UNIVERSITY OF SOUTHAMPTON

Faculty of Engineering and Physical Science
School of Engineering

Life cycle assessment of an electric scooter battery - Materials, Manufacturing and Supply Chain Management

by

Abhinandan Thour - 32453515
MEng

April 2024

Academic Integrity Declaration

I declare that this coursework and the work presented in it is my own and has been generated by me as the result of my own original research.

I confirm that:

1. This work was done wholly or mainly while in candidature for a research degree at this University;
2. Where any part of this coursework has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated;
3. Where I have consulted the published work of others, this is always clearly attributed;
4. Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this coursework is entirely my own work;
5. I have acknowledged all main sources of help;
6. Where the coursework is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;
7. None of this work has been published before submission

0.1 Supply Chain

Currently, the production involves 10k units of existing Li-ion battery 18650 cell which utilises a Li-Co oxide as the cathode and graphite as the anode material with a lithium fluoride salt dissolved in an ethyl acetate electrolyte. The new 18650 Al-ion battery cell uses a 1xxx series aluminium anode, a nickel cathode and a proprietary aqueous salt electrolyte. The polyethene as a separator material and stainless-steel alloy for the container doesn't change. Thus defining the current Supplier strategy first could help identify what to change for the integration of the Al-ion battery without completely changing the supply chain.

The location is also crucial due to geopolitical situations and logistic costs, assuming the manufacturing company is based in Southampton, UK, located strategically to make seaborne transportation the preferred route for material arriving from outside the country.

0.1.1 Kraljic Matrix analysis of Li-ion and Al-ion battery packs

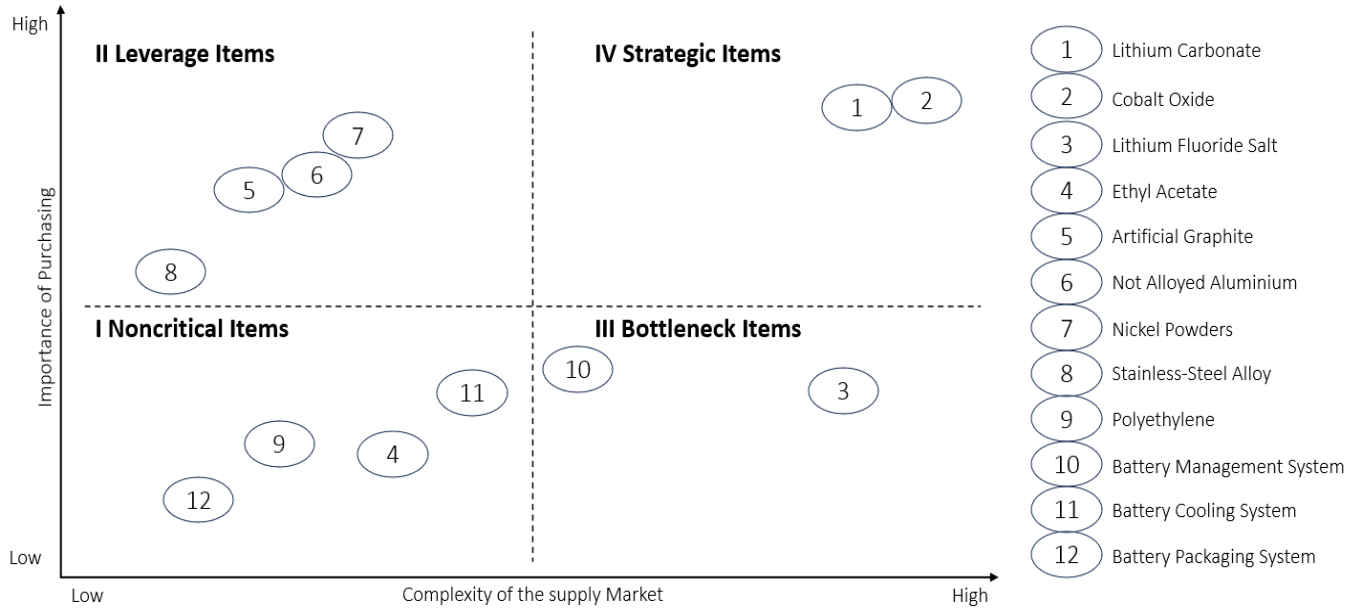

FIGURE 1: Kraljic Matrix for the supply chain of Li-ion & Ai-ion Batteries [Kraljic (1983)]

Strategic Items: **Lithium Carbonate (HS 2836.91) & Cobalt Oxide (HS 2822.00):** This is a critical raw material used in the production of Li-ion batteries. They have a high supply risk due to limited sources and geopolitical issues, and a high-profit impact because they directly influence the quality and cost of the final product. Ensure contract staggering to ensure a constant supply.

- **Strategy:** Develop strong relationships with multiple suppliers, explore long-term contracts to ensure supply continuity, invest in recycling technologies, and engage in strategic alliances or joint ventures in countries with these mineral resources.Kraljic (1983)
- **Possible Suppliers:**
  - **Lithium Carbonate:** Primarily Bisley (Australia), secondarily Chilean SQM (Chile) - Suppliers located in different countries and continents, diversification would ensure constant flow in case of changes in world geopolitical situation changes or unforeseen events such as on choke points. To reduce risk, if Bisley meets annual requirements through long-term contracts, annual agreements with Chilean SQM which include a rollover option. [Kraljic (1983)]
  - **Cobalt Oxide:** Primarily Glencore (Switzerland), secondarily ERG (Luxenburg), Glencore & ERG are one of the largest miners of cobalt in the DRC through their subsidiary Katanga Mining &

Metalkol RTR, China has a strong geopolitical influence in the DRC, existing EU companies would ensure the extraction of the material is ethically sourced. [European Parliament (2023)]

Bottleneck Items: Lithium Fluoride Salt (HS 2826.19) & Battery Management System (HS 8537.10): This chemical and electronic are critical components for the battery's performance but may come from a limited number of suppliers or for both, the production is affected by the raw material.

- **Strategy:** Identify alternative suppliers, invest in research for substitute materials, stock strategic reserves, and negotiate contracts that include volume insurance and risk-sharing provisions. Due to higher supplier strength, a more defensive strategy might be necessary so as not to jeopardize the relationship.
- **Possible Supplier:**
  - **Lithium Fluoride Salt:** Honeywell Specialty Chemicals (Germany/International) [Vest (2023)]
  - **Battery Management System:** Infineon Technologies (Germany).

Leverage Items: Artificial Graphite (HS 3801.10), Not Alloyed Aluminium (HS 7601.10), Nickely Powders (HS 7504.00) & Stainless-Steel Alloy (HS 7220.00): These are major components that dictate the performance of the battery packs. Although the supply risk is relatively low due to a growing number of suppliers, they have a high-profit impact.

- **Strategy:** Use competitive bidding and volume discounts, standardize designs to increase purchasing power and negotiate contracts with key suppliers to secure better terms and reliability.
- **Possible Supplier:**
  - **Artificial Graphite:** GrafTech International Ltd. (USA/International)
  - **Not Alloyed Aluminium:** All Foils (USA)
  - **Nickel Powders & Stainless-Steel:** Targray (USA)

  Targray (USA) can be used to supply Artificial Graphite, Aluminium, Nickel Powder, Electrolyte Solutions, Stainless-Steel cans and packaging material, this could be taken advantage of to leverage cost by increasing the purchased volume.

Non Critical Items: Polyethene (HS 3920.10), Battery Cooling System (HS 8419.50) & Battery Packaging System - Plastic (HS 3923.00) & Ethyl Acetate (HS 2915.31): These items are readily available and have multiple suppliers, with low impact on profits.

- **Strategy:** Standardize items to increase purchasing efficiency, automate reordering processes, and consolidate purchases to achieve lower prices through bulk buying. 90% big, 10% small local supplier.
- **Possible Suppliers:**
  - **Polyethene & Battery Packaging System:** Dow Chemical (USA), higher volume with one company would increase purchasing power.
  - **Battery Cooling System:** Tycorun Energy (USA)
  - **Ethyl Acetate:** Celanese (Mexico), Celanese is a major player this chemical solution. Ineos Europe Ag (UK) is a local supplier for diversification.

With the introduction of Al-ion battery packs, the supply chain would shift away from Strategic items to Leverage items due to the higher and diversified supply of the new materials. This would allow to change suppliers if geopolitical situations threaten the supply chain, such as the Cobalt situation where Congo is the major supplier, being a conflict material, buying Cobalt could contribute to violence and exploitation in the region and with the Chinese influence, constant supply is not guaranteed [RMI]. The proprietary aqueous solution introduced would reduce the bottleneck caused by the monopoly of some giant players in the industry or the dependence on raw materials. An own solution would also follow the suggested strategy of investing in a substitute solution.

0.2 Battery Manufacturing Conveyors

0.2.1 Question 1

Production Rate, Line Efficiency and Cycle time for current line:

We start by calculating the effective number of cells per pack that we need, accounting for the defect rate:

$$C_{\text{e-Li}} = \frac{C_{\text{Li}}}{1 - dr_{\text{Li}}} = 30.0075$$

Given that it takes 9 seconds to pick each cell, and an additional 15 seconds per pack, we calculate the total time per pack as:

$$T_{\text{pack-Li}} = \left(T_{\text{pick}} \cdot C_{\text{e-Li}}\right) + T_{\text{bin}} + \left(n_{\text{components}} \cdot F \cdot T_{\text{downtime}}\right) = 357.06 \text{ s}$$

This leads to the total cycle time:

$$T_c = \frac{T_{\text{pack-Li}} \cdot n_{\text{packs}}}{3600} = 991.85 \text{ hours}$$

Finally, we calculate the production rate per hour:

$$T_p = \frac{3600}{T_{\text{pack-Li}}} = \frac{3600}{357.06} = 10.08 \text{ units per hour}$$

The ideal time without any losses per pack is calculated as follows:

$$T_{\text{pack-Li-ideal}} = \left(T_{\text{pick}} \cdot C_{\text{e-Li-ideal}}\right) + T_{\text{bin}} = (9 \cdot 30) + 15 = 285 \text{ s}$$

This leads to an ideal total cycle time:

$$T_{c_{\text{ideal}}} = 791.66 \text{ hours}$$

which concludes with an ideal rate of production:

$$T_{p_{\text{ideal}}} = 12.63 \text{ units per hour}$$

This calculation allows us to determine the Line Efficiency E:

$$E = \frac{T_p}{T_{p_{\text{ideal}}}} = 79.81\%$$

To calculate the production rate in terms of annual demand, we use the following formula:

$$R_p = \frac{D_a}{W_y \cdot S_w \cdot H_{sh}} = \frac{10,000}{52 \cdot 7 \cdot 20} = 1.37 \text{ units per hour}$$

D_a is the annual demand, W_y is the number of weeks per year, S_w is the number of shifts per week, H_{sh} is the hours per shift. This rate is well below the maximum possible production rate.

Production Rate, Line Efficiency and Cycle time for Option 1 & 2:

For option 1, we analyze two identical yet separate production lines for Lithium and Aluminium, each tasked with producing 5000 battery packs.

The obtained data is presented in the table below:

TABLE 1: Option 1 - Production Rate, Line Efficiency & Cycle Time

Parameter	Lithium Battery Line	Aluminium Battery Line
Actual Cells per Pack	30.0075 cells	40.0200 cells
Actual Time per Pack	357.06 seconds	447.18 seconds
Actual Cycle per Batch	495.92 hours	621.08 hours
Actual Production Rate per Hour	10.08 ph	8.05 ph
Ideal Cells per Pack	30 cells	40 cells
Ideal Time per Pack	285 seconds	375 seconds
Ideal Cycle per Batch	395.83 hours	520.83 hours
Ideal Production Rate per Hour	12.63 ph	9.6 ph
Efficiency per Line	79.81%	83.85%

With both lines operational, the combined production rate is 18.13 units per hour. However, this rate decreases to 8.05 ph once all Lithium battery packs have been produced.

Option 2, instead, the rate of failure and cost increases due to the extra robot added to the manufacturing line. $C_{\text{e-Li}}$ and $C_{\text{e-Al}}$ do not change.

The total downtime T_{downtime} is calculated by considering the number of components, the failure rates for the Lithium and Aluminium lines, and the time associated with each failure. The formula is given by:

$$T_{\text{downtime}} = \left(n_{\text{components}} \cdot \left(\frac{F}{C_{\text{e-Li}}} + \frac{F}{C_{\text{e-Al}}} \right) \right) \cdot T_{\text{failure}} = 5.25 \text{ seconds per cell}$$

Thus, the total downtime for the Lithium battery pack is;

$$T_{\text{downtime Li}} = (T_{\text{downtime}} * C_{\text{e-Li}})/3600 = 218.72 \text{ hours}$$

Compared to Option 1, time per pack has increased from 9 seconds to 10 seconds due to the extra length that the aluminium cells need to do to reach its robot. The actual time per pack is:

$$T_{\text{pack-Li}} = (T_{\text{pick}} + (\frac{T_{\text{bin}}}{C_{\text{e-Li}}} + \frac{T_{\text{bin}}}{C_{\text{e-Al}}}) \cdot C_{\text{e-Li}}) + (T_{\text{downtime}} \cdot C_{\text{e-Li}}) = 483.90 \text{ seconds}$$

The ideal time per pack is:

$$T_{\text{pack-Li-ideal}} = (T_{\text{pick}} + (\frac{T_{\text{bin}}}{C_{\text{e-Li}}} + \frac{T_{\text{bin}}}{C_{\text{e-Al}}}) \cdot C_{\text{e-Li}}) = 326.25 \text{ seconds}$$

T_p, T_c & E have been calculated as shown in the equations for the current option and the results have been shown in the table below:

TABLE 2: Option 2 - Production Rate, Line Efficiency & Cycle Time

Parameter	Lithium Battery Line	Aluminium Battery Line
Actual Time per Pack	483.80 seconds	645.23 seconds
Actual Cycle per Batch	671.95 hours	896.15 hours
Actual Production Rate per Hour	7.44 ph	5.57 ph
Ideal Time per Pack	326.25 seconds	435 seconds
Ideal Cycle per Batch	453.12 hours	604.16 hours
Ideal Production Rate per Hour	11.03 ph	8.27 ph
Efficiency per Line	67.43%	67.42%

The total production rate in terms of annual demand is still the same but divided as $R_p = 0.69$ units/hour per type of battery pack.

Implementation/Operational Consideration for each option:

We have analyzed two categories of costs: Purchase Cost and Operational Cost.

Purchase cost is the total of all conveyors and robots currently owned. This includes 3 conveyors & 1 robot with a total value of £155k. This has been subtracted in all the following analyses to get a clear view of how much an implementation would cost.

Operational cost is divided into two sub-categories: Use Cost and Repair Cost.

$$C_{\text{conveyors-use}} = n_{\text{conveyor}} \cdot C_{\text{Repair rate conveyors}} \cdot (T_{\text{c-Li}} - T_{\text{downtime - Li}})$$

The repair cost for the conveyors is defined by:

$$C_{\text{conveyor-repair}} = n_{\text{conveyor}} \cdot C_{\text{Repair rate Robot}} \cdot \left(\frac{T_{\text{downtime - Li}}}{n_{\text{components}}} \right)$$

The same is done for Operational Robot costs.

A clear comparison can be made using the following table.

TABLE 3: Cost analysis of every option for 10'000 battery packs for the first 12 months

Line	Purchase Cost	Operational Cost	Total Cost
Current Option	£ 0	£ 233.3k	£ 233.3k
Option 1	£ 155k	£ 238.3k	£ 393.3k
Option 2	£ 125k	£ 450.6k	£ 575.6k

Latest Operational Day for each option to fulfil production requirement:

TABLE 4: Nearest Day new option could you operational to the 12-month unit production requirement

Line	Production Day	Final Day
Option 1	31.05 = 32 days	31st Nov
Option 2	44.8 = 45 days	17th Nov

The above calculation does not consider any production stops caused by Christmas or New Year.

Unit Processing Cost for the 12 months:

Unit processing Cost is defined as $C_p = \frac{C_o \cdot T_p}{60}$

TABLE 5: Unit Processing Cost for each line for first 12 months

Line	Li - C_o	Al - C_o
Current Option	23.32 £/unit	-
Option 1	38.82 £/unit	39.82 £/unit
Option 2	26.12 £/unit	39.00 £/unit

0.2.2 Question 2

Explanation of the preferred configuration:

The analysis compares the current operational line with two alternative proposed lines indicates a decrease in efficiency as the number of components increases per line. This suggests that optimizing line design for fewer components increases efficiency, thus reducing operational time and cost.

Using a manufacturing strategy similar to Option Two, where different battery chemistry's are produced on the same line, would offer a lower initial purchase cost due to the use of already-owned components. However, this approach results, due the increased number of failures associated with this setup, to a higher total cost over time.

Defining the most suitable option is challenging without clear criteria. Is the goal to maximize production volume in the shortest possible time? Is demand going to be stagnant over the long term? Or is minimizing costs the main concern (but by still improving the current line), even if it means accepting longer production times?

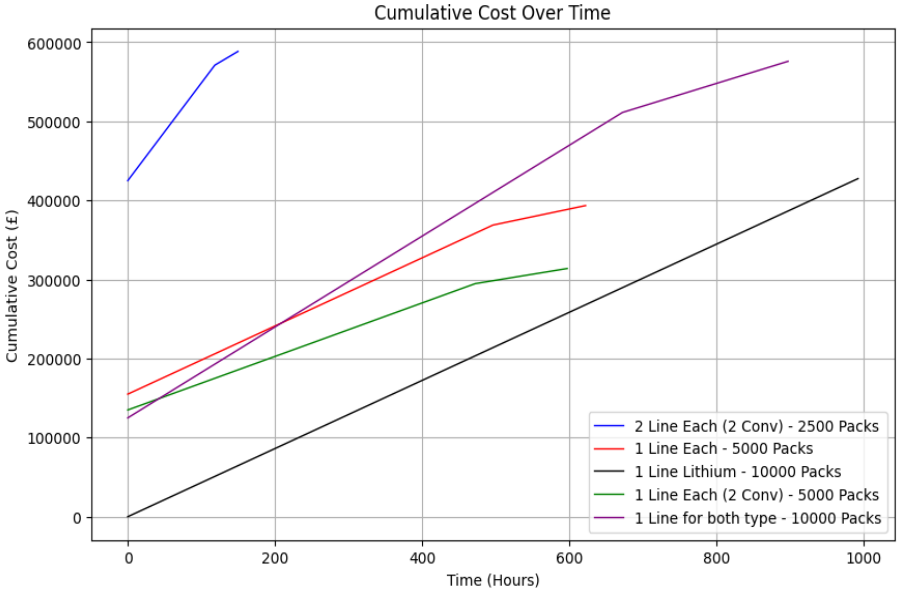

FIGURE 2: Eutrophication (kg PO_4 eq) Comparison between different configuration.

The unit processing costs for the first 12 months, in the table below, show the minimum price at which the battery must be sold to recover the initial purchase cost and the operational cost.

TABLE 6: Unit Processing Cost for each line

Line	**Li -** T_p	**Al -** T_p	**Li -** C_o	**Al -** C_o
Option 1	10.08 ph	8.05 ph	38.82 £/unit	39.82 £/unit
1 Line Each (2 Conv - 1 Rob)	10.61 ph	8.38 ph	30.99 £/unit	31.74 £/unit
2 Line Each (2 Conv - 1 Rob)	21.23 ph	16.77 ph	117.16 £/unit	117.91 £/unit

Based on the data, the best approach appears to favour smaller, separate production lines which improve efficiency. Separate lines also introduce an additional layer of safety; if one line stops, it doesn't stop the entire production, as the other would continue operating. In Figure 2, a setup with two manufacturing lines, each with two conveyors and one robot, can reduce manufacturing time by 70% but overall costs are 1.5 times more than Option 1. Although Option 1 was favoured compared to Option 2, due to its higher efficiency, lower cost, and shorter cycle times, reducing the number of conveyors in Option 1 increases the units produced per hour

by an average of 5% and decreases costs by 25%. A critical cost factor is the number of robots, which are 12.5 times more expensive than conveyors, they increase the unit process cost hugely to pay off the initial purchase cost in the first 12 months.

Based on this information, a compromise with cost and production seems to be the solution that has 2 conveyors with 1 robot for each battery type.

Preferred Configuration:

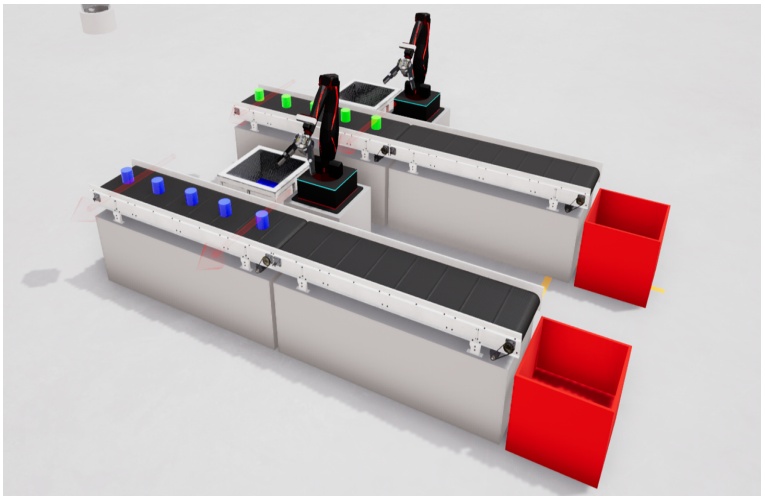

FIGURE 3: Preferred Configuration - 2 Conveyors & 1 Robot per Battery Type.

Discuss the digital twin environment:

- Advantages

Higher Accuracy in Simulation: A digital twin allows for the creation of a virtual model that simulates the production process, supporting the optimization and troubleshooting of the system before physical changes are made or because of failure. [Shahzad (2022)]

Live Monitoring and Adjustment: The digital twins offer the possibility of live data analysis to monitor performance, allowing for real-time adjustments. Thus improving efficiency while providing faster responses to any eventuality of change or malfunction. [Rocha (2021)]

Predictive Maintenance and Downtime Reduction: It can help predict with greater accuracy the time when a machine could fail, or rather when it might need maintenance, cutting down on downtime and extending the life of the equipment.

Training and Safety: Virtual environments can be used in training without risks associated with training in a physical setting. This helps make sure that operators are very well trained on equipment that might be unsafe.

- Disadvantages

Complexity and Cost of Implementation: Setting up and maintaining a digital twin requires significant technological and expertise investment, both in initial setup and ongoing operations.

Data Security and Privacy: The increased use of Internet of Things devices and constant data could be a security and privacy concern.

System Response and Accuracy: There can be discrepancies between the virtual model and the actual real-world model due to uncontrollable factors such as human error, unforeseen equipment behaviours or errors not captured by the model, machine quality and real-life accuracy.

Component Parameters Variance: Theoretical component parameters such as cycle times, efficiency, and downtime are often very ideal and they can vary in practice due to environmental conditions or material properties, wear and tear, etc....

0.3 Battery Life Cycle Assessment

0.3.1 Goal Definition

Intended Application:

The study tried to provide preliminary data with which the company could decide whether to pursue an alternative battery chemistry in an electric scooter. To assess the possible environmental impacts that may happen with the introduction of the new battery chemistry, a Life-Cycle Assessment (LCA) was conducted using the ISO14040 standard about specific impact categories of concern, which include energy drawn, global warming potential, acidification, and eutrophication.

Reasons for carrying out the study:

The reason for carrying out the study is to investigate and assess the benefit of replacing the company's existing Li-ion battery product with a new Al-ion battery product. This evaluation was proposed by the potential for the Al-ion battery to be a more sustainable option from resource supply, manufacturing, and environmental perspectives.

Intended Audience:

This assessment will allow the intended audience, thus the company's board of directors to make an informed decision regarding the sustainability and feasibility of the Al-ion battery technology compared with the existing Li-ion batteries.

Comparative Assertation:

The LCA is a comparative assertion of the Al-ion battery against the baseline Li-ion battery focused on the cradle-to-gate part of the product system for both battery types and will not be disclosed to the public.

Commissioner of the Study:

The commissioner of the Life Cycle Assessment is the company's board of directors.

0.3.2 Scope Definition

Product System and Function:

Product System: A rechargeable battery used in an electric scooter.

Product Function: To store and provide electrical energy efficiently and safely for electric scooters. Additionally, it is designed to be recharged. The batteries must support the scooter's operational requirements, reliability of the energy capacity, weight, and longevity, all of which directly impact the scooter's performance, range, and usability.

Functional unit and reference flow:

Functional Unit: Defined as the capacity of an electric scooter battery pack of 300 Wh at 48 V over a specified time and usage scenario, such as powering a scooter for tot amount of miles.

Reference Flow: Considering the production of a fixed number of battery units, 5,000 Li-ion battery packs and 5,000 Al-ion battery packs of 300 Wh at 48 V, to help compare both batteries.

System Boundary:

The lifecycle of the scooter battery from cradle-to-grave involves everything from the purchase of raw materials, manufacturing and end of life. The manufacturing of a battery cell includes the anode, cathode, electrolyte, and container & separator. The same is then integrated with the Battery Management System and Packaging & Cooling System to form the complete battery pack.

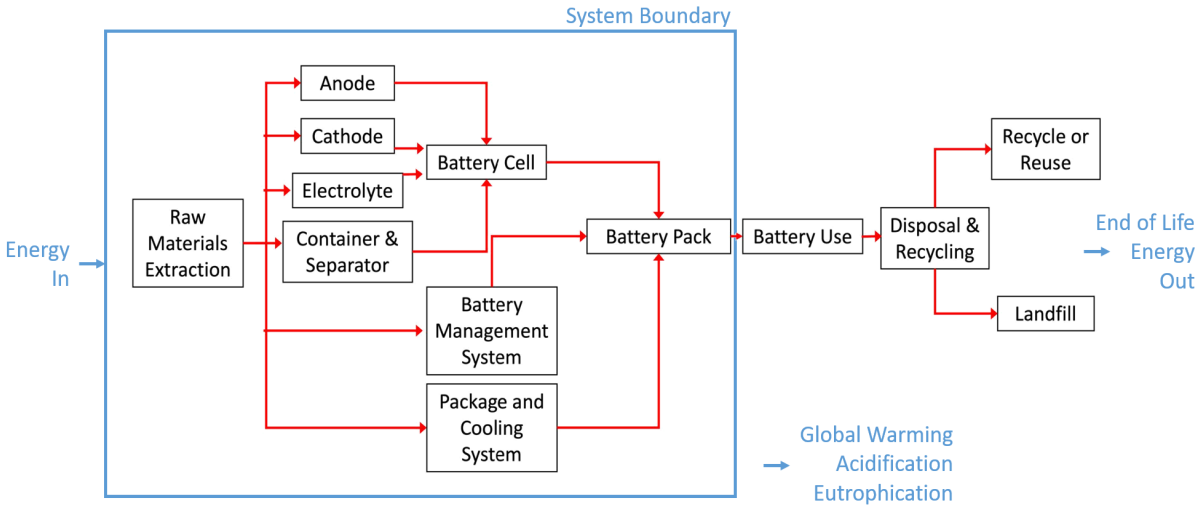

FIGURE 4: Product System and Boundary for the comparative assertion of an electric scooter battery chemistries.

Impact Categories, Methodology and Interpretation:

For this study, the selected impact categories are Energy (MJ), global warming potential (kg CO_2), acidification (kg SO_2 eq), and eutrophication (kg PO_4 eq).

The methodology consists of reviewing raw data collected by an external consultant specializing in life cycle assessment, specifically focusing on these four impact categories. A sensitivity analysis across the four impact categories on the number of cells in the Al-Ion pack to ±5 cells is performed.

Data Quality Requirements:

The data was obtained from a commercial subscription to a life cycle assessment software platform which is updated annually. The resolution of the data from all sources was determined by three significant figures for each of the respective impact categories. The accuracy of the data from each impact category varied but was reported to be within a standard error of ± 10% across each of the four impact categories published by the software company.

Assumptions and Limitation:

The LCA is limited to four categories of impacts: energy use, global warming potential, acidification, and eutrophication. These categories represent the environmental considerations of this study. The LCA's assumption is that of a 'cradle-to-gate' approach, it neglects the battery life cycle's use, disposal, and recycling phases. It is not assumed the effect of any potential future technology could change or reduce the impact categories. It is assumed that the quality of the product, material, and performance of each battery pack is consistent and identical. In addition, the manufacturing process is presumed to avoid any impurities, defects or cracks that could lead to variable emissions or environmental impacts.

A limitation of this study comes from the available data, particularly for aluminium-ion (Al-ion) batteries, which have seen limited use to date and with ongoing research papers. If more data is available to support

either option, Revisiting this LCA in the future is suggested. Another limitation is the monetary data that could impact support one battery over the other.

Type and Format of report and critical review:

The LCA is a 9-page written PDF document in accordance with ISO 14040 detailing the data sets used and presented to give as much information as possible to allow subsequent decisions and prepare for possible more in-depth LCA beyond the limit of the presented report. As the document is an internal document designed for the company's board of directors to assess the decision for the company to manufacture Al-ion batteries, an external critical review is not required.

0.3.3 Life Cycle Inventory Analysis

The raw data collected by the life cycle assessment practitioner for both Li-ion and Al-ion batteries is presented in Table 1 and Table 2, respectively. For each battery type, cradle-to-gate cumulative data has been provided to show the total environmental impacts from the raw material extraction phase to the factory gate.

TABLE 7: Raw Impact Data for a Li-Ion Scooter Battery

Component	Energy (MJ)	Global Warming (kg CO_2 eq)	Acidification (kg SO_2 eq)	Eutrophication (kg PO_4 eq)
Single Cell Anode	0.236	0.200	0.000489	0.000489
Single Cell Cathode	2.30	0.385	0.000707	0.000236
Single Cell Electrolyte	0.694	0.030	0.000109	5.43E-05
Container and Separator	0.0725	0.00217	7.25E-06	0.00
Battery Management System	129	8.75	0.0886	0.0375
Packaging and Cooling System	113	8.59	0.0201	0.0859
30 Cell Pack Manufacturing	49.5	4.03	0.0196	0.0141
Cradle-to-gate cumulative data	391	39.9	0.168	0.161
Pack End of Life	8.15	1.03	0.0304	0.00163

TABLE 8: Raw Impact Data for an Al-Ion Scooter Battery

Component	Energy (MJ)	Global Warming (kg CO_2 eq)	Acidification (kg SO_2 eq)	Eutrophication (kg PO_4 eq)
Single Cell Anode	1.48	0.0909	0.000236	0.000236
Single Cell Cathode	0.550	0.0717	0.000637	0.000106
Single Cell Electrolyte	0.116	0.0102	5.36E-05	2.26E-05
Container and Separator	0.0651	0.00115	6.45E-06	0.00
Battery Management System	129	8.75	0.0886	0.0375
Packaging and Cooling System	113	8.59	0.0201	0.0859
40 Cell Pack Manufacturing	56.7	4.62	0.0251	0.0132
Cradle-to-gate cumulative data	387	28.9	0.171	0.151
Pack End of Life	6.90	1.02	0.0211	0.00142

0.3.4 Life Cycle Impact Assessment

This section presents plots that show the variations in the different environmental impact categories, Energy Drawn [Figure 5], Global Warming Potential [Figure 6], Acidification [Figure 7], and Eutrophication [Figure 8], across three stages of the battery's lifecycle. These stages include the combined impacts of the Anode, Cathode, Electrolyte, Container, and Separator for each cell in a pack; the complete battery pack excluding manufacturing; and the complete battery pack including manufacturing and End of Life Emissions.

This analysis demonstrates that from an environmental perspective, using aluminium in battery production offers overall better outcomes, though the battery pack, including manufacturing, shows a 2.019% higher impact in the Acidification category compared to lithium.

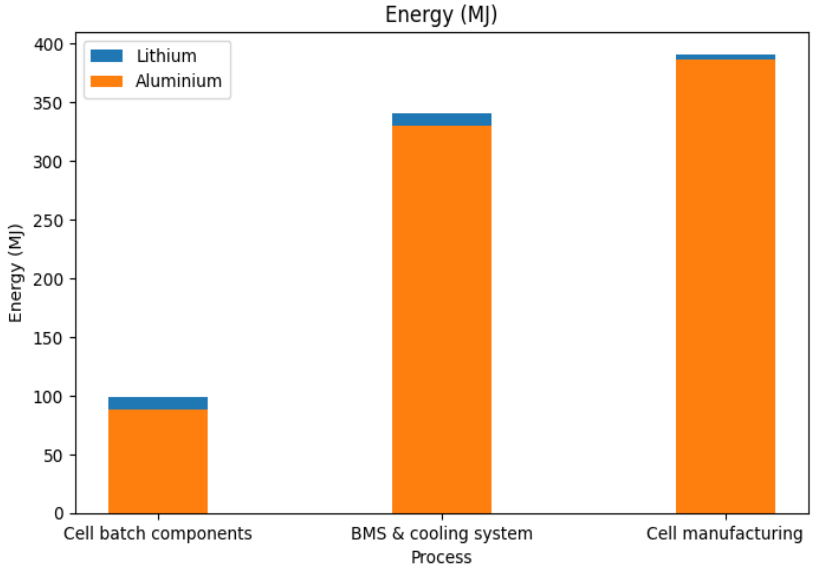

FIGURE 5: Energy needed(MJ) Comparison between Li-ion and Al-ion.

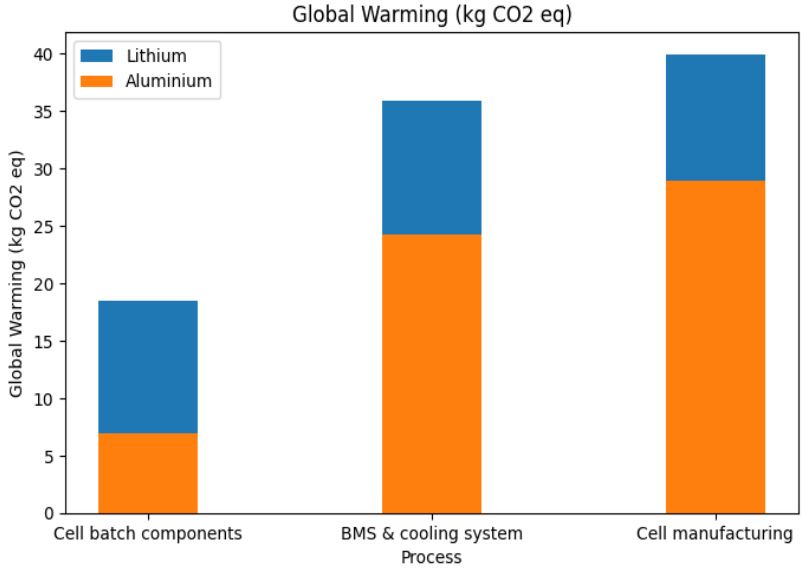

FIGURE 6: Global Warming (kg CO_2 eq) Comparison between Li-ion and Al-ion.

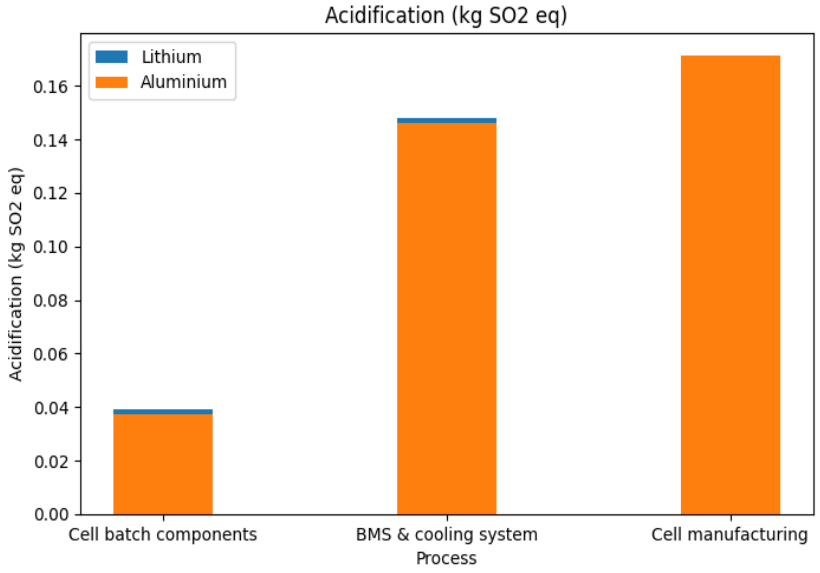

FIGURE 7: Acidification (kg SO_2 eq) Comparison between Li-ion and Al-ion.

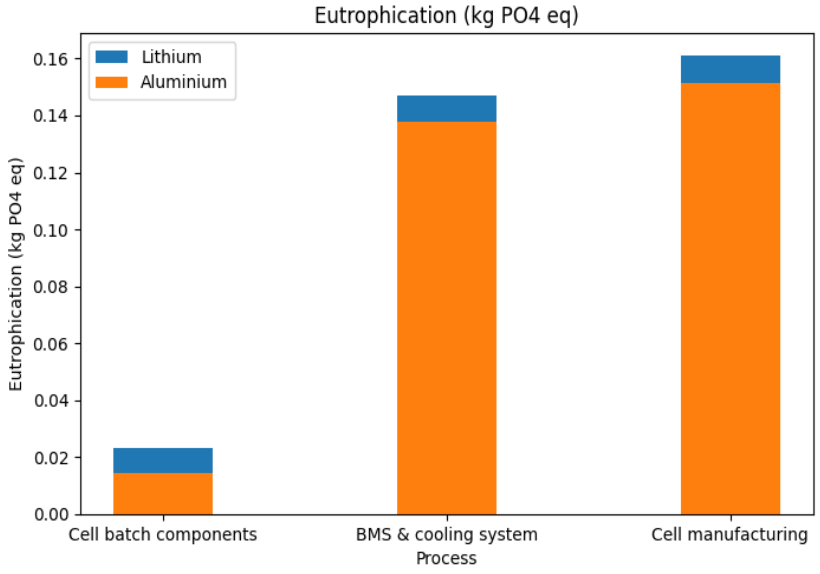

FIGURE 8: Eutrophication (kg PO_4 eq) Comparison between Li-ion and Al-ion.

Finally, Figure 9 shows the percentage magnitude difference between the Li-ion baseline for all the impact categories throughout the cradle-to-gate process. Another data that needs to be discussed is the End of Life Emission, this being 15.33 % lower for Aluminium, the energy saved and the lower emission at a larger scale would make an important impact on the environment.

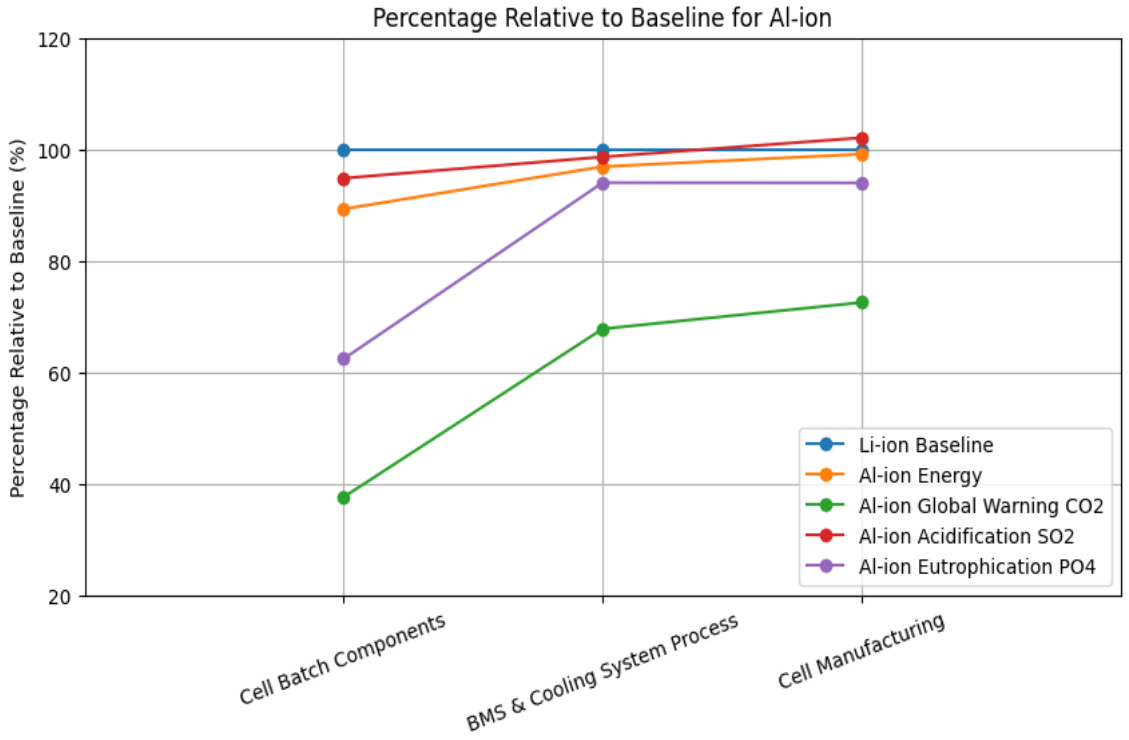

FIGURE 9: Eutrophication (kg PO_4 eq) Comparison between Li-ion and Al-ion.

0.3.5 Interpretation

The Al-ion battery cell components' life cycle impact assessment shows lower emissions than those of Li-ion batteries, particularly regarding global warming and eutrophication. This observation holds even when both battery types utilize identical battery management and cooling systems, showing a clear advantage for Al-ion technology. In addition, the benefits of Al-ion batteries become even more pronounced when considering their production scale, despite the manufacturing process for Al-ion batteries being approximately 14.54% more intensive than that for Li-ion batteries

The overall analysis of the impact categories shows a 26.78% reduction in global warming potential, a 6.49% reduction in eutrophication for Al-ion batteries, alongside a modest 1.18% decrease in energy use and a 2.09% increase in acidification. Therefore, Al-ion batteries offer a definitive environmental advantage within the cradle-to-gate system boundary.

A sensitivity analysis was also conducted to evaluate the environmental impacts of varying the number of cells in the Al-ion battery by ±5 cells, as shown in Figure 10. This analysis uses a standard error of ±10% across all data sets. The results show that the Aluminium-ion battery, with 50% more cells, consistently requires higher energy inputs and shows increased acidification at every stage of the battery pack's life cycle. Although these increases, global warming potential and eutrophication remain lower than traditional aluminium-based systems. This is largely attributable to the substantial contributions of the battery management system, cooling system, and battery pack manufacturing processes across all impact categories compared to the cell structure instead.

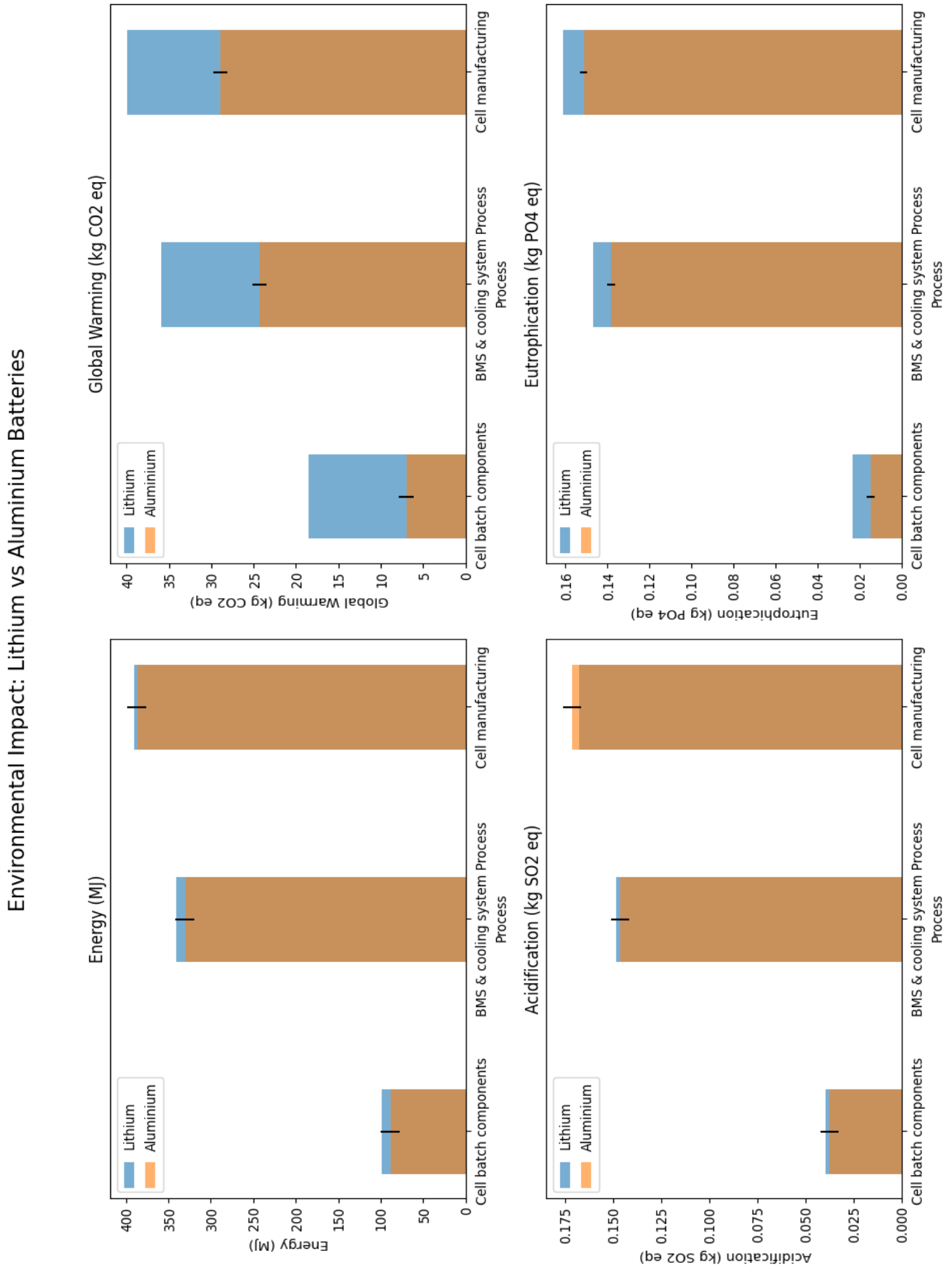

FIGURE 10: Eutrophication (kg PO_4 eq) Comparison between Li-ion and Al-ion.

0.4 Overall Recommendation

Starting with the supply chain analysis, aluminium seems to bring a strong case in diversifying the production type, especially due to the competitiveness of lithium and cobalt which could increase future costs and with the current geopolitical situation, having a plan to reduce the company's risk is fundamental to avoid complications. Aluminium and the proprietary solution would allow the company to have more leverage in sourcing the battery material and more independence for the latter as owning a material would allow it not to depend on external factors.

One point to be made is the increase in mass, $Mass_{\text{lithium-cell}} = 43$ g, while for aluminium it is $Mass_{\text{aluminium-cell}} = 55$ g. This signifies that for 5000 battery packs each, the total mass for lithium is 6451.59 kg and for aluminium is 11005.50 kg, an increase of 170%. This increase for aluminium implies that logistical costs for procuring raw materials and shipping final products are higher compared to lithium. This factor must be considered when evaluating the overall feasibility and cost-effectiveness of switching to or incorporating aluminium in the production process.

This analysis did not consider the addition of this cost to the total cost, which might or might not suggest reducing the total pack of aluminium to be produced.

From a cost analysis perspective, the calculations indicate a slightly higher unit processing cost due to an increased number of cells. However, the analysis did not prioritize whether speed of production or cost-effectiveness is more important. A compromise is achieved by reducing the lithium manufacturing line to two conveyors and replicating this structure for aluminium, as shown in Figure 3. This configuration results in lower unit production costs and higher efficiency due to fewer components. It is also 40% more time efficient compared to the current operating line. The restructuring and the addition of another assembly line have significantly contributed to these improvements. The proposed layout shows a possible balance between time and cost efficiencies. If time is deemed more important, doubling the number of lines would be suggested, provided the increase in unit production costs is acceptable.

From an environmental perspective, the introduction of aluminium in Al-ion batteries brings significant benefits, as shown in Figure 9. These types of batteries significantly reduce global warming potential and eutrophication. There is also a slight reduction in energy consumption and a minor increase in acidification, which are considered acceptable given the major reductions in global warming potential. However, this assessment may need to be revisited if an increase in production leads to unsustainable emissions.

Further research is essential to fully understand the life cycle and implementation impacts of Al-ion batteries to make an accurate decision, especially on factors that could affect overall sustainability and efficiency:

1. **Further Material Analysis:** Research to find cheaper, better-performing, and maybe greener replacements of Al-ion battery components.
2. **Battery Pack Weight Optimization:** Investigate how the weight and design of Al-ion battery packs affect the energy consumption of electric scooters. Lighter or more efficiently designed packs may lead to reduced energy loss and increased vehicle range.
3. **Techno-Economic Analysis:** Conduct a techno-economic analysis to compare the overall costs and benefits of Al-ion and Li-ion battery technologies over their entire life cycles. Consider factors such as initial capital costs, operational expenses and potential revenue from recycling or reusing battery materials.
4. **Regulatory Impact Studies:** Conduct studies on the impact of current and potential future regulations on the adoption and innovation in Al-ion battery technology.

In essence, if the higher logistic cost caused by the increase in weight is sustainable, the integration of the use of Aluminium-ion batteries is recommended.

0.5 Quanser Code

```
# Note use this method to get your qvl libraries to ensure you're using the
# latest version in GitHub. It is inserted first in the list to take precedence
# over all other libraries in your python path.
import sys
sys.path.insert(0, "../")

from qvl.qlabs import QuanserInteractiveLabs
from qvl.conveyor_curved import QLabsConveyorCurved
from qvl.conveyor_straight import QLabsConveyorStraight
from qvl.widget import QLabsWidget
from qvl.delivery_tube import QLabsDeliveryTube
from qvl.basic_shape import QLabsBasicShape
from qvl.shredder import QLabsShredder
from qvl.generic_sensor import QLabsGenericSensor
from qvl.qarm import QLabsQArm
from qvl.real_time import QLabsRealTime
import pal.resources.rtmodels as rtmodels
import time
import math
import struct
import numpy as np
import random
import cv2
import os
from random import randrange
import pandas as pd
import shutil
from controller import *
########## Backup Creation ###########
destination = "\\\\filestore.soton.ac.uk\\users\\" + os.getlogin() + "\\mydocuments\\Quanser Labs Documents"
def saveFile():

    currentFilePath = os.path.realpath(__file__)
    fileName = os.path.basename(currentFilePath)

    if not os.access(destination, os.F_OK):
        os.mkdir(destination, 0o700)
    saveLocation = os.path.join(destination, fileName)
    shutil.copy(src=currentFilePath, dst=saveLocation)
saveFile()
########## Main setup script ##########

# Core variables
scriptStartTime = time.strftime("%d_%m_%Y-%H.%M.%S", time.gmtime())
scriptStartTimeSec = int(time.time())

waitTime = 1.0

qlabs = QuanserInteractiveLabs()
print("Connecting to QLabs...")
try:
    qlabs.open("localhost")
except:
    print("Unable to connect to QLabs")

print("Connected")

qlabs.destroy_all_spawned_actors()
QLabsRealTime().terminate_all_real_time_models()


__qalDirPath = os.environ['RTMODELS_DIR']

QARMS = os.path.normpath(
    os.path.join(__qalDirPath, 'QArms'))

#region : create conveyors and their supports and tube
#region : create conveyors and their supports and tube
positionSecond = 1

########## Create two conveyors ##########
firstConvey = QLabsConveyorStraight(qlabs)
num = firstConvey.spawn_id_degrees(actorNumber = 0,
                                   location = [-0.12, 0, 0.3],
                                   rotation = [0, 0, 0],
                                   scale = [1,1,1],
                                   configuration = 2)
firstConvey.set_speed(0.1)

# secondConvey = QLabsConveyorStraight(qlabs)
# num = secondConvey.spawn_id_degrees(actorNumber = 1,
#                                     location = [-1.1, 0, 0.3],
#                                     rotation = [0, 0, 0],
#                                     scale = [1,1,1],
#                                     configuration = 2)
# secondConvey.set_speed(0.1)
```

```
thirdConvey = QLabsConveyorStraight(qlabs)
num = thirdConvey.spawn_id_degrees(actorNumber = 3,
                                   location = [0.85, 0, 0.3],
                                   rotation = [0, 0, 0],
                                   scale = [1,1,1],
                                   configuration = 2)
thirdConvey.set_speed(0.1)


### SECOND CONVEYER BELT
########## Create two conveyors ##########
fourthConvey = QLabsConveyorStraight(qlabs)
num = fourthConvey.spawn_id_degrees(actorNumber = 70,
                                   location = [-0.12, positionSecond, 0.3],
                                   rotation = [0, 0, 0],
                                   scale = [1,1,1],
                                   configuration = 2)
fourthConvey.set_speed(0.1)

# secondConvey = QLabsConveyorStraight(qlabs)
# num = secondConvey.spawn_id_degrees(actorNumber = 1,
#                                     location = [-1.1, 0, 0.3],
#                                     rotation = [0, 0, 0],
#                                     scale = [1,1,1],
#                                     configuration = 2)
# secondConvey.set_speed(0.1)

sixthConvey = QLabsConveyorStraight(qlabs)
num = sixthConvey.spawn_id_degrees(actorNumber = 73,
                                   location = [0.85, positionSecond, 0.3],
                                   rotation = [0, 0, 0],
                                   scale = [1,1,1],
                                   configuration = 2)
sixthConvey.set_speed(0.1)

# Create a simple supports for the conveyors
firstStand = QLabsBasicShape(qlabs)
firstStand.spawn_id_and_parent_with_relative_transform(actorNumber = 98,
                                                       location = [0.5, 0, -0.15],
                                                       rotation = [0, 0, 0],
                                                       scale = [0.95, 0.3, 0.3],
                                                       configuration = 0,
                                                       parentClassID = firstConvey.classID,
                                                       parentActorNumber = 0,
                                                       parentComponent = 0,
                                                       waitForConfirmation = True)
firstStand.set_material_properties(color = [0.3, 0.3, 0.3],
                                   roughness = 0.4,
                                   metallic = False)

# secondStand = QLabsBasicShape(qlabs)
# secondStand.spawn_id_and_parent_with_relative_transform(actorNumber = 99,
#                                                         location = [0.5, 0, -0.15],
#                                                         rotation = [0, 0, 0],
#                                                         scale = [0.95, 0.3, 0.3],
#                                                         configuration = 0,
#                                                         parentClassID = secondConvey.classID,
#                                                         parentActorNumber = 1,
#                                                         parentComponent = 0,
#                                                         waitForConfirmation = True)
# secondStand.set_material_properties(color = [0.3, 0.3, 0.3],
#                                     roughness = 0.4,
#                                     metallic = False)

thirdStand = QLabsBasicShape(qlabs)
thirdStand.spawn_id_and_parent_with_relative_transform(actorNumber = 990,
                                                       location = [0.5, 0, -0.15],
                                                       rotation = [0, 0, 0],
                                                       scale = [0.95, 0.3, 0.3],
                                                       configuration = 0,
                                                       parentClassID = thirdConvey.classID,
                                                       parentActorNumber = 3,
                                                       parentComponent = 0,
                                                       waitForConfirmation = True)
thirdStand.set_material_properties(color = [0.3, 0.3, 0.3],
                                   roughness = 0.4,
                                   metallic = False)

### SUPPORT FOR SECOND CONVEYER
# Create a simple supports for the conveyors
fourthStand = QLabsBasicShape(qlabs)
fourthStand.spawn_id_and_parent_with_relative_transform(actorNumber = 798,
                                                       location = [0.5, positionSecond, -0.15],
                                                       rotation = [0, 0, 0],
                                                       scale = [0.95, 0.3, 0.3],
                                                       configuration = 0,
```

```
                                                    parentClassID = fourthConvey.classID,
                                                    parentActorNumber = 0,
                                                    parentComponent = 0,
                                                    waitForConfirmation = True)
fourthStand.set_material_properties(color = [0.3, 0.3, 0.3],
                                    roughness = 0.4,
                                    metallic = False)

# secondStand = QLabsBasicShape(qlabs)
# secondStand.spawn_id_and_parent_with_relative_transform(actorNumber = 99,
#                                                         location = [0.5, 0, -0.15],
#                                                         rotation = [0, 0, 0],
#                                                         scale = [0.95, 0.3, 0.3],
#                                                         configuration = 0,
#                                                         parentClassID = secondConvey.classID,
#                                                         parentActorNumber = 1,
#                                                         parentComponent = 0,
#                                                         waitForConfirmation = True)
# secondStand.set_material_properties(color = [0.3, 0.3, 0.3],
#                                     roughness = 0.4,
#                                     metallic = False)

sixthStand = QLabsBasicShape(qlabs)
sixthStand.spawn_id_and_parent_with_relative_transform(actorNumber = 7990,
                                                    location = [0.5, positionSecond, -0.15],
                                                    rotation = [0, 0, 0],
                                                    scale = [0.95, 0.3, 0.3],
                                                    configuration = 0,
                                                    parentClassID = sixthConvey.classID,
                                                    parentActorNumber = 3,
                                                    parentComponent = 0,
                                                    waitForConfirmation = True)
sixthStand.set_material_properties(color = [0.3, 0.3, 0.3],
                                    roughness = 0.4,
                                    metallic = False)


########## Create a widget tube ##########
deliveryTube = QLabsDeliveryTube(qlabs)
deliveryTube.spawn_id_degrees(actorNumber = 1,
                              location = [1.75, 0, 8],
                              rotation = [0, 180, 0],
                              scale = [1, 1, 1],
                              configuration = 1,
                              waitForConfirmation = True)
deliveryTube.set_height(height = 7)

# ALUMINIUM DELIVERY TUBE
deliveryTube2 = QLabsDeliveryTube(qlabs)
deliveryTube2.spawn_id_degrees(actorNumber = 71,
                              location = [1.75, positionSecond, 8],
                              rotation = [0, 180, 0],
                              scale = [1, 1, 1],
                              configuration = 1,
                              waitForConfirmation = True)
deliveryTube2.set_height(height = 7)
#endregion


time.sleep(1)

#region : create bins and shredders

########## Create two shredders ##########
shredder = QLabsShredder(qlabs)
shredder.spawn(location=[1.35, -.5, -0.2], scale=[1.7,1.7,2.7], configuration=shredder.GREEN)
shredder = QLabsShredder(qlabs)
shredder.spawn(location=[1.35, -.5+positionSecond, -0.2], scale=[1.7,1.7,2.7], configuration=shredder.BLUE)

########## Create a disposal bin ##########
conveyorBinRed = QLabsBasicShape(qlabs)
conveyorBinRed.spawn_id_box_walls_from_center_degrees(actorNumbers = [2, 3, 4, 5, 6],
                                                    centerLocation = [-0.28, 0, 0],
                                                    yaw = 0,
                                                    xSize = 0.3, ySize = 0.3, zHeight = 0.2,
                                                    wallThickness = 0.01,
                                                    floorThickness = 0.1,
                                                    wallColor = [0.5, 0, 0],
                                                    floorColor = [0.5, 0, 0],
                                                    waitForConfirmation = True)

GreenCover = QLabsBasicShape(qlabs)
GreenCover.spawn_id(50, [1.35, -0.5, .18], scale=[.3,.3,.04])
GreenCover.set_material_properties(color=[0,.5,0])
GreenCover.set_physics_properties(enableDynamics=False, dynamicFriction=0,
frictionCombineMode=GreenCover.COMBINE_MIN, restitution= .5, restitutionCombineMode=GreenCover.COMBINE_MAX)
#### ALUMINIUM
```

```
########## Create a disposal bin ##########
conveyorBinRed2 = QLabsBasicShape(qlabs)
conveyorBinRed2.spawn_id_box_walls_from_center_degrees(actorNumbers = [72, 73, 74, 75, 76],
                                                       centerLocation = [-0.28, positionSecond, 0],
                                                       yaw = 0,
                                                       xSize = 0.3, ySize = 0.3, zHeight = 0.2,
                                                       wallThickness = 0.01,
                                                       floorThickness = 0.1,
                                                       wallColor = [0.5, 0, 0],
                                                       floorColor = [0.5, 0, 0],
                                                       waitForConfirmation = True)

BlueCover = QLabsBasicShape(qlabs)
BlueCover.spawn_id(750, [1.35, -0.5+positionSecond, .18], scale=[.3,.3,.04])
BlueCover.set_material_properties(color=[0,0,.5])
BlueCover.set_physics_properties(enableDynamics=False, dynamicFriction=0,
frictionCombineMode=BlueCover.COMBINE_MIN, restitution= .5, restitutionCombineMode=BlueCover.COMBINE_MAX)


#endregion

#region : create arms and supports
# ########## Create an arm ##########
firstArm = QLabsQArm(qlabs)
firstArm.spawn_id_degrees(actorNumber = 10,
                          location = [1, -0.5, 0.3],
                          rotation = [0, 0, 0],
                          scale = [1, 1, 1],
                          configuration = 0,
                          waitForConfirmation = True)

# Create a simple support for the arm
firstArmStand = QLabsBasicShape(qlabs)
firstArmStand.spawn_id_and_parent_with_relative_transform(actorNumber = 100,
                                                          location = [0, 0, -0.15],
                                                          rotation = [0, 0, 0],
                                                          scale = [0.3, 0.3, 0.3],
                                                          configuration = 0,
                                                          parentClassID = firstArm.classID,
                                                          parentActorNumber = 10,
                                                          parentComponent = 0,
                                                          waitForConfirmation = True)
firstArmStand.set_material_properties(color = [0.3, 0.3, 0.3],
                                      roughness = 0.4, metallic = False)

#### ALUMINIUM
firstArm2 = QLabsQArm(qlabs)
firstArm2.spawn_id_degrees(actorNumber = 710,
                          location = [1, -0.5+positionSecond, 0.3],
                          rotation = [0, 0, 0],
                          scale = [1, 1, 1],
                          configuration = 0,
                          waitForConfirmation = True)

# Create a simple support for the arm
firstArmStand2 = QLabsBasicShape(qlabs)
firstArmStand2.spawn_id_and_parent_with_relative_transform(actorNumber = 7100,
                                                          location = [0, 0+positionSecond, -0.15],
                                                          rotation = [0, 0, 0],
                                                          scale = [0.3, 0.3, 0.3],
                                                          configuration = 0,
                                                          parentClassID = firstArm2.classID,
                                                          parentActorNumber = 10,
                                                          parentComponent = 0,
                                                          waitForConfirmation = True)
firstArmStand2.set_material_properties(color = [0.3, 0.3, 0.3],
                                      roughness = 0.4, metallic = False)

#endregion

#region : Create beam sensors
########## Create beam sensors ##########
beamSensorSpawn = QLabsGenericSensor(qlabs)
beamSensorSpawn.spawn_id_degrees(actorNumber = 101,
                          location=[1.65, .3, 0.45],
                          rotation=[0, 0, -90],
                          scale=[1, 1, 1],
                          configuration = 0,
                          waitForConfirmation = True)

beamSensorSpawn.show_sensor(showBeam=True,
                        showOriginIcon=True,
                        iconScale=0.1,
                        waitForConfirmation=True)
beamSensorSpawn.set_beam_size(startDistance=0,
                          endDistance=0.5,
                          heightOrRadius=0.01,
```

```
                                width=0.01,
                                waitForConfirmation=True)


beamSensorDrop = QLabsGenericSensor(qlabs)
beamSensorDrop.spawn_id_degrees(actorNumber = 105,
                                location=[1.8, 0, 0.43],
                                rotation=[0, 0, 180],
                                scale=[1, 1, 1],
                                configuration = 0,
                                waitForConfirmation = True)

beamSensorDrop.show_sensor(showBeam=False,
                           showOriginIcon=False,
                           iconScale=0.1,
                           waitForConfirmation=True)
beamSensorDrop.set_beam_size(startDistance=0,
                                endDistance=0.12,
                                heightOrRadius=0.01,
                                width=0.01,
                                waitForConfirmation=True)


beamSensorArm1 = QLabsGenericSensor(qlabs)
beamSensorArm1.spawn_id_degrees(actorNumber = 102,
                                location=[1, .3, 0.45],
                                rotation=[0, 0, -90],
                                scale=[1, 1, 1],
                                configuration = 0,
                                waitForConfirmation = True)

beamSensorArm1.show_sensor(showBeam=True,
                           showOriginIcon=True,
                           iconScale=0.1,
                           waitForConfirmation=True)
beamSensorArm1.set_beam_size(startDistance=0,
                                endDistance=0.5,
                                heightOrRadius=0.01,
                                width=0.01,
                                waitForConfirmation=True)

### ALUMINIUM
beamSensorSpawn2 = QLabsGenericSensor(qlabs)
beamSensorSpawn2.spawn_id_degrees(actorNumber = 7101,
                                location=[1.65, .3+positionSecond, 0.45],
                                rotation=[0, 0, -90],
                                scale=[1, 1, 1],
                                configuration = 0,
                                waitForConfirmation = True)

beamSensorSpawn2.show_sensor(showBeam=True,
                           showOriginIcon=True,
                           iconScale=0.1,
                           waitForConfirmation=True)
beamSensorSpawn2.set_beam_size(startDistance=0,
                                endDistance=0.5,
                                heightOrRadius=0.01,
                                width=0.01,
                                waitForConfirmation=True)


beamSensorDrop2 = QLabsGenericSensor(qlabs)
beamSensorDrop2.spawn_id_degrees(actorNumber = 7105,
                                location=[1.8, 0+positionSecond, 0.43],
                                rotation=[0, 0, 180],
                                scale=[1, 1, 1],
                                configuration = 0,
                                waitForConfirmation = True)

beamSensorDrop2.show_sensor(showBeam=False,
                           showOriginIcon=False,
                           iconScale=0.1,
                           waitForConfirmation=True)
beamSensorDrop2.set_beam_size(startDistance=0,
                                endDistance=0.12,
                                heightOrRadius=0.01,
                                width=0.01,
                                waitForConfirmation=True)


beamSensorArm2 = QLabsGenericSensor(qlabs)
beamSensorArm2.spawn_id_degrees(actorNumber = 7102,
                                location=[1, .3+positionSecond, 0.45],
                                rotation=[0, 0, -90],
                                scale=[1, 1, 1],
```

```
                                configuration = 0,
                                waitForConfirmation = True)

beamSensorArm2.show_sensor(showBeam=True,
                           showOriginIcon=True,
                           iconScale=0.1,
                           waitForConfirmation=True)
beamSensorArm2.set_beam_size(startDistance=0,
                                endDistance=0.5,
                                heightOrRadius=0.01,
                                width=0.01,
                                waitForConfirmation=True)

#endregion


# Spawn a cylinder
cylinder = QLabsWidget(qlabs)

# # Start spawn model
QLabsRealTime().start_real_time_model(QARMS+'/QArm_Spawn0', actorNumber=10,
additionalArguments='-uri_hil tcpip://localhost:18900 -uri_video tcpip://localhost:18901')
QLabsRealTime().start_real_time_model(QARMS+'/QArm_Spawn1', actorNumber=11,
additionalArguments='-uri_hil tcpip://localhost:18902 -uri_video tcpip://localhost:18903')

def createCylinder(cylinderNo):
        # value = ['green', 'blue', 'red']   ## uncomment these 6 lines and
comment out the 3 below to implement dual cell type simulation
        # color = [[0,1,0], [0,0,1], [1,0,0]]
        # if cylinderNo % 2 == 0:
        #     position = 1
        # else:
        #     position = 0
        failedCell = randrange(2000)
        position = 0
        if failedCell == 1:
            position = 1            ## change when using dual cell simulation
        value = ['green', 'red']
        color = [[0,1,0], [1,0,0]]

        cylinder.spawn(location = [1.75, 0, 1],
               rotation = [0, 0, 0],
               scale = [.05, .05, .05],
               configuration = cylinder.CYLINDER,
               color = color[position],
               measuredMass = 1,
               properties=value[position])
        cylinderNo += 1
        return cylinderNo

# ALUMINIUM CYLINDER
def createCylinder2(cylinderNo2):
        # value = ['green', 'blue', 'red']   ## uncomment these 6
lines and comment out the 3 below to implement dual cell type simulation
        # color = [[0,1,0], [0,0,1], [1,0,0]]
        # if cylinderNo % 2 == 0:
        #     position = 1
        # else:
        #     position = 0
        failedCell = randrange(4000)
        position = 0
        if failedCell == 1:
            position = 1            ## change when using dual cell simulation
        value = ['blue', 'red']
        color = [[0,0,1], [1,0,0]]

        cylinder.spawn(location = [1.75, 1, 1],
               rotation = [0, 0, 0],
               scale = [.05, .05, .05],
               configuration = cylinder.CYLINDER,
               color = color[position],
               measuredMass = 1,
               properties=value[position])
        cylinderNo2 += 1
        return cylinderNo2

def moveConveyors(speed, stopped, notMovingTime):
    if stopped == True and speed == 0:
        if elapsed_time(notMovingTime) > 60:
            speed = 0.1
    if stopped == False and speed  == 0:
        notMovingTime = time.time()
        stopped = True
    if speed != 0:
        notMovingTime = time.time()
        stopped = False

    firstConvey.set_speed(speed)
```

```
    # secondConvey.set_speed(speed)
    thirdConvey.set_speed(speed)
    return stopped, notMovingTime

# ALUMINIUM CONVEYER
def moveConveyors2(speed, stopped, notMovingTime):
    if stopped == True and speed == 0:
        if elapsed_time(notMovingTime) > 60:
            speed = 0.1
    if stopped == False and speed  == 0:
        notMovingTime = time.time()
        stopped = True
    if speed != 0:
        notMovingTime = time.time()
        stopped = False

    fourthConvey.set_speed(speed)        ## fouth conveyor
    # fifthConvey.set_speed(speed)         ## fifth conveyor
    sixthConvey.set_speed(speed)         ## sixth conveyor
    return stopped, notMovingTime

def elapsed_time(startTime):
        return time.time() - startTime

def emptyBins(coverObject, startTime, timeToMove, timeOpened):
    out = False
    _,location,_,_ = coverObject.get_world_transform()
    if elapsed_time(startTime) < timeToMove:
        y = elapsed_time(startTime) * -(.3/timeToMove)
        coverObject.set_transform(location = [location[0], -0.5+y,
location[2]], rotation=[0,0,0], scale=[.3,.3,.04], waitForConfirmation=False)
    elif elapsed_time(startTime)<(timeToMove + timeOpened):
        pass
    elif elapsed_time(startTime) < (timeToMove*2 + timeOpened):
        y = (elapsed_time(startTime)-(timeToMove + timeOpened)) * -(.3/timeToMove)
        coverObject.set_transform(location = [location[0], -0.8-y,
location[2]], rotation=[0,0,0], scale=[.3,.3,.04], waitForConfirmation=False)
    else:
        out = True

    return out

def recordResults(data, greenObjects):
    newRow = {'Elapsed Time (s)':int(elapsed_time(scriptStartTimeSec)), 'Green':greenObjects}
    data = data._append(newRow, ignore_index=True)
    pd.DataFrame.to_csv(data, ("C:\\Users\\" + os.getlogin() +
"\\Documents\\Quanser\\examples\\VirtualFactory\\Results_" + scriptStartTime + ".csv"), index=False)
    return data

def findObj():
    data = pd.DataFrame(columns = ['Elapsed Time (s)', 'Green'])
    update = True
    cylinderNo=1
    ## ALUMINIUM CYLINDER ADDED
    cylinderNo2=1
    ########## Test for object ##########
    ## Relevant code for second cell type has been commented out in
the following section
    qarm1_State = 6
    qarm2_State = 6
    greenBinLimit = 30
    blueBinLimit = 40
    greenObjects = 0
    blueObjects = 0
    emptyingGreen = 0
    emptyingBlue = 0

    startTime = time.time()

    notMovingTime = time.time()
    stopped = False
    notMovingTime2 = time.time()
    stopped2 = False

    # make sure port number matches ones from spawn model start
    qarm1 = createQarm(18900)
    pickAndPlace(qarm1, qarm1_State)

    qarm2 = createQarm(18902)
    pickAndPlace(qarm2, qarm2_State)

    stopped, notMovingTime = moveConveyors(0.1, stopped, notMovingTime)
    stopped2, notMovingTime2 = moveConveyors2(0.1, stopped2, notMovingTime2)

    cylinderNo = createCylinder(cylinderNo)
    cylinderNo2 = createCylinder2(cylinderNo2)

    startTimeSpawn = startTimeQarm1 = startTimeGreen = startTimeBlue = time.time()
```

```
    startTimeSpawn = startTimeQarm1 = startTimeQarm2 = startTimeGreen
 = startTimeBlue = time.time()
    while True:

        _, hitSpawn, _,_,_,_ = beamSensorSpawn.test_beam_hit_widget()
        _, hitDrop, _,_,_,_ = beamSensorDrop.test_beam_hit_widget()
        _, hitArm1, _,_,_, propertiesArm1 = beamSensorArm1.test_beam_hit_widget()

        # ALUMINIUM
        _, hitSpawn2, _,_,_,_ = beamSensorSpawn2.test_beam_hit_widget()
        _, hitDrop2, _,_,_,_ = beamSensorDrop2.test_beam_hit_widget()
        _, hitArm2, _,_,_, propertiesArm2 = beamSensorArm2.test_beam_hit_widget()


        # LITHIUM
        if update == True:
            data = recordResults(data, greenObjects)
            update = False

        if hitSpawn and not hitDrop:
            if elapsed_time(startTimeSpawn) > 1:
                cylinderNo = createCylinder(cylinderNo)
                startTimeSpawn = time.time()

        if hitArm1 and propertiesArm1 == 'green':
            stopped, notMovingTime = moveConveyors(0, stopped, notMovingTime)
            if emptyingGreen == 0 and qarm1_State == 6:
                qarm1_State = 0
                pickAndPlace(qarm1, qarm1_State)
                startTimeQarm1 = time.time()

        if qarm1_State < 6:
            if elapsed_time(startTimeQarm1) > 1.5:
                qarm1_State =  qarm1_State + 1
                pickAndPlace(qarm1, qarm1_State)
                startTimeQarm1 = startTimeQarm1 + 1.5

                if qarm1_State == 4:
                    stopped, notMovingTime = moveConveyors(0.1, stopped, notMovingTime)

                if qarm1_State == 5:
                    greenObjects = greenObjects + 1
                    update = True


        # ALUMINIUM
        if hitSpawn2 and not hitDrop2:
            if elapsed_time(startTimeSpawn) > 1:
                cylinderNo2 = createCylinder2(cylinderNo2)
                startTimeSpawn = time.time()

        if hitArm2 and propertiesArm2 == 'blue':
            stopped2, notMovingTime2 = moveConveyors2(0, stopped2, notMovingTime2)
            if emptyingBlue == 0 and qarm2_State == 6:
                qarm2_State = 0
                pickAndPlace(qarm2, qarm2_State)
                startTimeQarm2 = time.time()

        if qarm2_State < 6:
            if elapsed_time(startTimeQarm2) > 1.5:
                qarm2_State =  qarm2_State + 1
                pickAndPlace(qarm2, qarm2_State)
                startTimeQarm2 = startTimeQarm2 + 1.5

                if qarm2_State == 4:
                    stopped2, notMovingTime2 = moveConveyors2(0.1, stopped, notMovingTime)

                if qarm2_State == 5:
                    blueObjects = blueObjects + 1

        if greenObjects == greenBinLimit:
            emptyingGreen = 1
            greenObjects = 0
            startTimeGreen = time.time()

        if emptyingGreen == 1:
            emptied = emptyBins(coverObject=GreenCover,
startTime=startTimeGreen,timeToMove=1,timeOpened = 13)
            if emptied:
                emptyingGreen = 0
                greenObjects = 0

        if blueObjects == blueBinLimit:
            emptyingBlue = 1
            blueObjects = 0
            startTimeBlue = time.time()

        if emptyingBlue == 1:
            emptied = emptyBins(coverObject=BlueCover,
```

```
startTime=startTimeBlue,timeToMove=1,timeOpened = 13)
            if emptied:
                emptyingBlue = 0
                blueObjects = 0

findObj()
```

References

European Parliament. European parliament resolution on a global eu strategy for covid-19 vaccination, 2023. URL https://www.europarl.europa.eu/doceo/document/E-9-2023-000997_EN.html.

Peter Kraljic. Purchasing must become supply management. 1983. URL https://hbr.org/1983/09/purchasing-must-become-supply-management.

RMI. What are conflict minerals? https://www.responsiblemineralsinitiative.org/about/faq/general-questions/what-are-conflict-minerals/#:~:text=%E2%80%9CConflict%20minerals%2C%E2%80%9D%20as%20defined,%2Dtantalite%20and%20wolframite%2C%20respectively.

Rocha. Digital twin-based optimiser for self-organised production, 2021. URL https://doi.org/10.1016/j.mfglet.2021.07.007.

Shahzad. Digital twins in built environments: An investigation, 2022. URL https://www.mdpi.com/2075-5309/12/2/120.

Kala Vest. Lithium fluoride market analysis, 2023. URL https://www.linkedin.com/pulse/lithium-fluoride-market-analysis-size-share-growth-current-kala-vest-tp6yf/.