OPEN-SOURCE LEG 2.0

Development of a control strategy

ProtheseAcademie

Stijn Brouwer | 3641171

Glossary

Word	Definition
OSL	Open-Source Leg 2.0
Active Joint	A prosthetic joint that provides external power through motors.
Passive Joint	A prosthetic joint that allows for no movement.
Gait Cycle	Repetitive human walking pattern (cyclic).
BMS	Battery Measurement System
BLDC	Brushless Direct Current
FSM	Finite State Machine

Abstract

In front of you lies a master project report, on the development of the best control strategy for the Open-Source Leg 2.0 at the ProtheseAcademie. The report will take you through an introduction and methods section which will highlight how optimal and most feasible use cases were selected. The different control strategies will then be evaluated to see which is most suitable for the chosen applications. The report was written as an intern at the ProtheseAcademie department of UMCG in Groningen, where the OSL has been assembled and will continue to be worked with when this project is concluded. The report has been written to the standards of the engineering department of Rijksuniversiteit Groningen, and has been worked on since mid-April for 10.5 weeks until the beginning period of July. I would like to thank both my supervisors at the ProtheseAcademie, Han Houdijk and Verena Schuurmans, for their guidance and assistance throughout the project, pointing me in the right direction. Alongside this project I have had the pleasure of being able to help two PhD students with their projects, who I would like to thank for the experience. Chengxiang (Oran) Liu, who is working on the 'MyLeg' prosthesis, and Thijs Tankink who is working on optimising an ankle prosthesis.

Table of Contents

1. Introduction

The development of lower extremity prosthetics has seen significant advancements over the past decade, driven by enhancing user mobility and improving quality of life for individuals with limb impairments [1]. Despite these advancements, the integration and collaboration among various research groups working on prosthetics have often been fragmented. This is due to the fact that all researchers must build their own test platform, such as their own prosthetic, limiting the full potential of collective work. Leg prosthetics come in different forms, the two main ones being regular mechanical prosthetics with passive joints, and microprocessor-controlled prosthesis with active joints. A combination of passive and active joints is also a possibility and regular occurrence, as most active knee prosthetics have a passive ankle. In response to the challenge of everyone conducting individual research, an open-source robotic leg, the Open-Source Leg (OSL) 2.0, a microprocessor-controlled leg has been developed, serving as a platform to push the boundaries of prosthetic research and encourage collaboration across multiple research domains [2]. It bridges the gap between isolated research projects by providing a platform that all backgrounds can access.

One of the most significant challenges in the development of lower extremity prosthetics has been the control of the leg prosthesis. Effective control strategies are crucial for ensuring that the joints in the prosthetic leg respond accurately and intuitively to the user's movements. These joints should mimic the biological neuro-muscular structures that are missing when a leg is amputated. However, achieving this level of control has been a major drawback, hindering the full potential and usability of these devices [3]. Addressing this challenge is critical to optimising the functionality and user experience of prosthetic legs. The Open-Source Leg allows us to tackle this challenge by incorporating advanced control systems that allow real-time adjustments and responses to movements through feedback. This enhances comfort, usability, and confidence of the user when walking with a robotic leg.

The goal for the Open-Source Leg is to allow many researchers around the world to be involved and contribute to the control of robotic leg prosthesis and any problems surrounding this. All hardware and software can be obtained from the Open-Source website, improving the chances of these control methods being shared and implemented in the real world after developments are completed. This means that even those without specialised facilities, such as movement scientists, can develop their own mechatronics system.

This report outlines the approach taken to determine the optimal use cases and corresponding control strategy for the robotic leg. Through a series of stakeholder interviews involving potential users, engineers, researchers, technicians and insurance representatives, insight was gathered into the needs and preferences of the prosthetic user community. It will delve into the methodology for stakeholder discussions, present the key findings from the interviews, and discuss how it influences the direction of the OSL project at the ProtheseAcademie. This includes the development of a control strategy to optimise the robotic leg's performance for the identified use cases. Through this, the aim is to find a clear path forward for the development and application of the open-source robotic leg.

1.1 Gait Cycle

Before delving into what control strategy is best suited for various use cases, it is crucial to first understand the intricacies of human gait, as it will need to be replicated with the OSL. The natural walking pattern, is a highly coordinated and complex process involving the integration of multiple physiological systems. It involves the rhythmic and cyclical motion of the limbs, the balance and posture adjustments by the core, and the dynamic interaction with the ground. By understanding the fundamental mechanics and phases of human gait, the requirements and challenges faced when designing effective control strategies for applications such as prosthetics can be tackled [4]. ***Figure 1*** shows the different phases of human gait. ***Figure 2*** shows the leg angles.

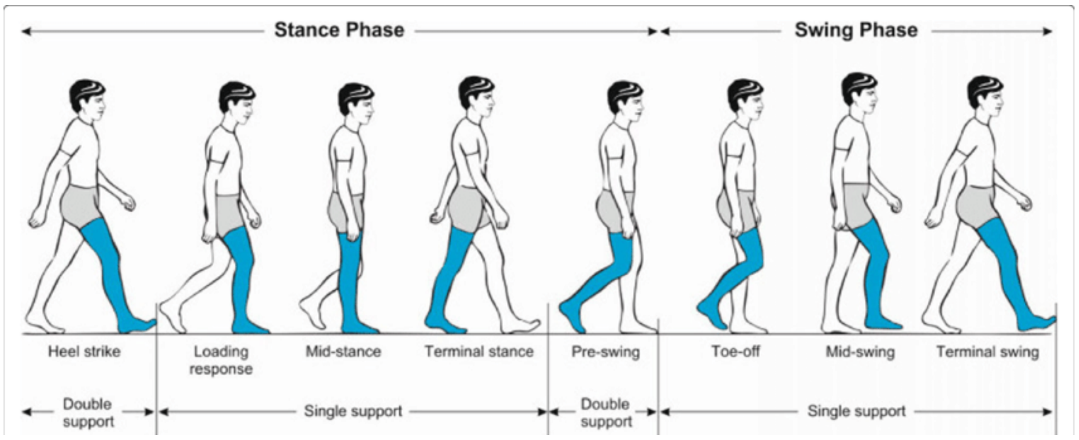

Figure 1 - Gait Cycle [5]

Stance Phase [6]:

1) Heel Strike – **Knee** = 0° (full extension), **Ankle** = 0° (neutral)
2) Loading Response – **Knee** = 15° (flexion), **Ankle** = 0 - 5° (plantarflexion)
3) Mid-Stance – **Knee** = 5° (flexion), **Ankle** = 5° (dorsiflexion)
4) Terminal Stance – **Knee** = 0° (full extension), **Ankle** = 0° (neutral)
5) Pre-Swing – **Knee** = 30° (flexion), **Ankle** = 20° (plantarflexion)

Swing Phase [6]:

1) Toe-Off – **Knee** = 60° (flexion), **Ankle** = 10° (plantarflexion)
2) Mid-Swing – **Knee** = moves to 30° (flexion), **Ankle** = 0° (neutral)
3) Terminal Swing – **Knee** = 0° (full extension), **Ankle** = 0° (neutral)

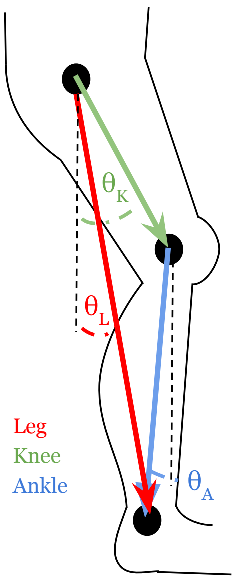

Figure 2 - Gait Leg Angles

The list below shows gait requirements needed for the Open-Source Leg 2.0:

- The prosthetic must be able to support the load from the user throughout all phases of gait.
- The knee joint of the prosthetic must allow at least 60° of flexion.
- The ankle joint of the prosthetic must allow at least 5° of dorsiflexion.
- The ankle joint of the prosthetic must allow at least 20° of plantarflexion.

2. Problem Analysis

2.1 Problem Definition

The primary objective of this project is to identify the most relevant use cases through discussion with a range of stakeholders, and to then investigate to which extent these use cases are feasible for the open-source robotic leg. This use case will guide the development and application of the prosthetic, ensuring it tackles the most pressing needs within the prosthetic user community while also serving as a versatile tool for researchers. In addition to identifying these use cases this project will also focus on exploring which control strategy is best suited and available in the OSL for the given applications. The control strategy is a critical component, as it dictates how the robotic leg responds to user inputs, ultimately influencing the effectiveness. The parameters should be easily adjustable, enabling the OSL to be used across a variety of different applications.

2.2 Overview Open-Source Leg 2.0

The Open-Source Leg 2.0 can be seen in ***Figure 3***. It is primarily constructed from aluminium and stainless steel. The mechanical system is composed of 40 machined parts, 18 screws, 2 actuators to aid the movement of both joints, and 2 batteries to supply each of them with power. It is designed to be easy to manufacture, assemble, and repair. The entire system weighs less than 5.4 kg, including both knee and ankle joints fitted with a foot attachment. The cost of the Open-Source Leg ranges from $9,000 to $19,000, depending on whether both the knee and ankle are being used as active joints or passive (degrees of freedom), and the sensing options chosen when building it, which is dependent on the control strategy selected. The knee and ankle joints are mechanically identical, which simplifies assembly, repair, and allows them to be controlled in the same way.

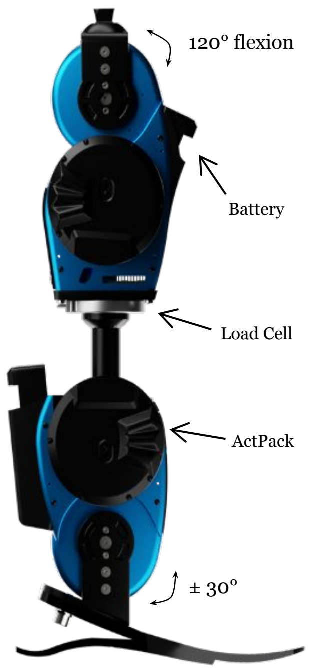

Figure 3 - Open-Source Leg 2.0 [2]

The minimum build height of the system is 451 mm, measured from the ground to the top of the knee pyramid adapter with a flat foot attachment. The height can be adjusted by changing the shaft length between the two joints. Unfortunately, the shaft is not adjustable, so it must be entirely replaced. The knee joint has a range of motion of 120° for flexion in the sagittal plane, while the ankle joint has a range of ± 30° for plantar / dorsiflexion also in the sagittal plane. Both joints support series elasticity, which can be achieved by fitting custom-designed radial springs without increasing the system's volume. Series elastic actuators, unlike rigid actuators, contain an elastic element in series with the mechanical energy source output. This gives the actuator passive mechanical energy storage, tolerance to impact loads, and low mechanical output impedance. However, impedance is only low if the springs are able to support the loads generated [7].

The control system of the Open-Source Leg includes 4 encoders to convert information from one format to another, 2 current sensors (1 at each joint), 2 inertial measurement units (1 at each joint) to measure orientation, acceleration, angular rates and gravitational forces. A load cell is used below the knee joint to measure load applied during walking, and a Raspberry Pi, which is a single-board computer, is used to control the electronic components. It is designed to be easy to sense, program, and control. The system can deliver up to 145 Nm of peak instantaneous torque at both the knee and ankle. The highest torque during human gait occurs when descending stairs, with a torque value of approximately 1.3 Nm/kg for the knee, and 1.55 Nm/kg for the ankle [8]. This means the OSL can support a user of up to 93 kg. It supports a variety of additional sensors and actuators through SPI, UART, I2C, and USB communication protocols. Communication protocols are a set of rules on how to transfer or exchange data. Each protocol serves a different purpose, and is chosen based on specific requirements such as power consumption, complexity, speed, etc. Safety is a priority, with built-in protections such as Battery I2t, to ensure the batteries can handle thermal load without overheating. Motor I2t is implemented to similarly stops the motor from overheating, and Voltage Limits to set maximum and minimum voltage for these components to safeguard both the user and the system.

The Open-Source Leg is a self-contained system with all necessary electronics embedded. It exhibits less than 0.045° of backlash at both the knee and ankle joints. The system supports various control strategies, including voltage, current, position, and impedance, for both joints. Developers can program the system using Python, C/C++, and MATLAB [2].

2.2.1 Adjustable Parameters

In this section, the adjustable parameters of the OSL will be discussed. These parameters can be fine-tuned to enhance the prosthetic's performance, ensuring it meets the specific needs and preferences of the user. ***Table 1*** shows an overview of the adjustable parameters along with a description for each. Other parameters that are not controlled specifically by software, but can still have influence on the OSL can be seen below the table.

Table 1 - Adjustable Parameters OSL

Parameters	**Description**
Joint Target Angles	Adjusting the desired flexion and extension angle of the knee joint, and dorsiflexion and plantarflexion angle of the ankle joint.
Threshold Angles	Angle where transition occurs from one phase to another (mid / late stance and swing).
Joint Stiffness	Modifying the resistance to movement in the knee and ankle joints.
Joint Damping	Adjusting the energy dissipation rate in the knee and ankle joints.
Torque Limits	Setting the maximum and minimum torque that can be applied by the knee and ankle actuator.
Trajectory	Planned movement path for the knee and ankle joint during different activities.
Gain Settings [9]	Gains for position, velocity, and force feedback control for the knee and ankle joints. Proportional Gain – Drives position error to zero. Derivative Gain – Drives derivative of error to zero. Integral Gain – Drives total error accumulated to zero.

Equilibrium – Alignment of Components

Usually this would include the alignment of the socket, knee, ankle, shaft, and the foot attachment. However, the OSL does not allow alignment of the ankle, knee, and shaft, as these are fixed. Therefore, the only components that can be adjusted are the socket (geometry, but also attachment site and orientation), foot attachment, and the length of the shaft between joints. Joint angles may also be considered a part of the leg's equilibrium.

Gait

This includes the timing parameters for different phases of the gait cycle (stance and swing phase), and adjusting the length and frequency of steps (speed) taken by the prosthetic leg.

2.2.2 Current Control Options

Before being able to decide the best control strategy for the OSL, research was conducted to gain a better understanding of all the possible options. As outlined in ***Section 2.2.0***, the system supports multiple control modes for both the knee and ankle joints, including voltage, current, position, and impedance, to generate the gait pattern desired by a specific use case. An overview of these control strategies is provided below, ***Table 2*** highlights their respective advantages and disadvantages. [10] [11]

Voltage Control - Voltage control involves regulating the voltage supplied to the motors or actuators by the two batteries specifically in the Open-Source Leg. By changing the voltage, you can control the speed and torque of the motor. Higher voltage typically means higher speed or more force. An example of this is that when a user is walking up a set of stairs, more force is required, so more voltage must be generated. [12] [13]

Current Control - Current control involves precisely regulating the current flowing through the motors or actuators. In the OSL current is detected with two current sensors. Current is directly related to the torque or force produced by a motor, so this control strategy is crucial for applications that require torque control. By controlling the current, you can control the force the motor applies. [12]

Position Control - Position control involves regulating the exact position of the robotic leg. It uses sensors (like encoders) that measure and provide feedback about the leg's position and adjusts the motor's actions to achieve the desired position. The OSL has 2 inertial measurement units (IMUs) to detect orientation and other variables [14]. In general position control is accurate, however response time, accuracy, and error are heavily dependent on how advanced the sensors are and the control system chosen. Calibration is also important.

Impedance Control - Impedance control manages the dynamic interaction between the robotic leg and its environment. It combines elements of position and force control to adjust the leg's behaviour based on external forces. The leg can be made to act "softer" or "stiffer" depending on the situation, by changing specific parameters. These specific parameters include stiffness (K), damping (D), and inertia (M). [14]

Table 2 - Advantages & disadvantages of control strategies

Control Strategy	**Advantages**	**Disadvantages**
Voltage	Cost-effective, and simple to implement. Makes it a valuable option for many practical applications.	Can be difficult to control the exact position or speed of the leg by adjusting only the voltage. Not very precise as the relationship between voltage and output is often not linear, but influenced by various other factors. Has no feedback mechanism, and is less energy efficient.
Current	Better for controlling force and torque when compared to voltage control. This makes the system more stable.	Needs more advanced sensors and feedback systems, so not perfect for precise position control. May lack sensor resolution and sensitivity needed. Inadequate feedback can result in errors or delays. Expensive hardware.
Position	Can make the robotic leg move to an exact location, so very precise and repeatable. The fact it is precise also increases safety.	Can be complex to implement as it requires precise sensors and feedback systems similar to current control. If sensors and feedback are not advanced enough it can lead to reduced performance. Expensive hardware.
Impedance	Great for tasks that require interaction with unpredictable environments as has good adaptability. This also means it is safe, versatile, and robust.	Requires good models of both the robot and its environment, as it must capture the interaction between the robot and surroundings. Complex and computationally intensive as must continuously compute desired impedance value (inertia, stiffness, damping), and optimise tuning of these parameters, so the actuator outputs the required force / torque. This requires accurate and sensitive sensors.

2.3 Market Research

2.3.1 Optimisation Options

Before selecting the best use case, it is essential to explore optimisation possibilities and applicable parameters for the Open-Source Leg. This was done by conducting market research. Currently there is not general consensus on what should be optimised in a prosthetic leg. Many prosthetic optimisations focus on metabolic or energy consumption, which can be measured with spirometry. Additionally, user feedback based on physical fatigue and comfort can provide valuable insights. Comfort is an optimisation in itself, which is influenced by load and stress on the user. Another factor to optimise is walking symmetry and step length, ensuring they are as close to the natural walking pattern as possible. By thoroughly investigating these aspects, the most effective optimisations for each use case can be determined.

2.4 Requirements

Table 3 shows an overview of the requirements for the OSL before interviews.

Table 3 - Requirements

	Description
User	- The product should be controlled to be capable of achieving K-Level 3 use cases when optimised (unrestricted outdoor walking) [15]. - The selected control system should enable any user to achieve their desired position or motion.
Function	- The product must fit all patients. The control system must allow adjustments to be made without influencing the accuracy and responsiveness of the prosthetic leg (changing socket or shaft). - The control of the product must be adaptable to cater to all patients. This can be done by altering parameters and noting influence.
Safety	- The product, including all electrical components, must conform with medical electrical ISO standards (IEC 80601-2-78:019) [16] - The OSL should be able to be optimised for all applications whilst maintaining safety standards.
Ergonomics	- The control of the product must assist patients with an above-knee, transfemoral amputation. - The control method of the product should be able to control the prosthetic on both the right and left side of the body.

2.5 Stakeholder Analysis

To be better prepared for the stakeholder interviews, such as knowing what questions to ask, an overview was made for each stakeholder with their expected thoughts and demands for the OSL. This can be seen in ***Table 4***. Stakeholders are essential to gain multiple opinions from a wide range of experts from different disciplines.

Table 4 - Stakeholder Analysis

Stakeholder	Focus	Expectations	Conclusions
Insurance Representative	Cost-effectiveness, reimbursement processes, and coverage policies	Information on cost, long-term benefits, and insurance coverage for the prosthetic	Whether the tool is financially viable and fits within coverage policies
Researcher	Developing new technologies and methodologies	Technical specifications, research data, and potential for future innovations	Interested in the scientific and technological advancement the tool represents
Orthopaedic Technician	Fitting and adjusting prosthetic devices	Usability, adjustability, and maintenance of the prosthetic tool	Ensures tool is effectively integrated and customised for patients
Engineer	Design, materials, and functionality	Engineering data, material properties, and performance metrics	Evaluate technical integrity and innovative aspects of the design
Rehabilitation Doctor	Patient recovery and therapy	Tools impact on patient rehabilitation, ease of use, and patient outcomes	Ensures tool aids in effective patient recovery and improves quality of life
Prosthetic Users	Actual users of the prosthetic leg tool	Comfort, ease of use, durability, and improved mobility	Feedback for understanding real-world efficiency and user satisfaction

3. Methods

To discover what the needs of the Open-Source robotic leg are for its potential users, a series of interviews were conducted with a wide range of stakeholders, as prosthetics is a multidisciplinary field. These interviews aimed to gather insights into how different stakeholders would utilise the robotic leg and to collaboratively brainstorm ideas for its application. Discussing with a broad spectrum of stakeholders ensured that all possible uses were covered, capturing an understanding of specific requirements, and limitations. This collaborative approach enabled the identification of practical use cases that will inform the development and optimisation of the control strategy and software integration, which was the outcome of this part of the project. Before the interviews, each participant received a presentation on the project and the Open-Source Leg (OSL) to ensure a common understanding and facilitate more insightful responses. The questions asked from this presentation can be seen in ***Appendix I***. Stakeholders were strategically chosen from a diverse array of relevant disciplines, including orthopaedic technicians, insurance representatives, engineers, rehabilitation doctors, prosthetic users, and researchers with expertise in the field. A summary of what was discussed with each stakeholder can be seen below.

3.1 Stakeholder Interviews

3.1.1 Insurance Representative (Tanja Bastiaansen – CZ)

During the meeting with Tanja, an insurance representative, Tanja noted that this prosthetic is currently too heavy, bulky, and expensive for regular use by amputees. Although it is primarily used as a research instrument, Tanja suggested it could be more effectively utilised during the provision process in a trial phase.

Specifically, Tanja recommended using the Open-Source Leg 2.0 to compare knee and ankle joints with different components and software. This approach is feasible because the leg's components are quickly and easily interchangeable, but more importantly that the OSL actuators can simulate other commercially available components. However, since the leg is not intended to be a definitive long-term solution, it will not be covered by insurance.

To implement this use-case, a protocol must be developed to ensure the smooth changing of electrical components such as sensors, and mechanical ones like the shaft/pylon and the foot. Additionally, there should be a mechanism for switching between different control scripts or software versions.

3.1.2 Researcher (Bob van der Windt– TU Delft)

During the meeting with Bob, a researcher, potential applications for the Open-Source Leg 2.0 (OSL) were discussed. Bob outlined a few key areas where the OSL could be beneficial:

The Open-Source Leg 2.0 could be used for a number of different applications. One use is in developing different walking algorithms for the knee and ankle joint during walking, enabling analysis of their impact on user gait patterns. By analysing how this relationship between joints affects the user's gait, researchers can better understand and improve walking patterns for prosthetic users.

Another potential application could focus on the dynamics of standing up from a sitting position and vice versa, rather than only looking on gait. By altering the torque in the knee joint, researchers can study its impact on these movements, and find the torque required. This is especially relevant to Bob's current research, which is aimed at developing a prosthetic specifically designed to assist with these motions.

A different suggestion was to use the leg as a decision-making tool for fitting prosthetics in hospitals. By having patients wear the OSL and adjusting its parameters, clinicians can determine the optimal conditions for each patient. However, it is important to consider that the OSL is heavier than most commercial prosthetics, which must be considered in this application.

The final use case mentioned would be to test software. Bob mentioned his work on the ERiK leg, which has an unwanted button on the front of the thigh part. If the OSL could be used to develop a controller that replicates the function of this button, it could then be implemented into the ERiK leg, enhancing its usability.

3.1.3 Orthopaedic Technician (Jeroen Olsman – OIM)

In the meeting with Jeroen, an Orthopaedic Technician, several potential uses for the Open-Source Leg 2.0 (OSL) were explored. Jeroen highlighted the following key points:

The Open-Source Leg 2.0 could be used as a tool to compare different components and hardware or as a decision-making tool for a client's prosthetic needs. Currently, opinions are subjective and rely on the experience of the technician, but with the OSL, these could be turned into objective assessments. This objectivity could be particularly beneficial in improving the confidence of new technicians, as it means there is less need for years of experience.

Using the OSL in the provision process, which currently involves a lot of trial and error, could significantly reduce the need to construct multiple prosthetics. This approach would make the process more efficient and less resource-intensive.

While most prosthetics are designed to be lightweight, the OSL is quite heavy. To address this, Jeroen suggested the possibility of implementing a handicap factor. This adjustment would compensate for the weight difference between the OSL and the final prosthetic chosen for the user.

3.1.4 Engineer (Herman van der Kooij – Universiteit Twente)

In the meeting with Herman, he suggested the potential of the Open-Source Leg 2.0 as a development tool for creating new interfaces and controllers for various different leg prosthetics. Currently, the testing process is expensive due to the need for acquiring different components and hardware. Using the OSL could significantly reduce costs by reducing testing to a single leg, thereby minimising the need to purchase numerous components. To make implementation across different prosthetics easier, Herman recommended having a universal software solution alongside the OSL. However, it's important to note that the OSL currently lacks certification, which may impact its broader use.

3.1.5 Rehabilitation Doctor (Aline Vrieling – UMCG)

In the meeting with Aline, she suggested that the Open-Source Leg 2.0 can serve as a crucial tool during the start-up phase of prosthetics, aiding in the selection of optimal options for the provision process. Currently, most rehabilitation prosthetics are passive and assessed visually, lacking precise measurements from sensors for accurate feedback to doctors. Various joint options are borrowed for testing purposes. With the OSL, parameters of these joints can be simulated to assess them and compare against alternatives, and it can even operate in passive mode for comparison with mechanical prosthetics.

Establishing clear instructions and protocols will be essential to enable easy parameter and component adjustments by rehabilitation doctors and others. However, the impact of the OSL's weight and distribution needs consideration, alongside its capability to accommodate different sockets. Additionally, achieving CE certification is crucial.

3.1.6 Prosthetic Users (Johan, Wybe – ProtheseAcademie)

Unlike most other stakeholders, Johan believes that the OSL is not too heavy to be used as a standard leg prosthetic. He states that if the OSL can reduce fatigue and function well as a prosthetic, its weight and aesthetics are of secondary importance. Johan, who uses a socket prosthetic himself, emphasises the importance of the type of socket used, such as vacuum or pin lock. He also believes the prototype is not too expensive compared to other prosthetics. Johan suggests that the OSL could be particularly beneficial for beginner prosthetic users. Currently, mechanical legs are used to introduce new users; however, Johan disagrees with this approach, opting instead for an adjustable leg tailored to the specific needs of the user.

Wybe, on the other hand, believes that the OSL should be used as a tool to compare both microprocessor-controlled and mechanical prosthetics (with a passive mode). Unlike Johan, Wybe uses an osseointegration prosthetic, so the optimal fitting of a socket is not a concern for him. While he thinks that comparison is a valuable application for the OSL, he thinks that the initial highest priority is ensuring the leg functions properly. Once that is achieved, different use cases can be tested to determine the best application. Wybe suggests that the OSL should be tested by experienced users who are comfortable and not afraid of trying new technologies.

When asked about current limitations in leg prosthetics and their wishes for future developments, both mentioned the difficulty of walking on terrain with varying heights. They believe this could be improved with better suspension, as the current method of compensating for height differences is to bend the prosthetic knee. I believe that having two active joints, as opposed to the current passive ankle, would make adjustments easier since both joints could change angles. Additionally, they expressed a desire for the leg to be waterproof and able to react to resistance, which they currently lack.

3.2 Stakeholder Summary

Table 5 shows a summary of the various opinions from different stakeholders on what the application of the OSL should be.

Table 5 - Summary Stakeholder Interviews

Stakeholder	Opinion OSL Application
Insurance Representative	Testing / Comparison Tool
Researcher	Gait Analysis (algorithms), Decision-Making Tool, Development Tool
Orthopaedic Technician	Testing / Comparison Tool, Decision-Making Tool
Engineer	Development Tool
Rehabilitation Doctor	Testing / Comparison Tool
Prosthetic Users	Standard Prosthesis, Testing / Comparison Tool

3.3 Use Case Selection

Following discussion with all stakeholders, each potential application mentioned in the meetings was further investigated to give a better overview and find the most relevant and feasible use cases to prioritise. The different use cases were compared in terms of viability, including technical and operational feasibility, as well as usefulness including the benefits, impact, and demand. The potential drawbacks and complications were also looked into. Finally, the parameters needed for the selected optimisation will be looked into to see which is the best suited control strategy.

3.3.1 Use Case I – Testing / Comparison Tool

The Open-Source Leg 2.0 (OSL) could serve as a tool for testing and comparing different prosthetic components and software. By allowing for quick changes, as well as simulation of components and software, existing prosthetics can be mimicked through parameter adjustments, and the OSL significantly enhances both research and clinical practices. In order for this to be successful, the components must be easily changed with a clear set of instructions for researchers. Additionally, a comparing protocol should be developed to ensure everything is covered and nothing is over looked. For this specific use case, the optimisation goals are not limited, but can be selected based on what is wanting to be compared. Therefore, all parameters should be able to be adjusted in this use case. Having a smooth switch between programming languages will make this application very beneficial. Other than mimicking, which does not require switching of physical components, there are a number of different components that can be compared. This includes electrical components like sensors, comparing their accuracy (error), or how their placement affects this. It can also include mechanical components such as a foot attachment, and seeing how this affects gait.

3.3.2 Use Case II – Control Algorithm Tool

The Open-Source Leg 2.0 (OSL) could serve as a tool for developing and testing different algorithms for the relationship between the knee and ankle joints during walking. It will give a better understanding of the combination of two active joint prosthetics, as current research is mostly working with a passive ankle. By enabling real-time adjustments and detailed gait analysis, the OSL can help optimise walking patterns, and enhance functional movements. This not only improves the prosthetic's performance but also ensures a more personalised and effective process for users. Something I believe this could specifically target is improving walking on terrain with uneven heights, as the difference in height should be easier accounted for with two joints rather than one.

3.3.3 Use Case III – Decision-Making Tool

The Open-Source Leg 2.0 (OSL) could serve as a decision-making tool for amputees and clinicians to determine the best prosthetic leg options for them specifically. By developing a detailed testing protocol and focusing on specific optimisation goals such as metabolic consumption and walking symmetry, the OSL can provide valuable insights and data-driven recommendations. This approach ensures a personalised and well-informed prosthetic selection process, leading to improved user satisfaction and functional outcomes. The testing protocol should include a set of steps that focus on specific aspects of the protocol based on the wanted optimisation. This can include damping, stiffness, equilibrium, etc. for energy consumption.

3.3.4 Use Case IV – Development Tool

The Open-Source Leg 2.0 (OSL) could serve as a development tool for advancing prosthetic technology. By enabling the testing of mechanical components and the development of control systems, the OSL can help researchers and engineers refine and improve other prosthetics in their development stage. An example of this is an engineer not being happy with a specific component of their prosthetic before it is brought to the market. They would reach out to use to OSL as a tool to come up with a solution to this problem, as is allows multiple different software, control strategies, and components to be tested. This approach leads to significant advancements in prosthetic design and functionality, ultimately benefiting users with more effective and comfortable prosthetic options. It would also reduce time and money needed for constructing multiple prosthetics. This use case is not limited to finding the best parameters or components for a prosthetic, but instead is used to tackle the task of implementing a new feature or control system to a prosthetic.

3.3.5 Use Case V – Standard Prosthesis

The Open-Source Leg 2.0 (OSL) could serve as a standard socket prosthesis for amputees. The OSL may be heavier than the average upper leg prosthesis of 8 lbs (3.6 kg) [10], but it can still be a viable option for a prosthetic leg. As long as functionality is high, for some prosthetic users, this will outweigh the aesthetics and bulkiness of the leg. If the OSL is used for this application, then the socket fitment will be of highest importance, as it distributes the user's weight and helps with smooth movements.

3.4 Use Case Overview

Table 6 shows an overview of the different use case's, along with the advantages and disadvantages of each of these applications.

Table 6 - Advantages & disadvantages of use cases

Use Case	Advantages	Disadvantages
Testing / Comparison Tool	**Flexibility:** Easy changing of prosthetic components and software, including programming language, allows testing and comparison. **Mimicking Prosthetics:** Can adjust parameters to mimic existing prosthetics, which enhances research,	**Complexity:** Requires a clear set of instructions for researchers to change components. **Protocol:** A comparison protocol must be developed, which can be time-consuming and complex.

	and also makes the product customisable. **Research and Clinical Benefits:** Improves research and clinical outcomes through detailed comparison protocols.	**Resource Intensive:** Will require many resources to develop and maintain the testing and comparison if components are being tested instead of mimicked.
Control Algorithm Tool	**Real-Time Adjustments:** Allows real-time adjustments and detailed gait analysis, leading to optimised walking patterns, and functionality. **Dual Joint Research:** Gives an insight on the combination of active knee and ankle joints, which is rarely explored in current research. This can improve things such as walking on uneven terrain by accounting for height differences with two joints.	**Complexity:** Developing and testing walking algorithms can be challenging and requires research and expertise. **Resource Intensive:** Will require computational resources and real-time adjustment capabilities (possible with OSL). **Specialised Equipment:** Will require specialised equipment for detailed gait analysis and real-time adjustments.
Decision-Making Tool	**Personalised:** Provides recommendations from data for selecting the best prosthetic options. This improves user satisfaction. **Specific Optimisation:** Can focus on specific optimisation goals such as metabolic consumption and walking symmetry. **Versatile Testing:** Includes detailed testing protocols that address various aspects like damping, stiffness, and equilibrium.	**Protocol:** Developing a detailed testing protocol can be complex and time-consuming, as translation of optimal settings from the OSL to a daily prosthesis may be difficult. **Training:** Requires training for clinicians to effectively interpret results. **Initial Investment:** Significant initial investment in developing the decision-making framework and tools.
Development Tool	**Technological Advancement:** Helps in advancing prosthetic technology by testing mechanical components and developing control systems. **Versatile Testing:** Allows for testing multiple software, control strategies, and components, leading to improved designs.	**Resource Intensive:** Requires significant resources for testing and development. **Technical Expertise:** Needs research and expertise for effective utilisation and development. **Infrastructure:** Requires a robust infrastructure to support extensive testing and development.

	Cost-Effective: Reduces the need for constructing multiple prosthetics, saving time and money. **Problem Solving:** Provides a platform for engineers to address specific issues with prosthetic components before market release.	
Standard Prosthesis	**Viable Prosthetic:** Can serve as a standard socket prosthesis despite being heavier than average, as for some users, high functionality may outweigh concerns about aesthetics and bulkiness. **Socket Fitment:** Emphasises the importance of socket fitment for weight distribution and smooth movement, which can enhance comfort and usability.	**Weight:** Heavier than the average upper leg prosthesis, which could be a drawback. **Aesthetics and Bulkiness:** Not be as aesthetically pleasing or as lightweight as other prosthetic options. This could make it less successful once placed on the market. **Certification:** Will be required to meet certain standards and certification, due to it being an everyday long-term assistive device.

3.5 Use Case Control Strategies

Selection of the best control strategy for each use case depends on the specific requirements and goals for each use case. Any of the four control strategies could be a viable option for each application. Below recommendations for each use case can be found based on typical control strategy characteristics.

Testing / Comparison Tool: Impedance Control

- Impedance control is flexible and allows for fine-tuning of the interaction between the prosthetic and the user. It can mimic different prosthetic behaviours and is ideal for testing various components, such as motors, load cells, sensors, but also non electrical things for example foot attachments under different conditions. It allows researchers to adjust stiffness and damping parameters easily, making it suitable for comparison purposes, and looks to be feasible for the OSL. Although position control is precise, it does not offer the same level of interaction properties needed for extensive testing and comparison. Current control focuses on torque of the actuator, but it lacks the ability to adjust parameters like stiffness and damping. Voltage control is more suitable for basic control of actuators and less for the testing of dynamic interactions.

Control Algorithm Tool: Position Control

- Position control is important for precise gait analysis and the development of walking algorithms. It ensures that the prosthetic moves to the desired positions accurately, which is essential for testing and optimising gait patterns. It allows for detailed study and refinement of walking dynamics, especially when dealing with complex terrains and joint coordination. Impedance control could also be a viable option as it is adaptive, however it is less precise when it comes to detailed positional adjustments that are needed in gait analysis. Current control focuses on torque control rather than positional accuracy needed for this application. Voltage control is also not effective for precise movement.

Decision-Making Tool: Impedance Control

- Impedance control offers a balance between force and position control, providing a more natural and adaptive interaction with the user. This is important for prosthetic selection based on specific user needs and optimising the OSL. It can adjust to different walking conditions, which is beneficial for decision-making processes, and can change many parameters in the joints such as stiffness, inertia, and damping. These parameters have an influence on user interaction with the OSL. Position control lacks the adaptability required; current control is too focused on actuator output as opposed to user interaction. Finally, voltage control does not have the response capabilities needed for adjustments.

Development Tool: All Control Methods

- As the development tool is so broad, all control strategies can be a viable option based on the specific application it is used for. Impedance control allows testing under various conditions and fine tuning; therefore, it would be best for adaptive component testing. Position control is ideal where positional accuracy is needed, so it is well suited for ensuring accurate movement and gait simulation. Current control would be optimal for motor and actuator testing as it focuses on torque management and component durability. Voltage control would be effective in basic component and electrical circuit testing, and is a very simple method.

Standard Prosthesis: Impedance Control

- For a standard prosthesis, impedance control offers the best compromise between adaptability and ease of use. It allows the prosthesis to respond dynamically to the user's movements and varying walking conditions, providing a more comfortable and natural experience. While it may be heavier, the focus on functionality through impedance control can improve user satisfaction. Position control could be viable, but it lacks the adaptive response needed for everyday use. Voltage and current control don't provide the user comfort and adaptability required.

4. Results

4.1 Final Use Case

Each use case of the OSL offers unique advantages and challenges, making it suitable for various applications in the field of prosthetics. Rather than developing a control strategy for the best use case, the focus should lie on the development of a control strategy that can effectively be applied across various applications. This makes the product much more versatile, increasing the chances of it aiding the advancement of these technologies.

4.2 Control Strategy Selection

For the possible applications, position and impedance control seem like the best choice. They have higher precision and accuracy than other control strategies, and give feedback. This section will give an overview of these two control strategies with a step by step on how it works and some examples.

4.2.1 Strategy I – Position Control

When using position control for an active prosthetic leg, it usually does so by controlling the angle of both joints, to achieve the desired movements to get to the needed positions. Below is an overview of how a position control system works, and how it would be implemented into the OSL.

1) Joint Angle: Both joints will have sensors that will measure the current angle of the joint and provide feedback.

2) Desired Position: A desired position or movement input will be set in the control script. This can come from various sources such as sensors, or can even pre-programmed.

3) Control Algorithm: The error (difference) between the desired (input) and current (sensors) position can be calculated with a control algorithm. This could be a PID controller, which calculates proportional, integral, and derivative responses. This could also be just a P (proportional) controller.

$$PID\ Control\ Output = K_p * Error + K_i * \int Error\ dt + K_d * \frac{d(Error)}{dt}$$

4) Actuator Torque or Force: Based on the error, the control algorithm determines how much torque or force is required for the joints to move to the desired position. The actuators then apply this torque or force.

5) Feedback Loop: The sensors continuously provide feedback to the control algorithm. This allows real-time adjustments to achieve the desired positions of the joints.

6) Safety: To ensure safety, the control script should include limits to prevent the joints from exerting too much force and moving past their desired positions.

Example Code

The example code below shows an example of position control with a PID controller. Trajectory is also implemented, and comments in green explain the code.

```
# Import necessary libraries
import time
# Provides various time-related functions to manage timing and delays in a control
loop

import math
# Provides mathematical functions defined by C standard

import prosthetic_module
# Replace with actual modules needed (units, Event, State, StateMachine,
OpenSourceLeg)

# PID constants (tune for specific hardware)
Kp = 1.0    # Proportional gain
Ki = 0.1    # Integral gain
Kd = 0.05   # Derivative gain

# Initialise variables
previous_error_knee = 0
# Previous error for derivative term calculation (Knee)

integral_knee = 0
# Integral of errors (Knee)

previous_error_ankle = 0
# Previous error for derivative term calculation (Ankle)

integral_ankle = 0
# Integral of errors (Ankle)

# Trajectory parameters
trajectory_period = 10
# Total time of trajectory in seconds

time_step = 0.01
# Control loop time in seconds (interval at which updates)

current_time = 0
# Tracks the current time

# Function to generate desired trajectory
def desired_trajectory(t):
# Function with parameter 't' to compute and return desired angle at 't'

# Example sinusoidal trajectories for knee and ankle joints
    desired_angle_knee = 30 * math.sin(2 * math.pi * t / trajectory_period)
# 30 degrees amplitude sin wave so joint will move (± 30°)
```

```
    desired_angle_ankle = 15 * math.sin(2 * math.pi * t / trajectory_period)
# 15 degrees amplitude sin wave so joint will move (± 15°)

    return desired_angle_knee, desired_angle_ankle

# math.sin generates sine wave and * by 2 to get full cycle (360°)
# Divide by duration/period (time for one complete cycle)

# Main control loop
while current_time <= trajectory_period:

    # Get desired joint angles from the trajectory function
    desired_angle_knee, desired_angle_ankle = desired_trajectory(current_time)

    # Sensors read joint angles
    current_angle_knee = prosthetic_module.read_knee_angle()
    current_angle_ankle = prosthetic_module.read_ankle_angle()

    # Calculate errors (difference)
    error_knee = desired_angle_knee - current_angle_knee
    error_ankle = desired_angle_ankle - current_angle_ankle

    # Update integral term
    integral_knee += error_knee * time_step
    integral_ankle += error_ankle * time_step

    # Calculate PID control outputs
    control_output_knee = Kp * error_knee + Ki * integral_knee + Kd * (error_knee
- previous_error_knee) / time_step
    control_output_ankle = Kp * error_ankle + Ki * integral_ankle + Kd *
(error_ankle - previous_error_ankle) / time_step

    # Update previous errors for next iteration
    previous_error_knee = error_knee
    previous_error_ankle = error_ankle

    # Apply control outputs to actuators
    prosthetic_module.control_knee_actuator(control_output_knee)
    prosthetic_module.control_ankle_actuator(control_output_ankle)

    # Increment current time
    current_time += time_step

    # Sleep to maintain the control loop timing
    time.sleep(time_step)
# Adjust depending on the control loop frequency
```

To generate the desired trajectory, a sinusoidal wave is used to provide a mathematical function. A feedback control system (PID) is then used to adjust the actuators position to the desired trajectory. The general form of a sinusoidal can be seen below:

$$x(t) = A\sin(wt + \phi) + B$$

Where:

- $x(t)$ is position at time t
- A is the amplitude of the wave (maximum displacement from equilibrium, so desired angle for this case)

- w is angular frequency where $w = \frac{2\pi}{T}$ and T is period
- ϕ is the phase shift (determines where wave starts at $t = 0$, if $\phi < 0$ the wave is shifted to right so peaks and troughs reached slower)
- B is offset (represents equilibrium position)

A basic motion script using position control, developed by Kevin Best, can be seen in ***Appendix II***. The script works with a proportional controller and trajectory, and comments have been added to explain each step.

4.2.3 Strategy II – Impedance Control

When using impedance control for an active prosthetic leg, it involves setting up parameters that determine how the joints respond to user inputs and external forces. Stiffness, damping, and inertia can be adjusted to provide the user with a more responsive and natural movement experience. Below is an overview of how impedance control system works, and how it would be implemented into the OSL.

1) Model Dynamics: This is done to get a better understanding of how forces affect each joint before defining parameters. Dynamics such as mass, inertia, and damping are modelled, and how the product interacts with the environment is considered.

2) Define Parameters: The desired stiffness (K), damping (D), and inertia (M) are defined. These parameters determine the responsiveness of the OSL during use.

3) Design Controller: The control method must be decided upon. Position control can be used as a part of the impedance control in a hierarchal manner. If this is the case, position control is used for desired trajectories, and impedance determines interaction forces with the environment, making it adaptable. An impedance controller adjusts the torque / force applied based on the current position, external forces, and velocity. An FSM controller can be implemented to manage the phases of the leg during gait, which would be needed for the OSL. Again, position control can be used alongside impedance for this.

$$Control\ Output = M(\ddot{x}_d - \ddot{x}) + D(\dot{x}_d - \dot{x}) + K(x_d - x)$$

$$(x_d - x) = Position\ Error, \qquad K = Stiffness\ Matrix$$

$$(\dot{x}_d - \dot{x}) = Velocity\ Error, \qquad D = Damping\ Matrix$$

$$(\ddot{x}_d - \ddot{x}) = Acceleration\ Error, \qquad M = Inertia\ Matrix$$

4) Feedback: The sensors give feedback to continuously adjust the impedance parameters, allowing the OSL to adapt to changes in gait or

the environment. Here the error (difference) between current and desired is calculated to help compute control torque / force.

5) Safety: To ensure safety, the control script should include limits to prevent the joints from exerting too much force or torque, that could damage to prosthetic.

Example Code

The example code below shows an example of impedance control with comments made in green explain the code.

```
# Import necessary libraries
import numpy as np
# For numerical operations, to handle matrices and arrays

import time
# Provides time-related functions to manage timing, delays in control loop

# Initialise stiffness (K), damping (D), and inertia (M) matrices
K = np.array([[500, 0], [0, 200]])  # Stiffness values for knee and ankle
D = np.array([[50, 0], [0, 20]])    # Damping values for knee and ankle
M = np.array([[1, 0], [0, 1]])      # Inertia values for knee and ankle
# Here numpy creates 2D array/matrix, example: [500, 0], [0, 200] represents a 2x2
matrix: $K = \begin{bmatrix} 500 & 0 \\ 0 & 200 \end{bmatrix}$

# Desired trajectories (sinusoidal trajectories used for this example)
def desired_trajectories(t):

    x_d = np.array([0.1 * np.sin(t), 0.05 * np.sin(t)])
# Desired positions in the two dimensions (x and y), 0.1 being the amplitude (x)

    v_d = np.array([0.1 * np.cos(t), 0.05 * np.cos(t)])
# Desired velocities, cos used here as derivative of sinusoidal position function

    a_d = np.array([-0.1 * np.sin(t), -0.05 * np.sin(t)])
# Desired accelerations, negative as second derivative of position function

    return x_d, v_d, a_d

# Placeholder functions to get sensor data
def joint_positions():
    # Replace the random number with actual sensor reading
    return np.random.randn(2) * 0.01

def joint_velocities():
    # Replace the random number with actual sensor reading
    return np.random.randn(2) * 0.01

def joint_accelerations():
    # Replace the random number with actual sensor reading
    return np.random.randn(2) * 0.01

# Placeholder function to apply control to the motors
def apply_control(control_force):
    # Replace with actual motor force driving the joints
    print(f"Applying control force: {control_force}")

# Set control loop parameters
```

```
control_loop_period = 0.01      # Control loop period in seconds
end_time = 10                   # Run the control loop for 10 seconds

# Main control loop
start_time = time.time()
# Records start time of control loop

while time.time() - start_time < end_time:
# While loop runs control loop for specific time
    t = time.time() - start_time
# 't' is time since start of loop

    # Desired trajectories
    x_d, v_d, a_d = desired_trajectories(t)

    # Sensor readings
    x = joint_positions()
    v = joint_velocities()
    a = joint_accelerations()

    # Calculate errors (difference) between desired and actual
    position_error = x_d - x
    velocity_error = v_d - v
    acceleration_error = a_d - a

    # Compute control force/torque using impedance control law
    control_force = M @ acceleration_error + D @ velocity_error + K @
position_error

    # Applies computed control forces to the joints
    apply_control(control_force)

    # Sleep for the control loop period
    time.sleep(control_loop_period)
    # Waits for next control loop iteration to maintain desired loop frequency
```

A script using impedance control with a FSM controller, developed by Raj Ayyappan, and Kevin Best, can be seen in ***Appendix II***. The script works with state transitions conditions, for transitioning between different phases (stance and swing), and event-driven transitions for events within phases such as foot-flat, heel-off, toe-off etc. The FSM uses mostly impedance control, but also with position control (joint angles) being a component of the impedance control strategy.

4.3 Electrical Components

This section will cover the current components implemented in the OSL. Any limitations will be discussed.

4.3.1 Battery

It is recommended to use a BA30 LiPo battery for the OSL, which refers to a type of Lithium Polymer (LiPo) battery, seen in ***Figure 4***. Its voltage is 33.3 V, and it has a maximum current of 10A, which should be sufficient for a prosthetic leg with two active joints. The batteries capacity is 1Ah (Amp Hours), however the duration of usage will depend on the power consumption of the OSL. This may be a limitation for the device if used intensively. It has an integrated BMS which will ensure it does not overcharge, and should have adequate longevity to for a prosthetic device. The battery is charged with a Dephy BA30 Smart Dock [17].

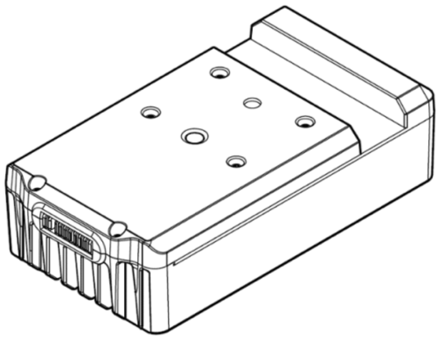
Figure 4 – BA30 LiPo Battery [17]

4.3.2 Load Cell

The M3564F load cell seen in ***Figure 5*** is from the M35XX series by Sunrise Instruments [18]. This series is a range of extra thin 6-axis stainless steel load cell designed to measure forces and torques in 3D space (Fx, Fy, Fz, Mx, My, Mz). They have a simple and small design (30mm – 70mm diameter), making them ideal for applications with limited space such as prosthetic legs. The M3564F has a diameter of 65mm and thickness of 9.2mm, can measure forces up to 2500N in the Fx, Fy direction and 5000N in the Fz direction. It can measure torques up to 200Nm in the Mx, My direction, and 100Nm in the Mz direction. This model is known for being lightweight and features a high resolution suitable for a prosthetic leg. It being a 6-axis load cell may increase complexity, but this is necessary for a high-tech prosthetic.

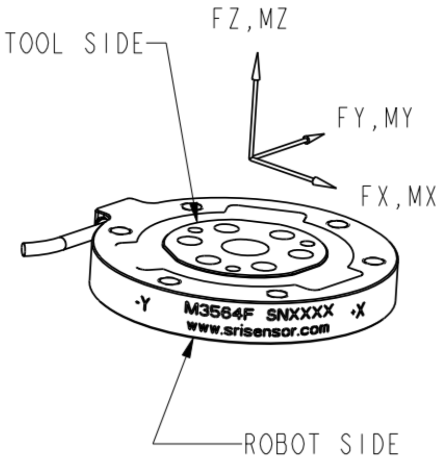

Figure 5 - M3564F Load Cell [19]

4.3.3 ActPack

An ActPack [20] is an advanced actuator system which integrates multiple components into a single, compact unit. It is designed to provide high-performance and precise actuation for robotic devices, including prosthetics. It works with a BLDC motor, which is highly efficient, durable, and provides precise control with high torque. It supports various communication protocols, programming languages, and control modes needed for a prosthetic leg. To ensure safety, it includes voltage limits, temperature safeguards, and battery

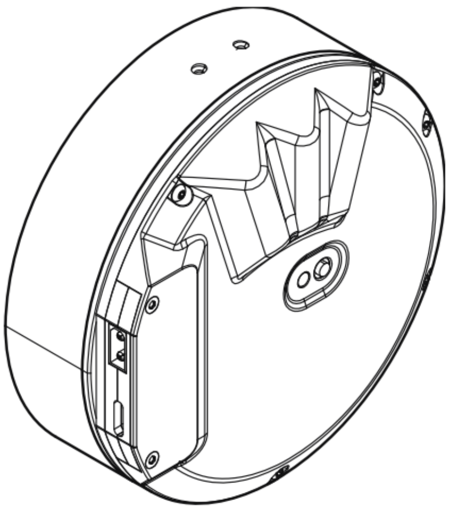
Figure 6 – Geared Actuator [20]

and motor operation. The actuator comes in three different configurations, the OSL using the 9:1 geared configuration seen in ***Figure 6***. This actuator gears down the motors output by a ratio of 9:1, meaning the output shaft turns once for every nine motor rotations. This configuration provides the highest torque and lowest speed from the three, which is ideal for prosthetic joints that require significant force, and where supporting body weight and stable movement is crucial. The voltage range required is 20 – 50 V which complies with the 33 V supplied by the batteries.

The ActPack also includes other components including a 6-axis IMU, voltage, current, temperature sensors, a rotary encoder, a strain gauge amplifier, and an expansion port [20].

The IMU is an ICM-20602 with a 3-axis gyroscope component that measures angular velocity across the x, y, z axes. The range is ±250 to ±2000 degrees per second. It also has a 3-axis accelerometer which measures linear acceleration across the same axes. The range is ±2g to ±16g (gravity). With these components it can track motion and orientation, providing data to make real-time adjustments of the actuator [20].

The encoder is a AS5048A/AS5048B magnetic rotary motor encoder with a high precision 14-bit resolution, meaning it can detect small changes in angles. The encoder measures speed, position, and direction of the motor shaft. This gives positional and rotational data as feedback. It has a broad temperature range (-40° – 150°) making it reliable in most environments [21].

The strain gauge amplifier used One Full-Bridge Channel amplifier. It has a 5V excitation voltage to generate a measurable output, and a gain of 203 to boost small signals by a factor of 203. It is used to amplify small voltage changes produced by the strain gauge, so that it can be easily read by the ActPack's control system. The Full-Bridge configuration ensures high sensitivity and accuracy by using four strain gauges in a Wheatstone arrangement [20].

5. Conclusion

From the extensive research done throughout this project, and the input given by stakeholders during their interviews, the optimal control strategy for the Open-Source Leg (OSL) should be a combination of impedance control with a Finite State Machine (FSM) controller, including aspects of position control (***Appendix II***). This hybrid approach ensures both precision and adjustability, addressing the diverse requirements of various use cases.

Impedance control, known for its ability to adapt to different walking conditions and provide a natural interaction with the user, stands out for its versatility and stability. This control strategy is especially beneficial in scenarios such as testing and comparison, where fine-tuning of interaction properties like stiffness and damping is of high importance. It also offers significant advantages for decision-making tools, where personalised adjustments to different gait conditions are necessary for optimal prosthetic selection and user satisfaction.

The FSM controller enhances impedance control by allowing smooth transitions between different control modes, thereby improving the overall responsiveness and adaptability of the prosthetic leg. This combination ensures that the prosthetic can handle the complex dynamics of human gait, providing a smooth and user friendly experience.

Incorporating aspects of position control adds an additional layer of accuracy essential for applications requiring exact positional adjustments, such as gait analysis and the development of walking algorithms. Position control ensures that the prosthetic moves accurately to the desired positions, which is critical for studying and refining walking dynamics.

In summary, a control strategy that integrates impedance control with a FSM controller and incorporates position control offers the best balance of adaptability, precision, and user interaction. This approach not only meets the diverse needs of various applications but also enhances the functionality and user experience of the Open-Source Leg.

Furthermore, all the electric components selected for the OSL, including the BA30 LiPo battery, M3564F load cell, and ActPack, appear to be well-chosen, providing the necessary performance, durability, and safety features required for an advanced prosthetic leg. This ensures that the OSL remains a versatile and effective tool for advancing prosthetic technology, optimising walking patterns, and improving the quality of life for prosthetic users.

6. References

[1] – Laferrier, J. Z., & Gailey, R. (2010). Advances in lower-limb prosthetic technology. *Physical medicine and rehabilitation clinics of North America*, *21*(1), 87–110. https://doi.org/10.1016/j.pmr.2009.08.003

[2] – *Open-Source leg*. (n.d.). https://www.opensourceleg.org/

[3] – Windrich, M., Grimmer, M., Christ, O., Rinderknecht, S., & Beckerle, P. (2016). Active lower limb prosthetics: a systematic review of design issues and solutions. *Biomedical engineering online*, *15*(Suppl 3), 140. https://doi.org/10.1186/s12938-016-0284-9

[4] – Amsaprabhaa, M., Jane, Y. N., & Nehemiah, H. K. (2021). A survey on spatio-temporal framework for kinematic gait analysis in RGB videos. *Journal of Visual Communication and Image Representation*, *79*, 103218. https://doi.org/10.1016/j.jvcir.2021.103218

[5] – Moltedo, Marta & Bacek, Tomislav & Verstraten, Tom & Rodriguez-Guerrero, Carlos & Vanderborght, Bram & Lefeber, Dirk. (2018). Powered ankle-foot orthoses: The effects of the assistance on healthy and impaired users while walking. Journal of NeuroEngineering and Rehabilitation. 15. 10.1186/s12984-018-0424-5.

[6] – *Joint range of motion during GAIT*. (n.d.). Physiopedia. https://www.physio-pedia.com/Joint_Range_of_Motion_During_Gait

[7] – Leal, A. G., Junior, De Andrade, R. M., & Filho, A. B. (2016). Series Elastic Actuator: Design, analysis and comparison. In *InTech eBooks*. https://doi.org/10.5772/63573

[8] – Park, K., Ahn, H. J., Lee, K. H., & Lee, C. H. (2020). Development and

Performance Verification of a Motorized Prosthetic Leg for Stair Walking. *Applied bionics and biomechanics*, *2020*, 8872362. https://doi.org/10.1155/2020/8872362

[9] – *Introduction to PID*. (n.d.). FIRST Robotics Competition Documentation. https://docs.wpilib.org/en/stable/docs/software/advanced-controls/introduction/introduction-to-pid.html

[10] – Lawson, Brian & Mitchell, Jason & Truex, Don & Shultz, Amanda & Ledoux, Elissa & Goldfarb, Michael. (2014). A Robotic Leg Prosthesis: Design, Control, and Implementation. Robotics & Automation Magazine, IEEE. 21. 70-81. 10.1109/MRA.2014.2360303.

[11] – Liu, C., Tagliabue, G., Raveendranathan, V., Houdijk, H. H., & Carloni, R. (n.d.). Control Architecture of a Variable Stiffness Prosthetic Knee for Energy Absorption and Restoration.

[12] – Wang, J., & Chortos, A. (2022). Control Strategies for soft robot Systems. *Advanced Intelligent Systems*, *4*(5). https://doi.org/10.1002/aisy.202100165

[13] – Sadeghijaleh, M. (2015). Voltage Control Strategy for Direct-drive Robots Driven by Permanent Magnet Synchronous Motors. *International Journal of Engineering*, *28*(5), 709-716.

[14] – Marchal-Crespo, L., Reinkensmeyer, D.J. Review of control strategies for robotic movement training after neurologic injury. *J NeuroEngineering Rehabil* 6, 20 (2009). https://doi.org/10.1186/1743-0003-6-20

[15] – *The importance of knowing your K-Level*. (n.d.). Össur Prosthetics. https://www.ossur.com/en-us/prosthetics/information/k-level

[16] – *IEC 80601-2-78:2019*. (n.d.). ISO. https://www.iso.org/standard/68474.html

[17] – Dephy. (2021). BA30 LiPO battery. In *Dephy*. https://dephy.com/docs/BA30/C_0002_DS_0001_V03_BA30DATA.pdf

[18] – Gd-Admin. (n.d.). *Best M35XX : 6 axis F/T load cell – Extra Thin Manufacturer and Supplier | SRI*. https://www.srisensor.com/. https://www.srisensor.com/m35xx-series-6-axis-load-cell-extra-thin-product/

[19] – *Open-Source leg*. (n.d.). https://www.opensourceleg.org/control

[20] – Coughlin, J. & Dephy, Inc. (2023). *ActPack 4.1*. https://dephy.com/docs/Dephy_DS_ActPack41.pdf

[21] – ams. (2018). *AS5048A/AS5048B Magnetic Rotary Encoder (14-Bit angular Position Sensor)*. https://eu.mouser.com/datasheet/2/588/AS5048_DS000298_4_00-2324531.pdf

[22] – *Basic Motion Test Script — Open-Source leg*. (n.d.). https://opensourceleg.readthedocs.io/en/latest/examples/basic_motion.html

[23] – *Finite State Machine Controller — Open-Source leg*. (n.d.). https://opensourceleg.readthedocs.io/en/latest/examples/finite_state_machine.html

Appendix I

Stakeholder Questions

General Questions:

- How would you like to use this leg?
- What requirements must the leg meet?
- Who would use the leg?
- Do you have any other ideas that are important for the functioning of the leg?

Specific Questions:

Researchers:

- In what different ways do you think this OSL can be used to advance research on lower limb prosthetics? (Gait analysis)

Engineers:

- Do you believe that the OSL can help in the development of building prosthetics?

Rehabilitation:

- Do you believe that a robotic knee and ankle joint would have an advantage over a mechanical leg prosthesis in terms of rehabilitation? (positive and negative points)

- Do you believe that the OSL can be used to identify abnormal gait in users?

Prosthetic Users:

- According to you, what are the differences between a microprocessor-controlled prosthesis and a mechanical prosthesis for this use? (positive and negative points)

Orthopaedic Technologist:

- Do you believe that the OSL can be used as a tool to improve the provision process?

Insurance:

- Do you believe it would be economically feasible to use the OSL as a prosthesis for amputees, or could it be better used as a tool for, for example, rehabilitation or research on lower limb prosthetics?

- When would the health insurer reimburse this leg?
- Do you believe that the OSL can be used as a tool to improve the provision process?

Appendix II

Position Control

Basic motion using position control, with a P-Controller and trajectory [22]:

```
import numpy as np
# for numerical operations

from opensourceleg.osl import OpenSourceLeg
# provides interface to control OSL

from opensourceleg.tools import units
# for unit conversion

osl = OpenSourceLeg(frequency=200)
# set the control frequency (how often control system updates)

osl.add_joint("knee", gear_ratio=9 * 83 / 18)
# specify gear ratio knee
osl.add_joint("ankle", gear_ratio=9 * 83 / 18)
# specify gear ratio ankle

def make_periodic_traj_func(period, minimum, maximum):
# function that generates lamba function
    amplitude = (maximum - minimum) / 2
    mean = amplitude + minimum
    return lambda t: amplitude * np.cos(t * 2 * np.pi / period) + mean

# generates periodic trajectory using cosine wave by taking a single
argument ‘t’ and calculating cosine wave value at that time.
# cos is used instead of sin when generating trajectories from peak
position as cos(0) = 1

ankle_traj = make_periodic_traj_func(10, -20, 20)
# defined to oscillate between -20 and 20 degrees with a period of 10
seconds

knee_traj = make_periodic_traj_func(10, 10, 90)
# defined to oscillate between 10 and 90 degrees with the same period

with osl:
    osl.home()
    input("Homing complete: Press enter to continue")
    osl.knee.set_mode(osl.knee.control_modes.position)
# control mode set to position control (knee)

    osl.ankle.set_mode(osl.ankle.control_modes.position)
# control mode set to position control (ankle)

    osl.knee.set_position_gains(kp=5)
# set proportional gains knee
```

```
    osl.ankle.set_position_gains(kp=5)
# set proportional gains ankle

    for t in osl.clock:
        osl.update()
        knee_setpoint = units.convert_to_default(knee_traj(t),
units.position.deg)
# set units to default (radians)

        ankle_setpoint = units.convert_to_default(ankle_traj(t),
units.position.deg)    # set units to default (radians)

        osl.knee.set_output_position(knee_setpoint)
# set desired position

        osl.ankle.set_output_position(ankle_setpoint)
# set desired position
        print(
            "Ankle Desired {:+.2f} rad, Ankle Actual {:+.2f} rad, Knee
Desired {:+.2f} rad, Ankle Desired {:+.2f} rad".format(
                ankle_setpoint,
                osl.ankle.output_position,
                knee_setpoint,
                osl.knee.output_position,
            ),
            end="\r",
        )

print("\n")
```

Impedance Control

Impedance control with a Finite State Controller and position control included [23]:

```
import numpy as np
# For numerical operations, to handle matrices and arrays

import opensourceleg.tools.units as units
# Provides unit conversion tools

from opensourceleg.control.state_machine import Event, State, StateMachine
# Provides classes for FSM

from opensourceleg.osl import OpenSourceLeg
# Provides OSL hardware control

offline_mode = False    # Set to true for debugging without hardware

# next define all tuneable FSM parameters (finite state machine)
# include impedance parameters for each state as well as transitions
between states

# ------------- TUNABLE FSM PARAMETERS ---------------- #
BODY_WEIGHT = 60 * 9.8

# STATE 1: EARLY STANCE
KNEE_K_ESTANCE = 99.372      # Knee stiffness early stance
```

```python
KNEE_B_ESTANCE = 3.180       # Knee damping early stance
KNEE_THETA_ESTANCE = 5       # Knee target angle early stance (position)
ANKLE_K_ESTANCE = 19.874     # Ankle stiffness early stance
ANKLE_B_ESTANCE = 0          # Ankle damping early stance
ANKLE_THETA_ESTANCE = -2     # Ankle target angle early stance (position)

LOAD_LSTANCE: float = -1.0 * BODY_WEIGHT * 0.25
# Load applied to leg

ANKLE_THETA_ESTANCE_TO_LSTANCE = 6.0
# Threshold angle where transition occurs

# STATE 2: LATE STANCE
KNEE_K_LSTANCE = 99.372       # Same as above, but for late stance
KNEE_B_LSTANCE = 1.272
KNEE_THETA_LSTANCE = 8
ANKLE_K_LSTANCE = 79.498
ANKLE_B_LSTANCE = 0.063
ANKLE_THETA_LSTANCE = -20
LOAD_ESWING: float = -1.0 * BODY_WEIGHT * 0.15

# STATE 3: EARLY SWING
KNEE_K_ESWING = 39.749        # Same as above, but for early swing
KNEE_B_ESWING = 0.063
KNEE_THETA_ESWING = 60
ANKLE_K_ESWING = 7.949
ANKLE_B_ESWING = 0.0
ANKLE_THETA_ESWING = 25
KNEE_THETA_ESWING_TO_LSWING = 50
KNEE_DTHETA_ESWING_TO_LSWING = 3

# STATE 4: LATE SWING
KNEE_K_LSWING = 15.899        # Same as above, but for late swing
KNEE_B_LSWING = 3.816
KNEE_THETA_LSWING = 5
ANKLE_K_LSWING = 7.949
ANKLE_B_LSWING = 0.0
ANKLE_THETA_LSWING = 15
LOAD_ESTANCE: float = -1.0 * BODY_WEIGHT * 0.4
KNEE_THETA_LSWING_TO_ESTANCE = 30
# ---------------------------------------------------- #

# specific parameters tuned for moderately paced walking gait (can be tuned
to suit use case)

# next enter main function, run_FSM_controller().
# first instantiate an OSL object, add joints, and add a loadcell
# (instantiating explained in adding actuator and loadcell tutorial)

def run_FSM_controller():
    """
    This is the main function for this script.
    It creates an OSL object and builds a state machine with 4 states.
    It runs a main loop that updates the state machine based on the
    hardware information and sends updated commands to the motors.
    """
    osl = OpenSourceLeg(frequency=200)
    osl.add_joint(name="knee", gear_ratio=41.4999,
offline_mode=offline_mode)                    # add knee joint
```

```
    osl.add_joint(name="ankle", gear_ratio=41.4999,
offline_mode=offline_mode)                                  # add ankle joint

    LOADCELL_MATRIX = np.array(
# NumPy array used for calibrating/configuring loadcell measurements
        [
            (-38.72600, -1817.74700, 9.84900, 43.37400, -44.54000,
1824.67000),
# col-1 coefficient x-axis force
            (-8.61600, 1041.14900, 18.86100, -2098.82200, 31.79400,
1058.6230),
# col-2 coefficient y-axis force
            (-1047.16800, 8.63900, -1047.28200, -20.70000, -1073.08800, -
8.92300),
# col-3 coefficient z-axis force
            (20.57600, -0.04000, -0.24600, 0.55400, -21.40800, -0.47600),
# col-4 coefficient x-axis torque
            (-12.13400, -1.10800, 24.36100, 0.02300, -12.14100, 0.79200),
# col-5 coefficient y-axis torque
            (-0.65100, -28.28700, 0.02200, -25.23000, 0.47300, -27.3070),
# col-6 coefficient z-axis torque
        ]
        # calibration coefficients determined during calibration process
where forces applied to load cell
        # calibration techniques (least squares regression) coefficients
determined to convert raw sensor readings
    )
    osl.add_loadcell(
        dephy_mode=False,
        offline_mode=offline_mode,
        loadcell_matrix=LOADCELL_MATRIX,
    )

# then create StateMachine instance

    fsm = build_4_state_FSM(osl)

# next configure the OSL log

    osl.log.add_attributes(container=osl, attributes=["timestamp"])     #
logging settings are applied to the osl object
    osl.log.add_attributes(                                             #
only attribute to be logged for the osl object is the "timestamp"
        container=osl.knee,
        attributes=[
            "output_position",
            "motor_current",
            "joint_torque",
            "motor_voltage",
            "accelx",
        ],
    )
    osl.log.add_attributes(
        container=osl.ankle,
        attributes=[
            "output_position",
            "motor_current",
            "joint_torque",
            "motor_voltage",
            "accelx",
        ],
    )
    osl.log.add_attributes(container=osl.loadcell, attributes=["fz"])
# force on z-axis logged
    osl.log.add_attributes(container=fsm.current_state,
attributes=["name"])                # log name assigned to each state

# MAIN LOOP
# everything set up, now home the OSL and enter main loop
# during each iteration of main loop, call update method for both the OSL
and the FSM
# write current impedance parameters for each joint to the hardware
# print statement also included for debugging

    with osl:
# resources acquired by osl object properly released after block of code
executes
        osl.home()
# instruct component to predefined home position
        fsm.start()
# start the FSM, initiating operation to control

        for t in osl.clock:
# iterates sequence time values provided by osl.clock. update/control
system's behaviour
            osl.update()
# update states for both OSL system and FSL controller at each time step
            fsm.update()

            if osl.knee.mode != osl.knee.control_modes.impedance:
                osl.knee.set_mode(mode=osl.knee.control_modes.impedance)
                osl.knee.set_impedance_gains()
            osl.knee.set_joint_impedance(
                K=units.convert_to_default(
                    fsm.current_state.knee_stiffness,
# updates the control parameters (such as impedance gains,
                    units.stiffness.N_m_per_rad,
# stiffness, damping, and position) for joints based on the
                ),
# current state of the FSM controller
                B=units.convert_to_default(
                    fsm.current_state.knee_damping,
                    units.damping.N_m_per_rad_per_s,
                ),
            )
            osl.knee.set_output_position(
                position=units.convert_to_default(
                    fsm.current_state.knee_theta, units.position.deg
                ),
            )

            if osl.ankle.mode != osl.ankle.control_modes.impedance:
                osl.ankle.set_mode(osl.ankle.control_modes.impedance)
                osl.ankle.set_impedance_gains()
            osl.ankle.set_joint_impedance(
                K=units.convert_to_default(
                    fsm.current_state.ankle_stiffness,
                    units.stiffness.N_m_per_rad,
                ),
                B=units.convert_to_default(
                    fsm.current_state.ankle_damping,
                    units.damping.N_m_per_rad_per_s,
                ),
            )
            osl.ankle.set_output_position(
                position=units.convert_to_default(
                    fsm.current_state.ankle_theta, units.position.deg
                ),
            )
            print(
                "Current time in state {}: {:.2f} seconds, Knee Eq {:.2f},
Ankle Eq {:.2f}, Fz {:.2f}".format(
                    fsm.current_state.name,
                    fsm.current_state.current_time_in_state,
                    fsm.current_state.knee_theta,
                    fsm.current_state.ankle_theta,
                    osl.loadcell.fz,
                ),
# prints status information to the console
                end="\r",
            )

# OSL library provides sensor values in default units (convert to prior if
library expects other units)
# can use units module and tools sub package to do this for example:
# ankle_angle_in_deg =
units.convert_from_default(osl.ankle.output_position, units.position.deg)

# BUILDING THE STATE MACHINE
# uses StateMachine functionality of opensourceleg.control module to make
a state machine with 4 states

def build_4_state_FSM(osl: OpenSourceLeg) -> StateMachine:
    """This method builds a state machine with 4 states.
    The states are early stance, late stance, early swing, and late swing.
    It uses the impedance parameters and transition criteria above.

    Inputs:
        OSL instance
    Returns:
        FSM object"""

    early_stance = State(name="e_stance")
    late_stance = State(name="l_stance")
    early_swing = State(name="e_swing")
    late_swing = State(name="l_swing")

# then assign impedance values for each state

    early_stance.set_knee_impedance_paramters(
        theta=KNEE_THETA_ESTANCE, k=KNEE_K_ESTANCE, b=KNEE_B_ESTANCE
    )
    early_stance.make_knee_active()
    early_stance.set_ankle_impedance_paramters(
        theta=ANKLE_THETA_ESTANCE, k=ANKLE_K_ESTANCE, b=ANKLE_B_ESTANCE
    )
    early_stance.make_ankle_active()
    late_stance.set_knee_impedance_paramters(
        theta=KNEE_THETA_LSTANCE, k=KNEE_K_LSTANCE, b=KNEE_B_LSTANCE
    )
    late_stance.make_knee_active()
    late_stance.set_ankle_impedance_paramters(
        theta=ANKLE_THETA_LSTANCE, k=ANKLE_K_LSTANCE, b=ANKLE_B_LSTANCE
    )
    late_stance.make_ankle_active()

    early_swing.set_knee_impedance_paramters(
        theta=KNEE_THETA_ESWING, k=KNEE_K_ESWING, b=KNEE_B_ESWING
    )
    early_swing.make_knee_active()
    early_swing.set_ankle_impedance_paramters(
        theta=ANKLE_THETA_ESWING, k=ANKLE_K_ESWING, b=ANKLE_B_ESWING
    )
    early_swing.make_ankle_active()

    late_swing.set_knee_impedance_paramters(
        theta=KNEE_THETA_LSWING, k=KNEE_K_LSWING, b=KNEE_B_LSWING
    )
    late_swing.make_knee_active()
    late_swing.set_ankle_impedance_paramters(
        theta=ANKLE_THETA_LSWING, k=ANKLE_K_LSWING, b=ANKLE_B_LSWING
    )
    late_swing.make_ankle_active()

# states defined, now define transition functions
# functions take osl instance as arguments and return a boolean when
transition criteria met
# example: first define transition from early stance to late stance based on
the loadcell z force
# and the ankle angle as:

    def estance_to_lstance(osl: OpenSourceLeg) -> bool:
        """
        Transition from early stance to late stance when the loadcell
        reads a force greater than a threshold.
        """
        assert osl.loadcell is not None
        return bool(
            osl.loadcell.fz < LOAD_LSTANCE
            and osl.ankle.output_position > ANKLE_THETA_ESTANCE_TO_LSTANCE
        )

# remaining transition functions defined similarly

    def estance_to_lstance(osl: OpenSourceLeg) -> bool:
        """
        Transition from early stance to late stance when the loadcell
        reads a force greater than a threshold.
        """
        assert osl.loadcell is not None
        return bool(
            osl.loadcell.fz < LOAD_LSTANCE
            and osl.ankle.output_position > ANKLE_THETA_ESTANCE_TO_LSTANCE
        )
    def lstance_to_eswing(osl: OpenSourceLeg) -> bool:
        """
        Transition from late stance to early swing when the loadcell
        reads a force less than a threshold.
        """
        assert osl.loadcell is not None
        return bool(osl.loadcell.fz > LOAD_ESWING)

    def eswing_to_lswing(osl: OpenSourceLeg) -> bool:
        """
        Transition from early swing to late swing when the knee angle
        is greater than a threshold and the knee velocity is less than
        a threshold.
        """
        assert osl.knee is not None
        return bool(
            osl.knee.output_position > KNEE_THETA_ESWING_TO_LSWING
            and osl.knee.output_velocity < KNEE_DTHETA_ESWING_TO_LSWING
        )

    def lswing_to_estance(osl: OpenSourceLeg) -> bool:
        """
        Transition from late swing to early stance when the loadcell
        reads a force greater than a threshold or the knee angle is
        less than a threshold.
        """
        assert osl.knee is not None and osl.loadcell is not None
        return bool(
            osl.loadcell.fz < LOAD_ESTANCE
            or osl.knee.output_position < KNEE_THETA_LSWING_TO_ESTANCE
        )

# next define events corresponding to state transitions using the Event
class

    foot_flat = Event(name="foot_flat")
    heel_off = Event(name="heel_off")
    toe_off = Event(name="toe_off")
    pre_heel_strike = Event(name="pre_heel_strike")
    heel_strike = Event(name="heel_strike")

# finally, make an instance of the StateMachine class and add the states,
events and transitions created
# the add_transition() method takes arguments of source state, a
destination state, an event, and the callback
# function defining when that transition occurs
# after, the FSM is fully built and can be returned

    fsm = StateMachine(osl=osl, spoof=offline_mode)
    fsm.add_state(state=early_stance, initial_state=True)
    fsm.add_state(state=late_stance)
    fsm.add_state(state=early_swing)
    fsm.add_state(state=late_swing)

    fsm.add_event(event=foot_flat)
    fsm.add_event(event=heel_off)
    fsm.add_event(event=toe_off)
    fsm.add_event(event=pre_heel_strike)
    fsm.add_event(event=heel_strike)
    fsm.add_transition(
        source=early_stance,
        destination=late_stance,
        event=foot_flat,
        callback=estance_to_lstance,
    )
    fsm.add_transition(
        source=late_stance,
        destination=early_swing,
        event=heel_off,
        callback=lstance_to_eswing,
    )
    fsm.add_transition(
        source=early_swing,
        destination=late_swing,
        event=toe_off,
        callback=eswing_to_lswing,
    )
    fsm.add_transition(
        source=late_swing,
        destination=early_stance,
        event=heel_strike,
        callback=lswing_to_estance,
    )
    return fsm

# finally, call main function:

if __name__ == "__main__":
    run_FSM_controller()
```