_4 = step.getZ();
[165]

```



```

[166]         type_4 = step.getType();
[167]
[168]         if (type_4 == 2)
[169]         {
[170]             LiftLeg(leg_4.actuators, z_offset, x_4, y_4, z_4 + pitch - roll);
[171]         }
[172]     }
[173] }
[174] ShiftLegs(actuators, z_offset, pitch, roll, x_1, y_1, z_1, x_2, y_2, z_2, x_3, y_3, z_3, x_4, y_4, z_4);
[175] delay(TIMEOUT);
[176] }
[177] }

[178] #pragma once
[179] #include "Arduino.h"
[180] #include "Step.h"
[181] #include "JsonHandler.h"
[182] #include "SR518.h"
[183] #include "InverseKinematics.h"
[184]
[185] #define TIMEOUT 250
[186]
[187] class GaitHandler
[188] {
[189] public:
[190]     static void ExecuteGait(vector<Actuator> actuators, String gait_name, int z_offset, int pitch_offset, int roll_offset);
[191]     static void LiftLeg(vector<Actuator> leg_actuators, int z_offset, int x, int y, int z);
[192]     static void ShiftLegs(vector<Actuator> actuators, int z_offset, int pitch, int roll, int x_1, int y_1, int z_1, int x_2, int y_2, int z_2, int x_3,
int y_3, int z_3, int x_4, int y_4, int z_4);
[193] private:
[194]     static vector<Actuator> getLegActuators(vector<Actuator> actuators, int leg_id);
[195] };

```

Appendix 9: InverseKinematics.cpp / .h (IK)

```

[1] #include <Arduino.h>
[2] #include <string>
[3] #include <cmath>
[4] #include "InverseKinematics.h"
[5] using namespace std;
[6]
[7] double InverseKinematics::getCoxaAngle(int x, int y)
[8] {
[9]     double x = (double)x;
[10]     double y = (double)y;
[11]     double coxa_angle = atan(x / y);
[12]     double result = coxa_angle * 180 / PI;
[13]     return result;
[14] }
[15]
[16] double InverseKinematics::getFemurAngle(int x, int y, int z)
[17] {
[18]     double x = (double)x;
[19]     double y = (double)y;
[20]     double z = (double)z;
[21]     double l1 = sqrt(pow(x, 2) + pow(y, 2));
[22]     double l = sqrt(pow(z, 2) + pow(l1 - COXA_LENGTH, 2));
[23]     double femur_angle_1 = acos(z / l);
[24]     double femur_angle_2 = acos((pow(TIBIA_LENGTH, 2) - pow(FEMUR_LENGTH, 2) - pow(l, 2)) / (-2 * FEMUR_LENGTH * l));
[25]     double femur_angle = femur_angle_1 + femur_angle_2;
[26]     double result = femur_angle * 180 / PI;
[27]     return result;
[28] }
[29]
[30] double InverseKinematics::getTibiaAngle(int x, int y, int z)
[31] {
[32]     double x = (double)x;
[33]     double y = (double)y;
[34]     double z = (double)z;
[35]     double l1 = sqrt(pow(x, 2) + pow(y, 2));
[36]     double l = sqrt(pow(z, 2) + pow(l1 - COXA_LENGTH, 2));
[37]     double tibia_angle = acos((pow(l, 2) - pow(TIBIA_LENGTH, 2) - pow(FEMUR_LENGTH, 2)) / (-2 * TIBIA_LENGTH * FEMUR_LENGTH));
[38]     double result = tibia_angle * 180 / PI;
[39]     return result;
[40] }
[41]
[42] InverseKinematics IK;

[43] #pragma once
[44] #ifndef InverseKinematics_h
[45] #define InverseKinematics_h
[46]
[47] #include <string>
[48] #include <cmath>
[49] using namespace std;
[50]
[51] #define COXA_LENGTH 55.00
[52] #define FEMUR_LENGTH 90.00
[53] #define TIBIA_LENGTH 140.00
[54] #define PI 3.14159265358979323846
[55]
[56] class InverseKinematics
[57] {
[58] public:
[59]     double getCoxaAngle(int x, int y);
[60]     double getFemurAngle(int x, int y, int z);
[61]     double getTibiaAngle(int x, int y, int z);
[62] private:
[63] };
[64]
[65] extern InverseKinematics IK;
[66]
[67] #endif InverseKinematics_h

```

Appendix 10: SR518.cpp /.h (Hub)

```
[1] #if defined(ARDUINO) && ARDUINO >= 100 // Arduino IDE Version
[2] #include "Arduino.h"
[3] #else
[4] #include "WProgram.h"
[5] #endif
[6] #include "SR518.h"
[7]
[8] int SR518::read_error()
[9] {
[10]     Time_Counter = 0;
[11]     while ((serialPort->available() < 5) & (Time_Counter < TIME_OUT))
[12]     { // Wait for Data
[13]         Time_Counter++;
[14]         delayMicroseconds(1000);
[15]     }
[16]
[17]     while (serialPort->available() > 0)
[18]     {
[19]         if ((serialPort->read() == START_FLAG) & (serialPort->peek() == START_FLAG))
[20]         {
[21]             serialPort->read(); //Checks for begining of Status Packet 0xFF 0xFF
[22]             serialPort->read(); // START_FLAG
[23]             serialPort->read(); // ID
[24]             Error_Byte = serialPort->read(); // LENGTH
[25]             Serial.print("Error: "); // ERROR
[26]             Serial.println(Error_Byte);
[27]             return (Error_Byte);
[28]         }
[29]     }
[30]     return (-1); // No Response
[31] }
[32]
[33] void SR518::setSerial(HardwareSerial *sPort)
[34] {
[35]     serialPort = sPort;
[36] }
[37]
[38] void SR518::begin(long baud, unsigned char directionPin)
[39] {
[40]     Direction_Pin = directionPin;
[41]     pinMode(Direction_Pin, OUTPUT);
[42]     serialPort->begin(baud);
[43] }
[44]
[45] int SR518::move(byte id, double position_angle)
[46] {
[47]     int position = position_angle * POSITION_RESOLUTION_H;
[48]     char pos_h = position >> 8;
[49]     char pos_l = position % 256;
[50]     byte checksum = ~lowByte(id + POSITION_LENGTH + WRITE + GOAL_POSITION_L + pos_l + pos_h);
[51]
[52]     digitalWrite(Direction_Pin, TX_MODE);
[53]     serialPort->write(START_FLAG);
[54]     serialPort->write(START_FLAG);
[55]     serialPort->write(id);
[56]     serialPort->write(POSITION_LENGTH); //Length
[57]     serialPort->write(WRITE); //Instruction
[58]     serialPort->write(GOAL_POSITION_L); //First Address of data write
[59]     serialPort->write(pos_l); //First data write
[60]     serialPort->write(pos_h); //Second data write
[61]     serialPort->write(checksum);
[62]     serialPort->flush();
[63]     digitalWrite(Direction_Pin, RX_MODE);
[64]     return (read_error());
[65] }
[66]
[67] int SR518::moveLeg(vector<Actuator> leg_actuators, double coxa_pos_angle, double femur_pos_angle, double tibia_pos_angle)
[68] {
[69]     int speed = MAX_SPEED;
[70]     char speed_h = speed >> 8;
[71]     char speed_l = speed % 256;
[72]
[73]     char servo_coxa_id = 0;
[74]     char servo_femur_id = 0;
[75]     char servo_tibia_id = 0;
[76]
[77]     double coxa_position_calibrated = 0.0;
[78]     double femur_position_calibrated = 0.0;
[79]     double tibia_position_calibrated = 0.0;
[80]
[81]     for (Actuator leg_servo : leg_actuators)
[82]     {
[83]         if (leg_servo.getJoint() == 1)
[84]         {
[85]             if (leg_servo.getInverted() == 1)
[86]             {
[87]                 coxa_position_calibrated = leg_servo.getMaxPosition() - _coxa_pos_angle;
[88]             }
[89]             else
[90]             {
[91]                 coxa_position_calibrated = _coxa_pos_angle + leg_servo.getMinPosition();
[92]             }
[93]             servo_coxa_id = leg_servo.getId();
[94]         }
[95]         if (leg_servo.getJoint() == 2)
[96]         {
[97]             femur_position_calibrated = leg_servo.getMaxPosition() - (_femur_pos_angle - 60);
[98]             servo_femur_id = leg_servo.getId();
[99]         }
[100]         if (leg_servo.getJoint() == 3)
[101]         {
[102]             tibia_position_calibrated = _tibia_pos_angle + leg_servo.getMinPosition();
[103]             servo_tibia_id = leg_servo.getId();
[104]         }
[105]     }
[106]
[107]     int coxa_position = coxa_position_calibrated * POSITION_RESOLUTION_H;
[108]     char coxa_pos_h = coxa_position >> 8;
[109]     char coxa_pos_l = coxa_position % 256;
[110]
[111]     int femur_position = femur_position_calibrated * POSITION_RESOLUTION_H;
[112]     char femur_pos_h = femur_position >> 8;
[113]     char femur_pos_l = femur_position % 256;
```

```

[114]
[115] int tibia_position = tibia_position_calibrated * POSITION_RESOLUTION_H;
[116] char tibia_pos_h = tibia_position >> 8;
[117] char tibia_pos_l = tibia_position % 256;
[118]
[119] byte Lenght = (READ_POS_LENGTH + 1) * 3 + 4; //Check Point ???
[120]
[121] byte checksum = ~lowByte(BROADCAST_ID + Lenght + SYNC_WRITE + 0x1E + READ_POS_LENGTH +
[122]     servo_coxa_id + coxa_pos_l + coxa_pos_h + speed_l + speed_h +
[123]     servo_femur_id + femur_pos_l + femur_pos_h + speed_l + speed_h +
[124]     servo_tibia_id + tibia_pos_l + tibia_pos_h + speed_l + speed_h);
[125]
[126] digitalWrite(Direction_Pin, TX_MODE);
[127] serialPort->write(START_FLAG);
[128] serialPort->write(START_FLAG);
[129] serialPort->write(BROADCAST_ID); // 0xFE
[130] serialPort->write(Lenght); // (DataLength + 1) * NumberOfServos + 4
[131] serialPort->write(SYNC_WRITE); // Instruction 0x83
[132] serialPort->write(0x1E);
[133] serialPort->write(READ_POS_LENGTH);
[134]
[135] serialPort->write(servo_coxa_id); //Length of data write
[136] serialPort->write(coxa_pos_l); //First Id
[137] serialPort->write(coxa_pos_h); //First data write
[138] serialPort->write(speed_l);
[139] serialPort->write(speed_h);
[140]
[141] serialPort->write(servo_femur_id); //Length of data write
[142] serialPort->write(femur_pos_l); //First Id
[143] serialPort->write(femur_pos_h); //First data write
[144] serialPort->write(speed_l);
[145] serialPort->write(speed_h);
[146]
[147] serialPort->write(servo_tibia_id); //Length of data write
[148] serialPort->write(tibia_pos_l); //First Id
[149] serialPort->write(tibia_pos_h); //First data write
[150] serialPort->write(speed_l);
[151] serialPort->write(speed_h);
[152]
[153] serialPort->write(checksum);
[154] serialPort->flush();
[155] digitalWrite(Direction_Pin, RX_MODE);
[156] return (read_error());
[157] }
[158]
[159]
[160] int SR518::moveAll(vector<Actuator> actuators, double coxa_1_angle, double femur_1_angle, double tibia_1_angle, double coxa_2_angle, double
femur_2_angle, double tibia_2_angle, double coxa_3_angle, double femur_3_angle, double tibia_3_angle, double coxa_4_angle, double femur_4_angle, double
tibia_4_angle)
[161] {
[162]     int speed = MAX_SPEED;
[163]     char speed_h = speed >> 8;
[164]     char speed_l = speed % 256;
[165]
[166]     //LEG 1 Variables
[167]     char servo_1_coxa_id = 0;
[168]     char servo_1_femur_id = 0;
[169]     char servo_1_tibia_id = 0;
[170]
[171]     double coxa_1_position_calibrated = 0.0;
[172]     double femur_1_position_calibrated = 0.0;
[173]     double tibia_1_position_calibrated = 0.0;
[174]
[175]     //LEG 2 Variables
[176]     char servo_2_coxa_id = 0;
[177]     char servo_2_femur_id = 0;
[178]     char servo_2_tibia_id = 0;
[179]
[180]     double coxa_2_position_calibrated = 0.0;
[181]     double femur_2_position_calibrated = 0.0;
[182]     double tibia_2_position_calibrated = 0.0;
[183]
[184]     //LEG 3 Variables
[185]     char servo_3_coxa_id = 0;
[186]     char servo_3_femur_id = 0;
[187]     char servo_3_tibia_id = 0;
[188]
[189]     double coxa_3_position_calibrated = 0.0;
[190]     double femur_3_position_calibrated = 0.0;
[191]     double tibia_3_position_calibrated = 0.0;
[192]
[193]     //LEG 4 Variables
[194]     char servo_4_coxa_id = 0;
[195]     char servo_4_femur_id = 0;
[196]     char servo_4_tibia_id = 0;
[197]
[198]     double coxa_4_position_calibrated = 0.0;
[199]     double femur_4_position_calibrated = 0.0;
[200]     double tibia_4_position_calibrated = 0.0;
[201]
[202]     for (Actuator servo : actuators)
[203]     {
[204]         //LEG 1 Actuator Data
[205]         if (servo.getJoint() == 1 && servo.getLeg() == 1)
[206]         {
[207]             if (servo.getInverted() == 1)
[208]             {
[209]                 coxa_1_position_calibrated = servo.getMaxPosition() - coxa_1_angle;
[210]             }
[211]             else
[212]             {
[213]                 coxa_1_position_calibrated = coxa_1_angle + servo.getMinPosition();
[214]             }
[215]             servo_1_coxa_id = servo.getId();
[216]         }
[217]         if (servo.getJoint() == 2 && servo.getLeg() == 1)
[218]         {
[219]             femur_1_position_calibrated = servo.getMaxPosition() - (femur_1_angle - 60);
[220]             servo_1_femur_id = servo.getId();
[221]         }
[222]         if (servo.getJoint() == 3 && servo.getLeg() == 1)
[223]         {
[224]             tibia_1_position_calibrated = tibia_1_angle + servo.getMinPosition();
[225]             servo_1_tibia_id = servo.getId();
[226]         }
[227]
[228]         //LEG 2 Actuator Data
[229]         if (servo.getJoint() == 1 && servo.getLeg() == 2)
[230]         {
[231]             if (servo.getInverted() == 1)

```

```

[232] {
[233]     coxa_2_position_calibrated = servo.getMaxPosition() - coxa_2_angle;
[234] }
[235] else
[236] {
[237]     coxa_2_position_calibrated = coxa_2_angle + servo.getMinPosition();
[238] }
[239] servo_2_coxa_id = servo.getId();
[240] }
[241] if (servo.getJoint() == 2 && servo.getLeg() == 2)
[242] {
[243]     femur_2_position_calibrated = servo.getMaxPosition() - (femur_2_angle - 60);
[244]     servo_2_femur_id = servo.getId();
[245] }
[246] if (servo.getJoint() == 3 && servo.getLeg() == 2)
[247] {
[248]     tibia_2_position_calibrated = tibia_2_angle + servo.getMinPosition();
[249]     servo_2_tibia_id = servo.getId();
[250] }
[251]
[252] //LEG 3 Actuator Data
[253] if (servo.getJoint() == 1 && servo.getLeg() == 3)
[254] {
[255]     if (servo.getInverted() == 1)
[256]     {
[257]         coxa_3_position_calibrated = servo.getMaxPosition() - coxa_3_angle;
[258]     }
[259]     else
[260]     {
[261]         coxa_3_position_calibrated = coxa_3_angle + servo.getMinPosition();
[262]     }
[263]     servo_3_coxa_id = servo.getId();
[264] }
[265] if (servo.getJoint() == 2 && servo.getLeg() == 3)
[266] {
[267]     femur_3_position_calibrated = servo.getMaxPosition() - (femur_3_angle - 60);
[268]     servo_3_femur_id = servo.getId();
[269] }
[270] if (servo.getJoint() == 3 && servo.getLeg() == 3)
[271] {
[272]     tibia_3_position_calibrated = tibia_3_angle + servo.getMinPosition();
[273]     servo_3_tibia_id = servo.getId();
[274] }
[275]
[276] //LEG 4 Actuator Data
[277] if (servo.getJoint() == 1 && servo.getLeg() == 4)
[278] {
[279]     if (servo.getInverted() == 1)
[280]     {
[281]         coxa_4_position_calibrated = servo.getMaxPosition() - coxa_4_angle;
[282]     }
[283]     else
[284]     {
[285]         coxa_4_position_calibrated = coxa_4_angle + servo.getMinPosition();
[286]     }
[287]     servo_4_coxa_id = servo.getId();
[288] }
[289] if (servo.getJoint() == 2 && servo.getLeg() == 4)
[290] {
[291]     femur_4_position_calibrated = servo.getMaxPosition() - (femur_4_angle - 60);
[292]     servo_4_femur_id = servo.getId();
[293] }
[294] if (servo.getJoint() == 3 && servo.getLeg() == 4)
[295] {
[296]     tibia_4_position_calibrated = tibia_4_angle + servo.getMinPosition();
[297]     servo_4_tibia_id = servo.getId();
[298] }
[299] }
[300]
[301] //LEG 1 Calibrated Position
[302] int coxa_1_position = coxa_1_position_calibrated * POSITION_RESOLUTION_H;
[303] char coxa_1_pos_h = coxa_1_position >> 8;
[304] char coxa_1_pos_l = coxa_1_position % 256;
[305]
[306] int femur_1_position = femur_1_position_calibrated * POSITION_RESOLUTION_H;
[307] char femur_1_pos_h = femur_1_position >> 8;
[308] char femur_1_pos_l = femur_1_position % 256;
[309]
[310] int tibia_1_position = tibia_1_position_calibrated * POSITION_RESOLUTION_H;
[311] char tibia_1_pos_h = tibia_1_position >> 8;
[312] char tibia_1_pos_l = tibia_1_position % 256;
[313]
[314] //LEG 2 Calibrated Position
[315] int coxa_2_position = coxa_2_position_calibrated * POSITION_RESOLUTION_H;
[316] char coxa_2_pos_h = coxa_2_position >> 8;
[317] char coxa_2_pos_l = coxa_2_position % 256;
[318]
[319] int femur_2_position = femur_2_position_calibrated * POSITION_RESOLUTION_H;
[320] char femur_2_pos_h = femur_2_position >> 8;
[321] char femur_2_pos_l = femur_2_position % 256;
[322]
[323] int tibia_2_position = tibia_2_position_calibrated * POSITION_RESOLUTION_H;
[324] char tibia_2_pos_h = tibia_2_position >> 8;
[325] char tibia_2_pos_l = tibia_2_position % 256;
[326]
[327] //LEG 3 Calibrated Position
[328] int coxa_3_position = coxa_3_position_calibrated * POSITION_RESOLUTION_H;
[329] char coxa_3_pos_h = coxa_3_position >> 8;
[330] char coxa_3_pos_l = coxa_3_position % 256;
[331]
[332] int femur_3_position = femur_3_position_calibrated * POSITION_RESOLUTION_H;
[333] char femur_3_pos_h = femur_3_position >> 8;
[334] char femur_3_pos_l = femur_3_position % 256;
[335]
[336] int tibia_3_position = tibia_3_position_calibrated * POSITION_RESOLUTION_H;
[337] char tibia_3_pos_h = tibia_3_position >> 8;
[338] char tibia_3_pos_l = tibia_3_position % 256;
[339]
[340] //LEG 4 Calibrated Position
[341] int coxa_4_position = coxa_4_position_calibrated * POSITION_RESOLUTION_H;
[342] char coxa_4_pos_h = coxa_4_position >> 8;
[343] char coxa_4_pos_l = coxa_4_position % 256;
[344]
[345] int femur_4_position = femur_4_position_calibrated * POSITION_RESOLUTION_H;
[346] char femur_4_pos_h = femur_4_position >> 8;
[347] char femur_4_pos_l = femur_4_position % 256;
[348]
[349] int tibia_4_position = tibia_4_position_calibrated * POSITION_RESOLUTION_H;
[350] char tibia_4_pos_h = tibia_4_position >> 8;
[351] char tibia_4_pos_l = tibia_4_position % 256;

```

```

[352]
[353] byte Lenght = (READ_POS_LENGTH + 1) * 12 + 4; //Check Point ???
[354]
[355] byte checksum = ~lowByte(BROADCAST_ID + Lenght + SYNC_WRITE + 0x1E + READ_POS_LENGTH +
[356] servo_1_coxa_id + coxa_1_pos_l + coxa_1_pos_h + speed_l + speed_h +
[357] servo_1_femur_id + femur_1_pos_l + femur_1_pos_h + speed_l + speed_h +
[358] servo_1_tibia_id + tibia_1_pos_l + tibia_1_pos_h + speed_l + speed_h +
[359] servo_2_coxa_id + coxa_2_pos_l + coxa_2_pos_h + speed_l + speed_h +
[360] servo_2_femur_id + femur_2_pos_l + femur_2_pos_h + speed_l + speed_h +
[361] servo_2_tibia_id + tibia_2_pos_l + tibia_2_pos_h + speed_l + speed_h +
[362] servo_3_coxa_id + coxa_3_pos_l + coxa_3_pos_h + speed_l + speed_h +
[363] servo_3_femur_id + femur_3_pos_l + femur_3_pos_h + speed_l + speed_h +
[364] servo_3_tibia_id + tibia_3_pos_l + tibia_3_pos_h + speed_l + speed_h +
[365] servo_4_coxa_id + coxa_4_pos_l + coxa_4_pos_h + speed_l + speed_h +
[366] servo_4_femur_id + femur_4_pos_l + femur_4_pos_h + speed_l + speed_h +
[367] servo_4_tibia_id + tibia_4_pos_l + tibia_4_pos_h + speed_l + speed_h);
[368]
[369] digitalWrite(Direction_Pin, TX_MODE);
[370]
[371] //Packet Header
[372] serialPort->write(START_FLAG);
[373] serialPort->write(START_FLAG);
[374] serialPort->write(BROADCAST_ID);
[375] serialPort->write(Lenght);
[376] serialPort->write(SYNC_WRITE);
[377] serialPort->write(0x1E);
[378] serialPort->write(READ_POS_LENGTH);
[379]
[380] //LEG 1 Packet Data
[381] serialPort->write(servo_1_coxa_id);
[382] serialPort->write(coxa_1_pos_l);
[383] serialPort->write(coxa_1_pos_h);
[384] serialPort->write(speed_l);
[385] serialPort->write(speed_h);
[386]
[387] serialPort->write(servo_1_femur_id);
[388] serialPort->write(femur_1_pos_l);
[389] serialPort->write(femur_1_pos_h);
[390] serialPort->write(speed_l);
[391] serialPort->write(speed_h);
[392]
[393] serialPort->write(servo_1_tibia_id);
[394] serialPort->write(tibia_1_pos_l);
[395] serialPort->write(tibia_1_pos_h);
[396] serialPort->write(speed_l);
[397] serialPort->write(speed_h);
[398]
[399] //LEG 2 Packet Data
[400] serialPort->write(servo_2_coxa_id);
[401] serialPort->write(coxa_2_pos_l);
[402] serialPort->write(coxa_2_pos_h);
[403] serialPort->write(speed_l);
[404] serialPort->write(speed_h);
[405]
[406] serialPort->write(servo_2_femur_id);
[407] serialPort->write(femur_2_pos_l);
[408] serialPort->write(femur_2_pos_h);
[409] serialPort->write(speed_l);
[410] serialPort->write(speed_h);
[411]
[412] serialPort->write(servo_2_tibia_id);
[413] serialPort->write(tibia_2_pos_l);
[414] serialPort->write(tibia_2_pos_h);
[415] serialPort->write(speed_l);
[416] serialPort->write(speed_h);
[417]
[418] //LEG 3 Packet Data
[419] serialPort->write(servo_3_coxa_id);
[420] serialPort->write(coxa_3_pos_l);
[421] serialPort->write(coxa_3_pos_h);
[422] serialPort->write(speed_l);
[423] serialPort->write(speed_h);
[424]
[425] serialPort->write(servo_3_femur_id);
[426] serialPort->write(femur_3_pos_l);
[427] serialPort->write(femur_3_pos_h);
[428] serialPort->write(speed_l);
[429] serialPort->write(speed_h);
[430]
[431] serialPort->write(servo_3_tibia_id);
[432] serialPort->write(tibia_3_pos_l);
[433] serialPort->write(tibia_3_pos_h);
[434] serialPort->write(speed_l);
[435] serialPort->write(speed_h);
[436]
[437] //LEG 4 Packet Data
[438] serialPort->write(servo_4_coxa_id);
[439] serialPort->write(coxa_4_pos_l);
[440] serialPort->write(coxa_4_pos_h);
[441] serialPort->write(speed_l);
[442] serialPort->write(speed_h);
[443]
[444] serialPort->write(servo_4_femur_id);
[445] serialPort->write(femur_4_pos_l);
[446] serialPort->write(femur_4_pos_h);
[447] serialPort->write(speed_l);
[448] serialPort->write(speed_h);
[449]
[450] serialPort->write(servo_4_tibia_id);
[451] serialPort->write(tibia_4_pos_l);
[452] serialPort->write(tibia_4_pos_h);
[453] serialPort->write(speed_l);
[454] serialPort->write(speed_h);
[455]
[456] //Packet End
[457] serialPort->write(checksum);
[458] serialPort->flush();
[459] digitalWrite(Direction_Pin, RX_MODE);
[460] return (read_error());
[461] }
[462]
[463] int SR518::setPosLimitMin(byte id, double position_angle)
[464] {
[465]     int position = position_angle * POSITION_RESOLUTION_H;
[466]     char pos_h = position >> 8;
[467]     char pos_l = position % 256;
[468]     byte checksum = ~lowByte(id + POSITION_LENGTH + WRITE + MIN_POSITION_L + pos_l + pos_h);
[469]
[470]     digitalWrite(Direction_Pin, TX_MODE);
[471]     serialPort->write(START_FLAG);

```

```

[472] serialPort->write(START_FLAG);
[473] serialPort->write(id);
[474] serialPort->write(POSITION_LENGTH);
[475] serialPort->write(WRITE);
[476] serialPort->write(MIN_POSITION_L);
[477] serialPort->write(pos_l);
[478] serialPort->write(pos_h);
[479] serialPort->write(checksum);
[480] serialPort->flush();
[481] digitalWrite(Direction_Pin, RX_MODE);
[482] return (read_error());
[483] }
[484]
[485] int SR518::setPosLimitMax(byte id, double position_angle)
[486] {
[487]     int position = position_angle * POSITION_RESOLUTION_H;
[488]     char pos_h = position >> 8;
[489]     char pos_l = position % 256;
[490]     byte checksum = ~lowByte(id + POSITION_LENGTH + WRITE + MAX_POSITION_L + pos_l + pos_h);
[491]
[492]     digitalWrite(Direction_Pin, TX_MODE);
[493]     serialPort->write(START_FLAG);
[494]     serialPort->write(START_FLAG);
[495]     serialPort->write(id);
[496]     serialPort->write(POSITION_LENGTH);
[497]     serialPort->write(WRITE);
[498]     serialPort->write(MAX_POSITION_L);
[499]     serialPort->write(pos_l);
[500]     serialPort->write(pos_h);
[501]     serialPort->write(checksum);
[502]     serialPort->flush();
[503]     digitalWrite(Direction_Pin, RX_MODE);
[504]     return (read_error());
[505] }
[506]
[507] int SR518::getTemperature(byte id)
[508] {
[509]     byte checksum = ~lowByte(id + TEMPERATURE_LENGTH + READ + PRESENT_TEMPERATURE + BYTE_READ);
[510]
[511]     digitalWrite(Direction_Pin, TX_MODE);
[512]     serialPort->write(START_FLAG);
[513]     serialPort->write(START_FLAG);
[514]     serialPort->write(id);
[515]     serialPort->write(TEMPERATURE_LENGTH);
[516]     serialPort->write(READ);
[517]     serialPort->write(PRESENT_TEMPERATURE);
[518]     serialPort->write(BYTE_READ);
[519]     serialPort->write(checksum);
[520]     serialPort->flush();
[521]     digitalWrite(Direction_Pin, RX_MODE);
[522]
[523]     Temperature_Byte = -1;
[524]     Time_Counter = 0;
[525]     while ((serialPort->available() < 6) & (Time_Counter < TIME_OUT))
[526]     {
[527]         Time_Counter++;
[528]         delayMicroseconds(1000);
[529]     }
[530]
[531]     while (serialPort->available() > 0)
[532]     {
[533]         if ((serialPort->read() == START_FLAG) & (serialPort->peek() == START_FLAG))
[534]         {
[535]             serialPort->read(); // Start Bytes
[536]             serialPort->read(); // Ax-12 ID
[537]             serialPort->read(); // Length
[538]             if ((Error_Byte = serialPort->read()) != 0) // Error
[539]                 return (Error_Byte * (-1));
[540]             Temperature_Byte = serialPort->read(); // Temperature
[541]         }
[542]     }
[543]     return (Temperature_Byte); // Returns the read temperature
[544] }
[545]
[546] double SR518::getVoltage(byte id)
[547] {
[548]     byte checksum = ~lowByte(id + VOLTAGE_LENGTH + READ + PRESENT_VOLTAGE + BYTE_READ);
[549]
[550]     digitalWrite(Direction_Pin, TX_MODE);
[551]     serialPort->write(START_FLAG);
[552]     serialPort->write(START_FLAG);
[553]     serialPort->write(id);
[554]     serialPort->write(VOLTAGE_LENGTH);
[555]     serialPort->write(READ);
[556]     serialPort->write(PRESENT_VOLTAGE);
[557]     serialPort->write(BYTE_READ);
[558]     serialPort->write(checksum);
[559]     serialPort->flush();
[560]     digitalWrite(Direction_Pin, RX_MODE);
[561]
[562]     Voltage_Byte = -1;
[563]     Time_Counter = 0;
[564]     while ((serialPort->available() < 6) & (Time_Counter < TIME_OUT))
[565]     {
[566]         Time_Counter++;
[567]         delayMicroseconds(1000);
[568]     }
[569]
[570]     while (serialPort->available() > 0)
[571]     {
[572]         if ((serialPort->read() == START_FLAG) & (serialPort->peek() == START_FLAG))
[573]         {
[574]             serialPort->read();
[575]             serialPort->read();
[576]             serialPort->read();
[577]             if ((Error_Byte = serialPort->read()) != 0)
[578]                 return (Error_Byte * (-1));
[579]             Voltage_Byte = serialPort->read();
[580]         }
[581]     }
[582]     return (double(Voltage_Byte) / 10);
[583] }
[584]
[585] double SR518::getPosition(byte id)
[586] {
[587]     byte checksum = ~lowByte(id + READ_POS_LENGTH + READ + PRESENT_POSITION_L + BYTE_READ_POS);
[588]
[589]     digitalWrite(Direction_Pin, TX_MODE);
[590]     serialPort->write(START_FLAG);
[591]     serialPort->write(START_FLAG);

```

```

[592] serialPort->write(id);
[593] serialPort->write(READ_POS_LENGTH);
[594] serialPort->write(READ);
[595] serialPort->write(PRESENT_POSITION_L);
[596] serialPort->write(BYTE_READ_POS);
[597] serialPort->write(checksum);
[598] serialPort->flush();
[599] digitalWrite(Direction_Pin, RX_MODE);
[600]
[601] Position_Long_Byte = -1;
[602] Time_Counter = 0;
[603] while ((serialPort->available() < 5) & (Time_Counter < TIME_OUT))
[604] {
[605]     Time_Counter++;
[606]     delayMicroseconds(1000);
[607] }
[608]
[609] while (serialPort->available() > 0)
[610] {
[611]     if ((serialPort->read() == START_FLAG) & (serialPort->peek() == START_FLAG))
[612]     {
[613]         serialPort->read();
[614]         serialPort->read();
[615]         serialPort->read();
[616]         if ((Error_Byte = serialPort->read()) != 0)
[617]             return (Error_Byte * (-1));
[618]
[619]         Position_Low_Byte = serialPort->read();
[620]         Position_High_Byte = serialPort->read();
[621]         Position_Long_Byte = Position_High_Byte << 8;
[622]         Position_Long_Byte = Position_Long_Byte + Position_Low_Byte;
[623]     }
[624] }
[625] return (double(Position_Long_Byte * POSITION_RESOLUTION_L));
[626] }
[627]
[628] SR518 Hub;

[629] #pragma once
[630]
[631] #ifndef SR518_h
[632] #define SR518_h
[633]
[634] //Instructions
[635] #define READ 2
[636] #define WRITE 3
[637] #define SYNC_WRITE 131
[638] #define ACTION 5
[639]
[640] //Data Lengths
[641] #define POSITION_LENGTH 5
[642] #define TEMPERATURE_LENGTH 4
[643] #define VOLTAGE_LENGTH 4
[644] #define SPEED_LENGTH 5
[645] #define READ_POS_LENGTH 4
[646]
[647] //Addresses
[648] #define MIN_POSITION_L 60
[649] #define MAX_POSITION_L 8
[650] #define GOAL_POSITION_L 30
[651] #define PRESENT_TEMPERATURE 43
[652] #define PRESENT_VOLTAGE 42
[653] #define PRESENT_SPEED_L 38
[654] #define PRESENT_POSITION_L 36
[655] #define ACCELERATING_SPEED 34
[656] #define DEACCELERATING_SPEED 35
[657]
[658] //Variables
[659] #define START_FLAG 255
[660] #define TX_MODE 1
[661] #define RX_MODE 0
[662] #define TIME_OUT 10
[663] #define BYTE_READ 1
[664] #define BYTE_READ_POS 2
[665] #define BROADCAST_ID 254
[666] #define MAX_SPEED 1000
[667] #define COXA_MIN_POS 100
[668] #define COXA_MAX_POS 200
[669] #define FEMUR_MIN_POS 40
[670] #define FEMUR_MAX_POS 200
[671] #define TIBIA_MIN_POS 100
[672] #define TIBIA_MAX_POS 230
[673] #define POSITION_RESOLUTION_L 0.29
[674] #define POSITION_RESOLUTION_H 3.45
[675]
[676] #include "Arduino.h"
[677] #include "Actuator.h"
[678] #include "InverseKinematics.h"
[679] #include <string>
[680]
[681] class SR518
[682] {
[683] public:
[684]     // Serial
[685]     void setSerial(HardwareSerial *sPort);
[686]     void begin(long baud, unsigned char directionPin);
[687]
[688]     // Movement
[689]     int move(byte id, double position);
[690]     int moveLeg(vector<Actuator> leg_actuators, double coxa_pos_angle, double femur_pos_angle, double tibia_pos_angle);
[691]     int moveAll(vector<Actuator> actuators, double coxa_1_angle, double femur_1_angle, double tibia_1_angle, double coxa_2_angle, double femur_2_angle,
double tibia_2_angle, double coxa_3_angle, double femur_3_angle, double tibia_3_angle, double coxa_4_angle, double femur_4_angle, double
tibia_4_angle);
[692]
[693]     // Config
[694]     int setPosLimitMin(byte id, double pos);
[695]     int setPosLimitMax(byte id, double pos);
[696]
[697]     // Readings
[698]     int getTemperature(byte id);
[699]     double getVoltage(byte id);
[700]     double getPosition(byte id);
[701]
[702] private:
[703]     HardwareSerial *serialPort;
[704]     unsigned char Checksum;

```

```

[705]     unsigned char Direction_Pin;
[706]     unsigned char Time_Counter;
[707]     unsigned char Incoming_Byte;
[708]     unsigned char Position_High_Byte;
[709]     unsigned char Position_Low_Byte;
[710]
[711]     int Temperature_Byte;
[712]     int Voltage_Byte;
[713]     int Position_Long_Byte;
[714]     int Error_Byte;
[715]
[716]     int read_error(void);
[717] };
[718]
[719] extern SR518 Hub;
[720]
[721] #endif

```

Appendix 11: Actuator.cpp / .h & Step.cpp / .h

```

[1] #pragma once
[2] #include <string>
[3] using namespace std;
[4] class Actuator
[5] {
[6] public:
[7]     Actuator();
[8]     Actuator(int, int, int, int, string, double, double, double, double, double);
[9]
[10]     void setId(int);
[11]     int getId();
[12]
[13]     void setLeg(int);
[14]     int getLeg();
[15]
[16]     void setJoint(int);
[17]     int getJoint();
[18]
[19]     void setJointName(string);
[20]     string getJointName();
[21]
[22]     void setInverted(int);
[23]     int getInverted();
[24]
[25]     void setTemperature(double);
[26]     double getTemperature();
[27]
[28]     void setVoltage(double);
[29]     double getVoltage();
[30]
[31]     void setPosition(double);
[32]     double getPosition();
[33]
[34]     void setMaxPosition(double);
[35]     double getMaxPosition();
[36]
[37]     void setMinPosition(double);
[38]     double getMinPosition();
[39]
[40] private:
[41]     int id;
[42]     int leg;
[43]     int joint;
[44]     string jointName;
[45]     int inverted;
[46]     double temperature;
[47]     double voltage;
[48]     double position;
[49]     double max_position;
[50]     double min_position;
[51] };
[52]
[53] #include "Actuator.h"
[54]
[55] Actuator::Actuator()
[56] {
[57]
[58]
[59] Actuator::Actuator(int id, int leg, int inverted, int joint, string jointName, double temperature, double voltage, double position, double
max_position, double min_position)
[60] {
[61]     setId(id);
[62]     setLeg(leg);
[63]     setInverted(inverted);
[64]     setJoint(joint);
[65]     setJointName(jointName);
[66]     setTemperature(temperature);
[67]     setVoltage(voltage);
[68]     setPosition(position);
[69]     setMaxPosition(max_position);
[70]     setMinPosition(min_position);
[71] }
[72]
[73] // Getters and Setters
[74] void Actuator::setId(int _id)
[75] {
[76]     id = _id;
[77] }
[78] int Actuator::getId()
[79] {
[80]     return this->id;
[81] }
[82]
[83] void Actuator::setLeg(int _leg)
[84] {
[85]     leg = _leg;
[86] }

```



```

[87]
[88] int Actuator::getLeg()
[89] {
[90]     return this->leg;
[91] }
[92]
[93] void Actuator::setInverted(int _inverted)
[94] {
[95]     inverted = _inverted;
[96] }
[97]
[98] int Actuator::getInverted()
[99] {
[100]     return this->inverted;
[101] }
[102]
[103] void Actuator::setJoint(int _joint)
[104] {
[105]     joint = _joint;
[106] }
[107]
[108] int Actuator::getJoint()
[109] {
[110]     return this->joint;
[111] }
[112]
[113] void Actuator::setJointName(string _joint_name)
[114] {
[115]     jointName = _joint_name;
[116] }
[117]
[118] string Actuator::getJointName()
[119] {
[120]     return this->jointName;
[121] }
[122]
[123] void Actuator::setTemperature(double _temperature)
[124] {
[125]     temperature = _temperature;
[126] }
[127]
[128] double Actuator::getTemperature()
[129] {
[130]     return this->temperature;
[131] }
[132]
[133] void Actuator::setVoltage(double _voltage)
[134] {
[135]     voltage = _voltage;
[136] }
[137]
[138] double Actuator::getVoltage()
[139] {
[140]     return this->voltage;
[141] }
[142]
[143] void Actuator::setPosition(double _position)
[144] {
[145]     position = _position;
[146] }
[147]
[148] double Actuator::getPosition()
[149] {
[150]     return this->position;
[151] }
[152]
[153] void Actuator::setMaxPosition(double _max_position)
[154] {
[155]     max_position = _max_position;
[156] }
[157]
[158] double Actuator::getMaxPosition()
[159] {
[160]     return this->max_position;
[161] }
[162]
[163] void Actuator::setMinPosition(double _min_position)
[164] {
[165]     min_position = _min_position;
[166] }
[167]
[168] double Actuator::getMinPosition()
[169] {
[170]     return this->min_position;
[171] }
[172]
[173] #include "Step.h"
[174]
[175] Step::Step(){
[176]
[177] }
[178]
[179] Step::Step(int _leg_id, int _type, int _x, int _y, int _z){
[180]     setLegId(_leg_id);
[181]     setType(_type);
[182]     setX(_x);
[183]     setY(_y);
[184]     setZ(_z);
[185] }
[186]
[187] void Step::setLegId(int _leg_id){
[188]     this->leg_id = _leg_id;
[189] }
[190]
[191] int Step::getLegId(){
[192]     return this->leg_id;
[193] }
[194]
[195] void Step::setType(int _type){
[196]     this->type = _type;
[197] }
[198]
[199] int Step::getType(){
[200]     return this->type;
[201] }
[202]
[203] void Step::setX(int _x){
[204]     this->x = _x;
[205] }
[206]

```

```

[207] int Step::getX(){
[208]     return this->x;
[209] }
[210]
[211] void Step::setY(int _y){
[212]     this->y = _y;
[213] }
[214]
[215] int Step::getY(){
[216]     return this->y;
[217] }
[218]
[219] void Step::setZ(int _z){
[220]     this->z = _z;
[221] }
[222]
[223] int Step::getZ(){
[224]     return this->z;
[225] }
[226]
[227] #pragma once
[228]
[229] class Step
[230] {
[231] public:
[232]     Step();
[233]     Step(int leg_id, int type, int x, int y, int z);
[234]     void setLegId(int _sequence);
[235]     int getLegId();
[236]     void setType(int _type);
[237]     int getType();
[238]     void setX(int _x);
[239]     int getX();
[240]     void setY(int _y);
[241]     int getY();
[242]     void setZ(int _z);
[243]     int getZ();
[244]
[245] private:
[246]     int leg_id;
[247]     int type;
[248]     int x;
[249]     int y;
[250]     int z;
[251] };

```

Appendix 12: JsonHandler.cpp / .h

```

[1] #include "JsonHandler.h"
[2]
[3] String JsonHandler::getJSONElement(String element, String file_name)
[4] {
[5]     File jsonFile = SpiffHandler::readFile(file_name);
[6]     std::unique_ptr<char[]> buf(new char[SIZE]);
[7]     jsonFile.readBytes(buf.get(), SIZE);
[8]     DynamicJsonDocument doc(SIZE);
[9]     deserializeJson(doc, buf.get());
[10]
[11]     String value = doc[element];
[12]     return value;
[13] }
[14]
[15] vector<Actuator> JsonHandler::getJsonActuators()
[16] {
[17]     File jsonFile = SpiffHandler::readFile(ACTUATOR_FILE_NAME);
[18]     std::unique_ptr<char[]> buf(new char[SIZE]);
[19]     jsonFile.readBytes(buf.get(), SIZE);
[20]     DynamicJsonDocument doc(SIZE);
[21]     deserializeJson(doc, buf.get());
[22]     JsonArray root = doc.as<JsonArray>();
[23]
[24]     vector<Actuator> actuators;
[25]
[26]     for (JsonObject item : root){
[27]         Actuator actuator = Actuator(item["id"], item["leg"], item["inverted"], item["joint"], item["joint_name"], item["temperature"], item["voltage"],
item["position"], item["max_position"], item["min_position"]);
[28]         actuators.push_back(actuator);
[29]     }
[30]     return actuators;
[31] }
[32]
[33] String JsonHandler::updateActuators(vector<Actuator> actuators)
[34] {
[35]     DynamicJsonDocument outputDoc(SIZE);
[36]     for (Actuator actuator : actuators)
[37]     {
[38]         int id = actuator.getId();
[39]         JsonObject obj = outputDoc.createNestedObject();
[40]         obj["id"] = String(id);
[41]         obj["leg"] = String(actuator.getLeg());
[42]         obj["inverted"] = String(actuator.getInverted());
[43]         obj["joint"] = String(actuator.getJoint());
[44]         obj["joint_name"] = String(actuator.getJointName().c_str());
[45]         obj["max_position"] = String(actuator.getMaxPosition());
[46]         obj["min_position"] = String(actuator.getMinPosition());
[47]         obj["temperature"] = String(Hub.getTemperature(id));
[48]         obj["voltage"] = String(Hub.getVoltage(id));
[49]         obj["position"] = String(Hub.getPosition(id));
[50]     }
[51]     String outputDocString = "";
[52]     serializeJson(outputDoc, outputDocString);
[53]     return outputDocString;
[54] }
[55]
[56] void JsonHandler::saveActuators(vector<Actuator> actuators)
[57] {
[58]     String outputDocString = updateActuators(actuators);
[59]     SpiffHandler::saveFile(outputDocString, ACTUATOR_FILE_NAME);
[60] }
[61]
[62]
[63] vector<vector<Step>> JsonHandler::readGait(String gait_file)
[64] {
[65]     File jsonFile = SpiffHandler::readFile(gait_file+"_gait.json");
[66]     std::unique_ptr<char[]> buf(new char[SIZE]);

```

```

[67]   jsonFile.readBytes(buf.get(), SIZE);
[68]   DynamicJsonDocument doc(SIZE);
[69]   deserializeJson(doc, buf.get());
[70]   JsonArray root = doc.as<JsonArray>();
[71]
[72]   vector<vector<Step>> gait_vector;
[73]
[74]   for (JsonObject sequence : root)
[75]   {
[76]     vector<Step> sequence_vector;
[77]     int sequence_name = sequence["sequence"];
[78]     JsonArray steps = sequence["steps"];
[79]
[80]     for (JsonObject step : steps)
[81]     {
[82]       Step step_object = Step(step["id"], step["type"], step["x"], step["y"], step["z"]);
[83]       sequence_vector.push_back(step_object);
[84]
[85]       Serial.print("ID: ");
[86]       const char *id = step["id"];
[87]       Serial.print(id);
[88]       Serial.print(" - X: ");
[89]       const char *x = step["x"];
[90]       Serial.print(x);
[91]       Serial.print(" - Y: ");
[92]       const char *y = step["y"];
[93]       Serial.print(y);
[94]       const char *z = step["z"];
[95]       Serial.print(" - Z: ");
[96]       Serial.println(z);
[97]     }
[98]     gait_vector.push_back(sequence_vector);
[99]   }
[100]   return gait_vector;
[101] }
[102]
[103]
[104]
[105]
[106] #pragma once
[107] #include <ArduinoJson.h>
[108] #include "SR518.h"
[109] #include "Actuator.h"
[110] #include "SpiffHandler.h"
[111] #include "Step.h"
[112]
[113] #define SIZE 32768
[114] #define ARRAY_SIZE 12
[115] #define ACTUATOR_FILE_NAME "actuator.json"
[116]
[117] class JsonHandler
[118] {
[119] public:
[120]   static String getJsonElement(String element, String file_name);
[121]   static vector<Actuator> getJsonActuators();
[122]   static String updateActuators(vector<Actuator> actuators);
[123]   static void saveActuators(vector<Actuator> actuators);
[124]   static vector<vector<Step>> readGait(String gait_name);
[125] private:
[126] };

```

Appendix 13: SpiffHandler.cpp / .h

```

[1]   #include "SpiffHandler.h"
[2]
[3]
[4]   void SpiffHandler::saveFile(String output_string, String file_name){
[5]     File file = SPIFFS.open(SPIFF_PATH + file_name, FILE_WRITE);
[6]     file.print(output_string);
[7]     file.close();
[8]   }
[9]
[10]  void SpiffHandler::saveFile(File input_file, String file_name){
[11]    File file_opened = SPIFFS.open(SPIFF_PATH + file_name, FILE_WRITE);
[12]    file_opened.print(input_file);
[13]    file_opened.close();
[14]  }
[15]
[16]  File SpiffHandler::readFile(String file_name){
[17]    File file = SPIFFS.open(SPIFF_PATH + file_name, FILE_READ);
[18]    return file;
[19]    file.close();
[20]  }
[21]
[22]
[23]
[24] #pragma once
[25] #include <SPIFFS.h>
[26]
[27] #define SPIFF_PATH "/"data/"
[28]
[29] class SpiffHandler
[30] {
[31] public:
[32]   static File readFile(String file_name);
[33]   static void saveFile(File file, String file_name);
[34]   static void saveFile(String output_string, String file_name);
[35] private:
[36] };
[37]

```