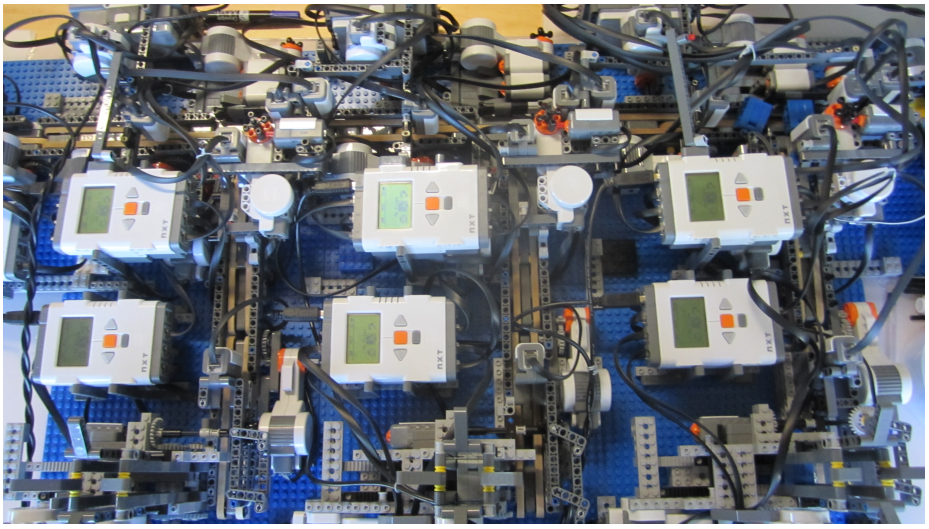
Improving the Control Capabilities of a Lego Model of a Production Line



Nikki Phoolchund (npslp2) & Tom Neat (ten23)

July - September 2012

Summary

Over the course of a ten week placement the control and teaching capabilities of a Lego model of a production line were upgraded.The project focussed on four main areas:

- Robustness Improvements and Intelligent Fault Detection
- Implementing Mainline Speed Control
- Implementing Mainline Buffering
- Improving the teaching capabilities of the unit

Both buffering and speed control capabilities for the mainline were added successfully, allowing two modes of operation: the existing mainline priority system and a new transfer line priority system using the new mainline buffering. New units in the form of an upstream and a new version of the splitter were added in order to allow the SIMIO model of the line to be replicated using the model. This allowed a more versatile approach to using the unit in control system teaching.Any experiment run in SIMIO can be better visualised using the real world model.Improving the robustness was a key goal, as the initial version of the system could not perform a sufficiently long run for an experiment to be completed. A variety of solutions were developed and integrated into the line to extend the running time. Finally a variety of user interfaces were developed to assist in teaching, removing the requirements for student users to understand Matlab commands in able to produce good quality results. These improvements would also aid in the upcoming fourth year projects being undertaken using the line.

Contents

1 Improving existing feedline buffering

The model and code provided at the start of the project included a strategy for buffering pallets along the conveyor belt from the feedline to the end of the transfer line. This strategy included 5 buffer positions. After multiple attempts to debug the existing code, it was decided that implementing a new strategy and writing a new code would be the most effective way of going about developing the feedline/transfer line buffering.

The new strategy developed allowed for buffer sizes from 0 to 4. For buffer sizes 0,1,2 and 3, 3 buffer positions have been defined, as illustrated in Figure 1. In the case of buffer size 0, it is expected that the pallet flows through the system without being held up at any of the buffer positions. In this configuration, the only position where the pallet may potentially be held up would be at the end of the transfer line, if a blockage is detected upstream on the mainline. If the pallet were to be held up at that point, the system would shutdown.

In the case of buffer sizes 1 and 2, the system would keep track of the number of pallets present in the stretch of belt from the feed unit to the end of the transfer line. As soon as the number of pallets present exceeded the buffer size, the system would shut down.

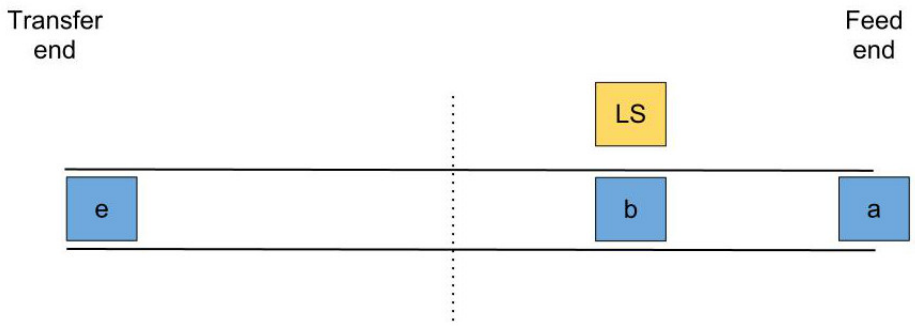


Figure 1: Buffer positions for buffer sizes 0,1,2 and 3

The shutdown scenarios for buffer sizes 3 and 4 are similar. In the last two buffer sizes, additional buffer positions are defined. When all buffer positions are filled and the feed unit attempts to feed in another pallet, the system shuts down and issues an error message informing the user that the buffer size has been exceeded. The flowcharts found in the Appendix provide an overview of the logic implemented to progress from one state to another as pallets enter and leave this section of the system.

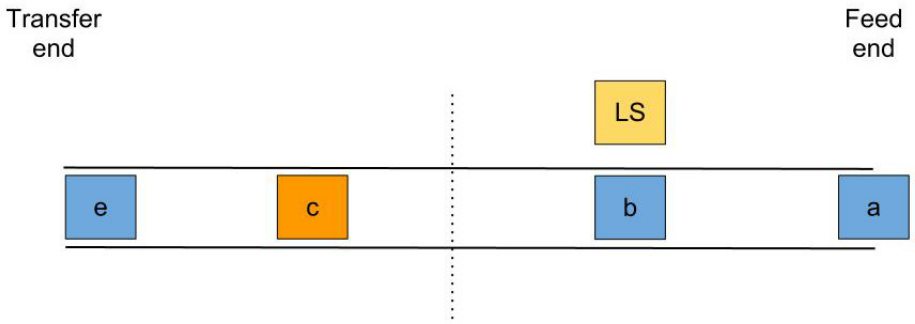


Figure 2: Buffer positions for buffer size 4

2 Speed control of the mainline to enable energy saving

2.1 Analysis of the problem

The existing system had no sensors on the mainline for tracking the presence of pallets on the mainline by the mainline NXT. It would therefore run continuously at constant speed as soon as the system was switched on. The mainline would be running even if there was no pallet present. This resulted in wasting energy and was

not useful for control purposes.

Aim: To design a speed control system for the mainline to allow its speed to be controlled and therefore minimise the amount of energy used.

Contraints:

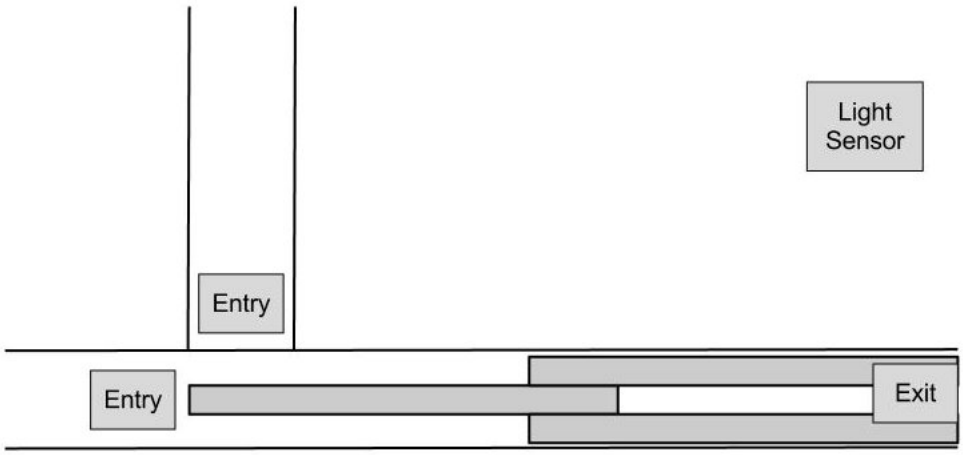
- No more than 4 sensors can be attached to each NXT on the mainline.
- The line layout must not exceed the base plate space if possible. Overhanging parts would make the model less portable.

Performance Indicators:

- System Throughput (to maximise)
- Motor time per run (to minimise)

2.2 Conceptual Design

Concept 1

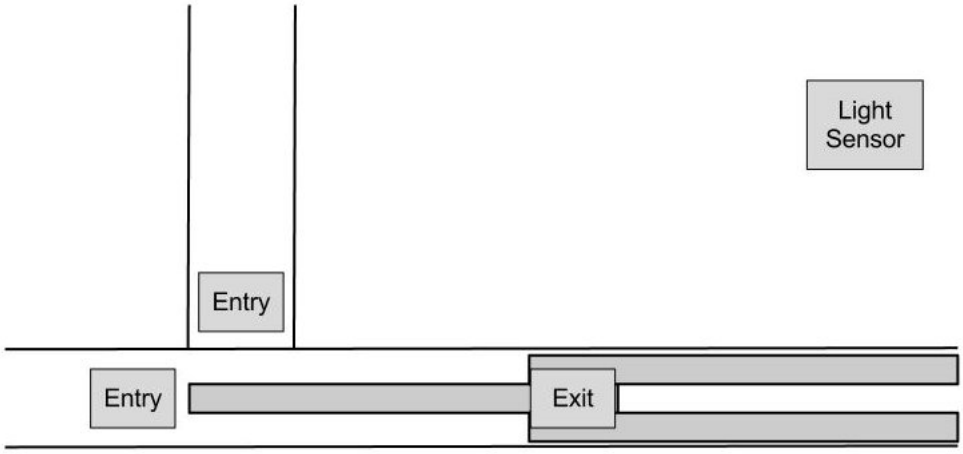


- Entry sensors will increase pallet counter by 1 when pallet enters.
- Single belt and double belt running both switched on when pallet present in this section of mainline
- Counter decreases by 1 when exit sensor detects pallet leaving section.
- Will require running belt for a certain amount of time after exit detection to ensure smooth handover.

Concept 2

Concept 2 will have the same entry sensor layout as concept 1 and no exit sensor. Instead of having both belts running from the time a pallet enters the system, the single belt will run for a fixed amount of time after entry is detected. Double belt will then be switched on and run for fixed period of time to carry pallet to exit.

Concept 3



- Entry sensors will increase pallet counter by 1 when pallet enters.
- Single belt runs as soon as entry is detected.
- Exit sensor decreases pallet counter. Single belt is switched off and double belt is switched on for a fixed time period.

Concept 4

Concept 4 involves varying the speed of the belts depending on the number of pallets present. The more pallets on the mainline, the faster the line would move. Having the belts moving slowly when there are less pallets allows more time of the transfer lines to feed in more pallets to fill in the gaps and hence increase pallet density. This would significantly save energy as the energy to run the line for a certain period of time is independent of the number of pallets on the line.

Concept 5

Concept 5 would involve shifting the mainline by one "pallet spacing" each time a pallet is fed in. This would maximise pallet density thereby moving the maximum number of pallets at the same time.

2.3 Concept Evaluation

Concept 1

✓ Can check if pallets are entering and exiting the section of mainline and hence can track whether a pallet got stuck.

✓ Can be used in buffering in conjunction with a set of states (similar to transfer line)

✓ Could check source of pallets to compare with simulation data for experiment

✓ No change in throughput

x Single and double belt control cannot be separated as the exact location of the pallet is not known when it is in the middle section.

x Requires a minimum of 3 sensors per section of mainline.

Concept 2

✓ Only requires two sensors

✓ No change in throughput

x Error detection by counting number of pallets at exiting is no longer possible.

x Belt control may generate more errors as it would require highly accurate timing to be effective.

Concept 3

- ✓ Single belt and double belt can be switched on and off separately therefore giving a more energy efficient system.
- ✓ Number of pallets in and out of section of mainline can be counted.
- ✓ Exact time of handover from single to double belt is known; ensuring smooth handover is possible.
- ✓ Entry sensor at next section of mainline will ensure smooth handover.
- ✓ No change in throughput
- x 3 sensors required per portion of mainline.

Concept 4

- Might change throughput of overall system depending on runtime.
- x 3 sensors required per portion of mainline.

Concept 5

This concept was unlikely to be successful; if the mainline were to accumulate a large number of pallets on the mainline higher upstream, the transfer lines downstream would be forced to buffer and might exceed their buffer limit.

- Changes in throughput of overall system.

2.4 Concept evaluation experiment

Aim: To determine which of the concepts provide the best throughput using minimum energy.

Method: A standard config file was used for the different concepts. All lines had a uniform feed rate (T=20) and buffer state of 4. The system was run for 180sec and the throughput and motor time were recorded.

Results

Concept	Throughput/pallets			Motor time/sec			Total		Motor time per pallet
	F1	F2	F3	M1	M2	M3	Throughput	Motor time	
0	3	2	2	360	360	360	7	1080	154.3
1	4	3	2	294	362	227	9	883	98.1
2	4	3	2	239	360	218.6	9	817	90.8
3	4	3	2	260	113	45	9	418	46.4

Conclusion: Concept 3 is by far the best in terms of amount of energy per pallet exiting the system.

2.5 Mechanical implications

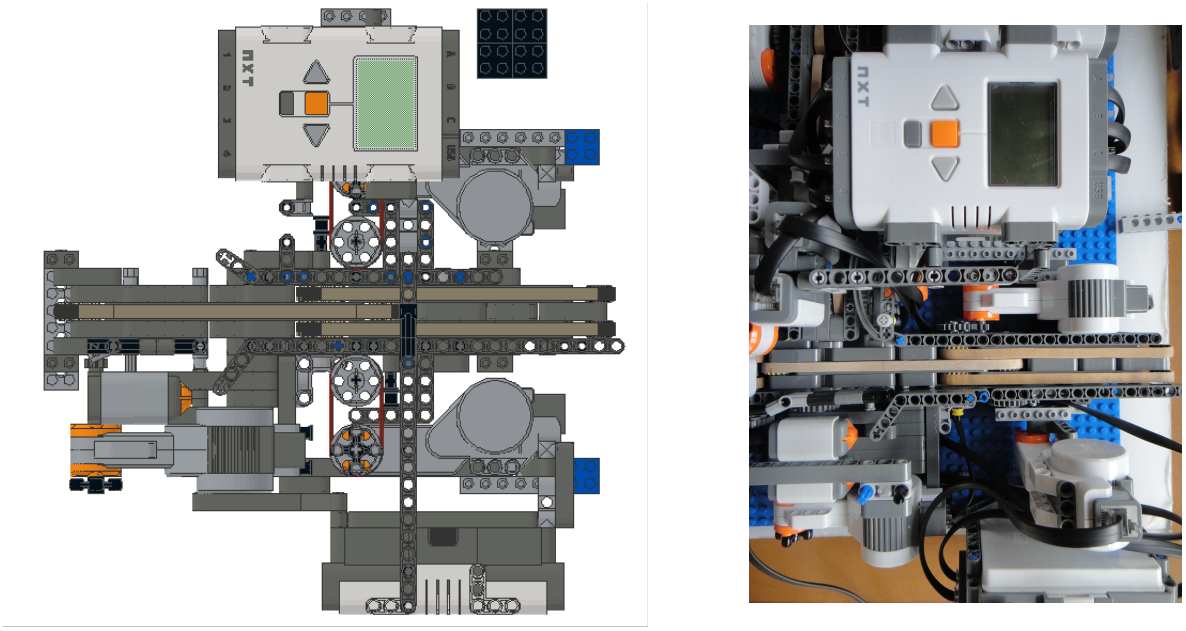
Touch sensor placement

The addition of the touch sensor to the existing row of touch sensors for the purpose of transfer detection by the mainline was found to be problematic initially. Various positions had to be considered for the touch sensor; the use of parallel touch sensors with the same mechanism raising the hammers for both sensors was used.

Motor drive mechanism

The model originally had the motors on the mainline connected via a wormgear mechanism. Due to the gear ratio used, even when the motors were running at maximum power, the mainline would be moving very slowly. With the transfer and feed lines moving at a higher speed than the mainline, the downstream feedlines might find themselves being held up by the pallets coming from upstream.

The gear mechanism was therefore and direct drive of the belt from the motors was implemented. Figure 3 shows the two different drive mechanisms.



(a) Gear drive

(b) Direct drive

Figure 3: Mainline motor drive mechanisms

3 Mainline buffering of pallets to improve productivity and/or enable particular orders to be intelligently fast-tracked through the system

3.1 Analysis of the problem

The mainline currently runs all pallets down to the end of the line without stopping. This implies that by default, the system gives priority to the mainline over the feed lines. However, it is desirable to have more flexibility and be able to buffer pallets on the mainline. This would be particularly useful when one of the feedlines is building up a buffer and needs to be cleared. This effectively is introducing the possibility of feedline priority over mainline.

Ideally, the improved system should be able to buffer a variable number of pallets. Implementation of this system should not require the addition of more than one sensor per section of mainline.

Capabilities of the Existing system:

- Two independently controllable sections of belt
- Current system can track number of pallets on each belt

Additional capabilities required:

- System must be able to track position of pallets on each belt
- Since there are two sections of belt, it is necessary to know if a pallet can be moved from one belt to another.

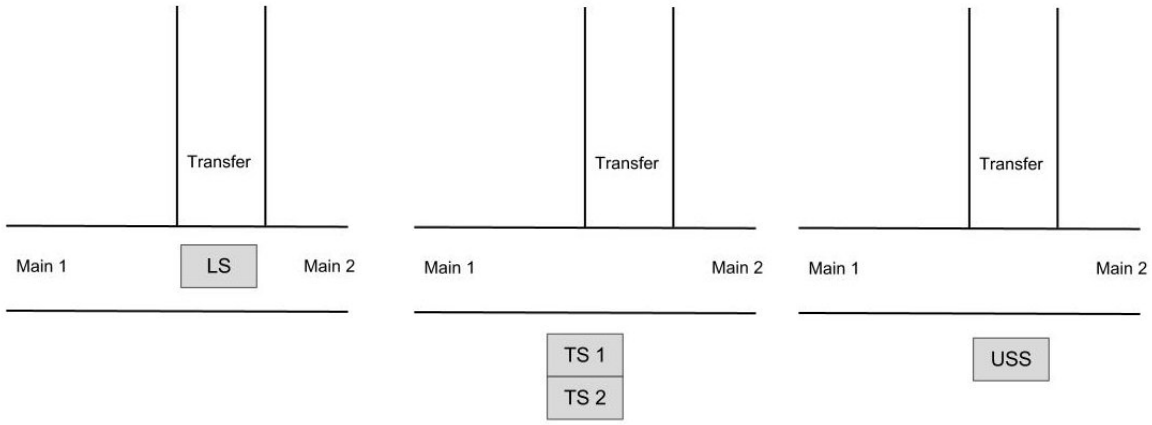
3.2 Determining transfer status

Mainline buffering requires knowledge of the status at the start of the next mainline unit. Pallets in one section of mainline need to be buffered only if the downstream section is transferring a pallet from the transfer unit onto the mainline.

Constraints: Ideally, only one sensor should be used to provide this information reliably.

Conceptual Design

Concept 1: Light sensor
This solution would require the addition of one light sensor directly above the transfer platform to determine if there is a pallet present or not.
Concept 2: Touch sensor
The transfer unit already makes use of a touch sensor triggered by the rising transfer platform to track when a pallet is being transferred. This concept involves the addition of a touch sensor next to the existing one with an identical trigger.
Concept 3: Ultrasound sensor
An ultrasound sensor pointing up the transfer line could potentially track the distance from the mainline to the approaching pallet to determine whether the pallet is about to be transferred.



(a) Concept 1: Light Sensor (b) Concept 2: Touch Sensor (c) Concept 3: Ultrasound Sensor

Concept Evaluation

All three concepts were tried out and it was found that the ultrasound sensor did not perform as required while the touch sensor and light sensor both showed the basic ability to detect the presence of the platform. These two concepts were then further investigated. Experiments run and conclusions are summarised below.

Concept 3: Ultrasound sensor
The aim was to get the sensor to read the distance from the pallet to the end of the transfer line as the pallet travelled from the feed unit to the end of the line.

Observations

- The cross sectional area of the pallet was found to be too small to be detected.
- The line is made up of several structures included gantries, sensors and other parts which would interfere with the ultrasound pulses. Several echoes would be detected at the sensor making it difficult to pick out the single echo corresponding to the pallet.

Conclusion: The ultrasound sensor concept was not feasible.

Concept 1: Light sensor
The light sensor could easily detect the presence/absence of a pallet at transfer station. In comparison to the

touch sensor, the position of the light sensor is more flexible. The next step was then to investigate whether the light sensor could distinguish between the pallet and the rising transfer platform.

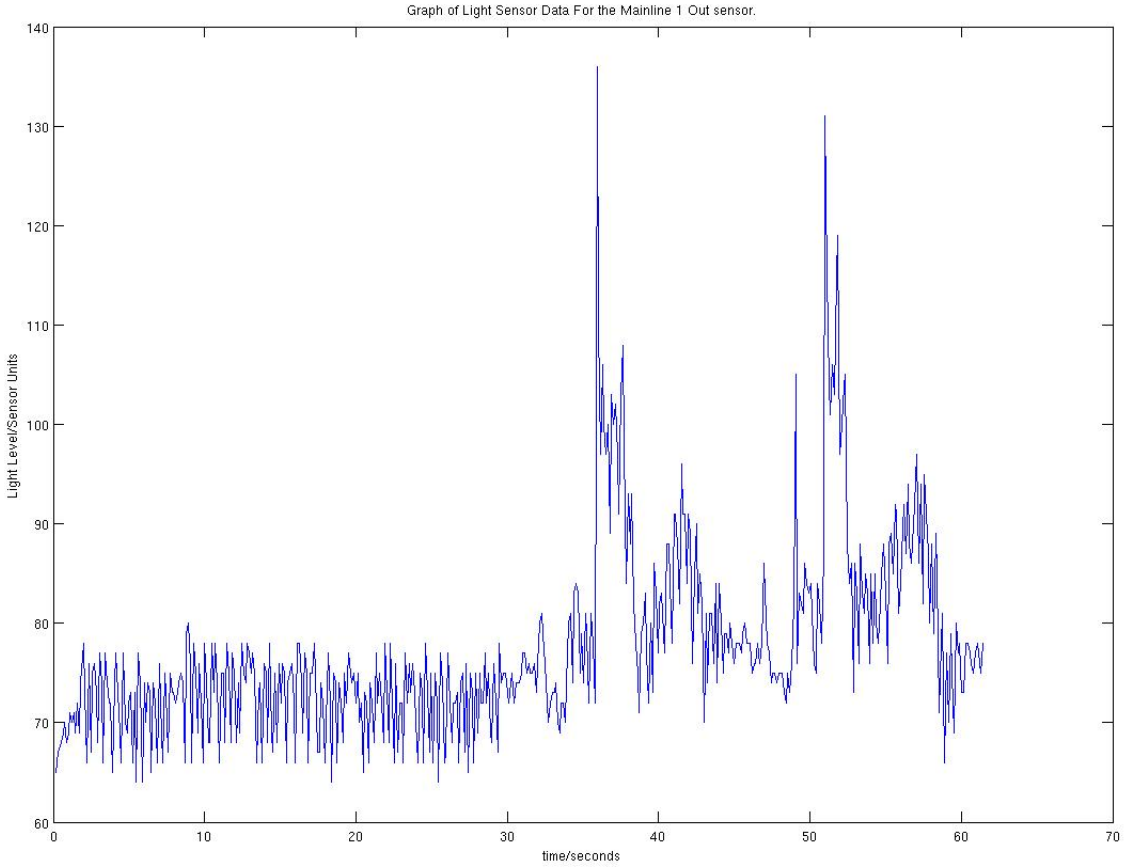


Figure 4: Light sensor data

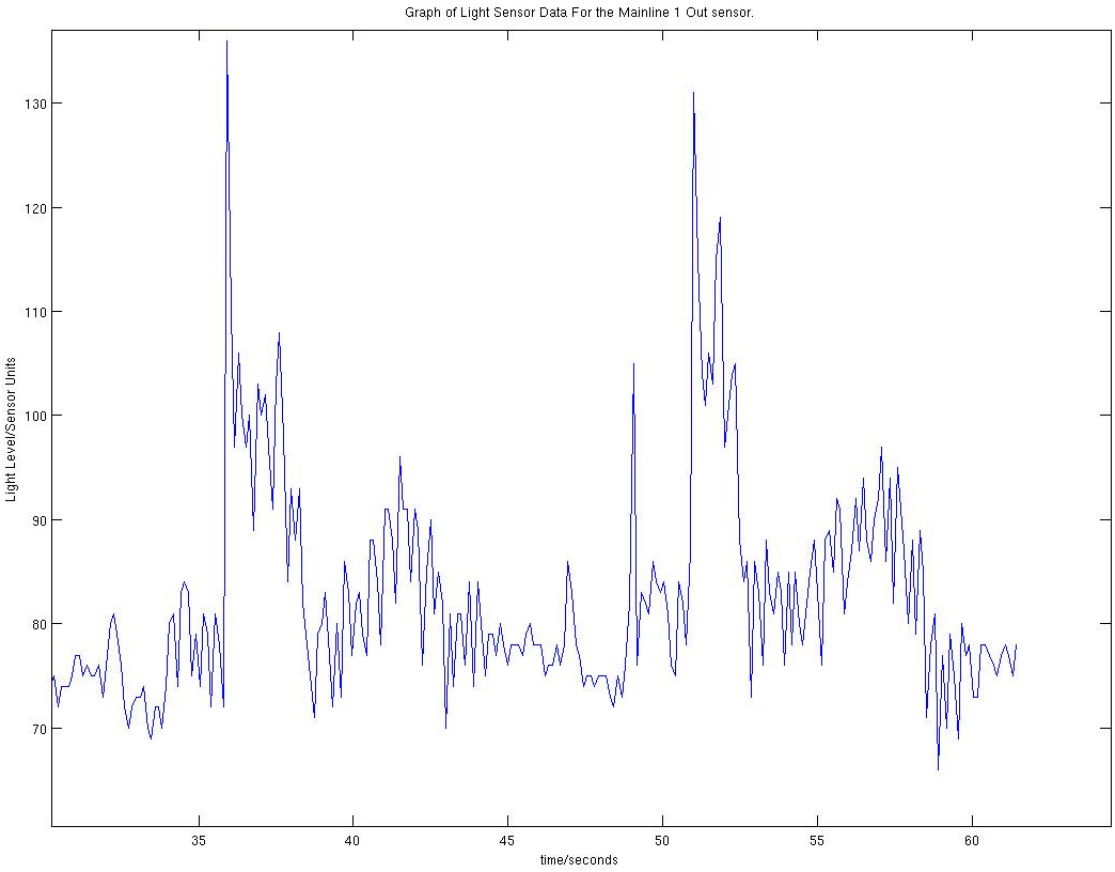


Figure 5: Zoomed-in light sensor data

Investigating the sensitivity of the transfer platform light sensor
*Aim:*To determine whether the light sensor placed over the platform can distinguish

- platform up v/s platform down
- pallet v/s no pallet
- platform up v/s pallet present

Observations

- It was not able to distinguish between platform up and platform down
- It was very reliable whent it came to differentiating between the presence and absence of a pallet
- It could detect when the pallet was up or down only if there was a pallet on it

Conclusion: While both the light and touch sensors could detect the transfer status, the light sensor could provide more information about the actual location of the pallet. The light sensor was therefore chosen as it gave more functionality.

3.2.1 Further concept evaluation

The position of the light sensor could significantly influence the reliability of the system. The next step was therefore to investigate the various placements possible for the light sensor.

Concept 1: Above transfer platform
Investigated earlier.

Concept 2: Shining up the transfer line

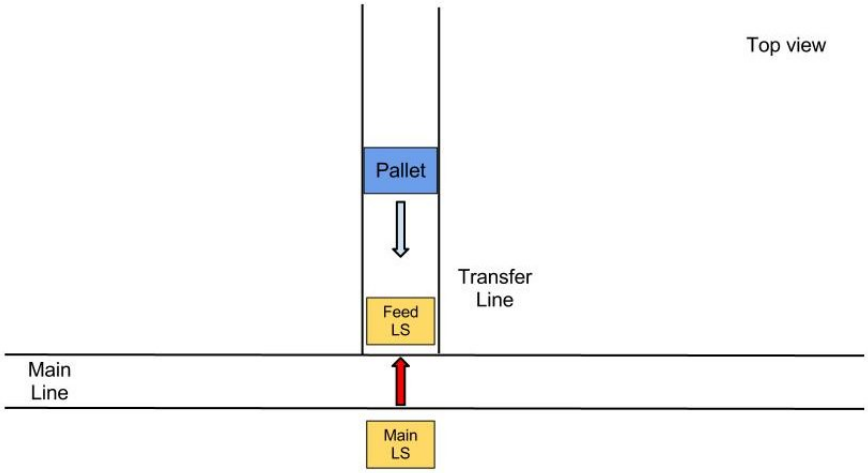


Figure 6: Concept 2: Shining up the transfer line

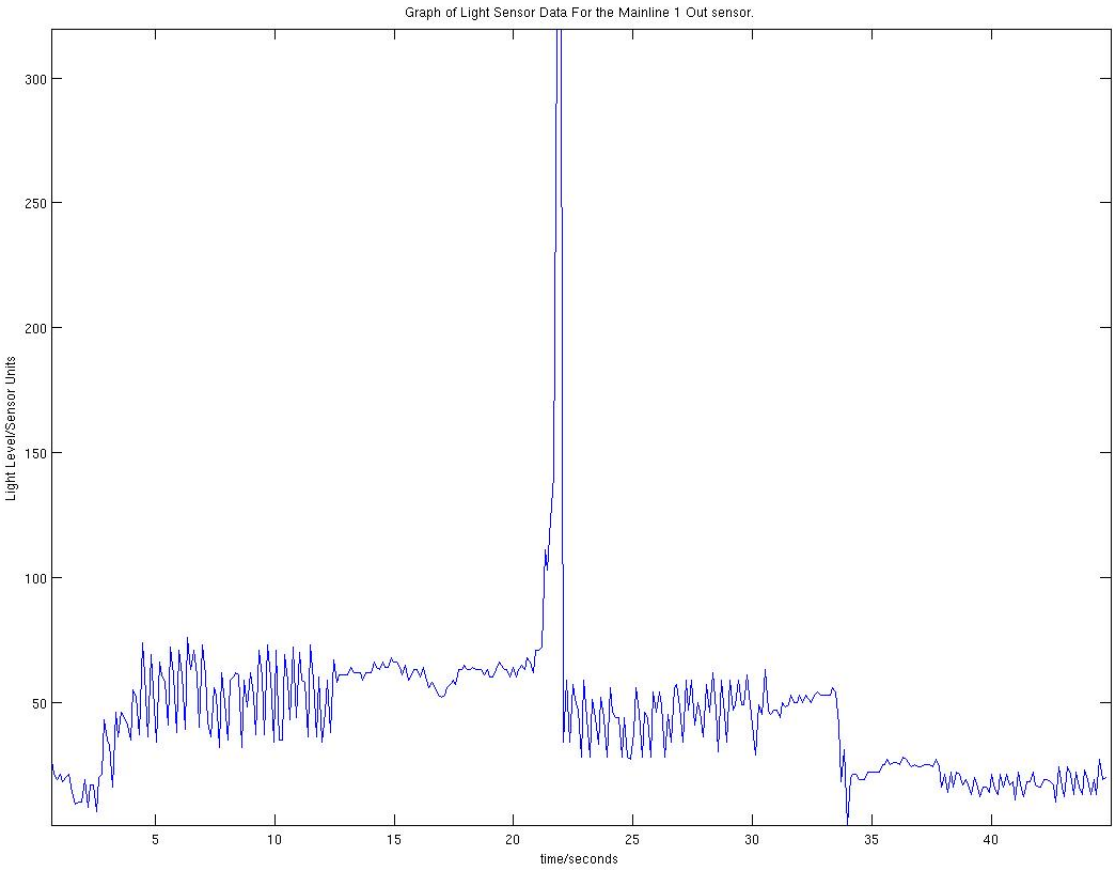


Figure 7: Light Sensor Output

The pallet was poorly detected. The spike seen in the plot was due to the light gate being blocked by the rising platform barrier during the transfer process. This setup effectively did not give any more information than using the touch sensor.

Concept 3: Main sensor overhanging feed sensor

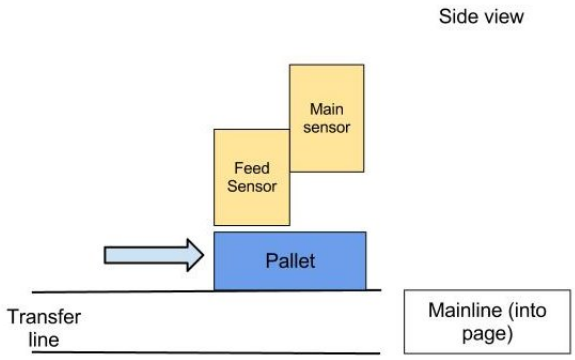


Figure 8: Concept 3: Main sensor overhanging feed sensor

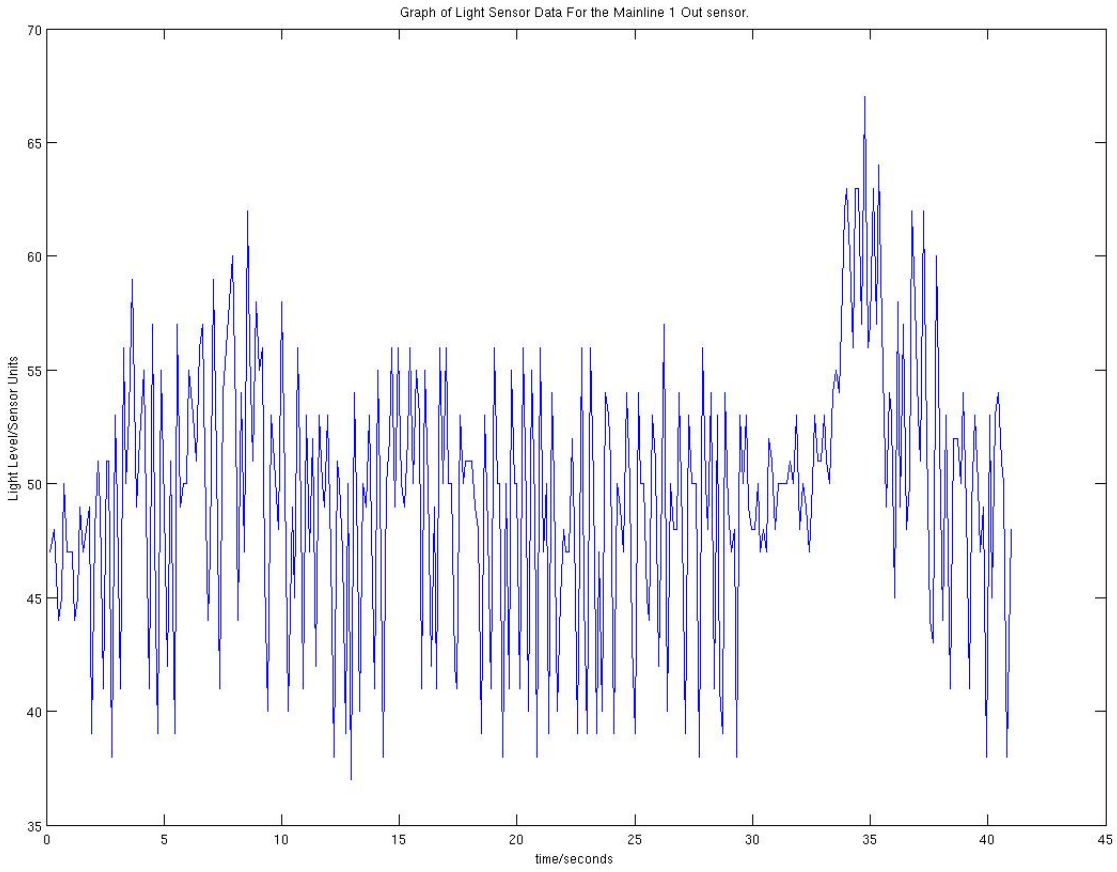


Figure 9: Light sensor data

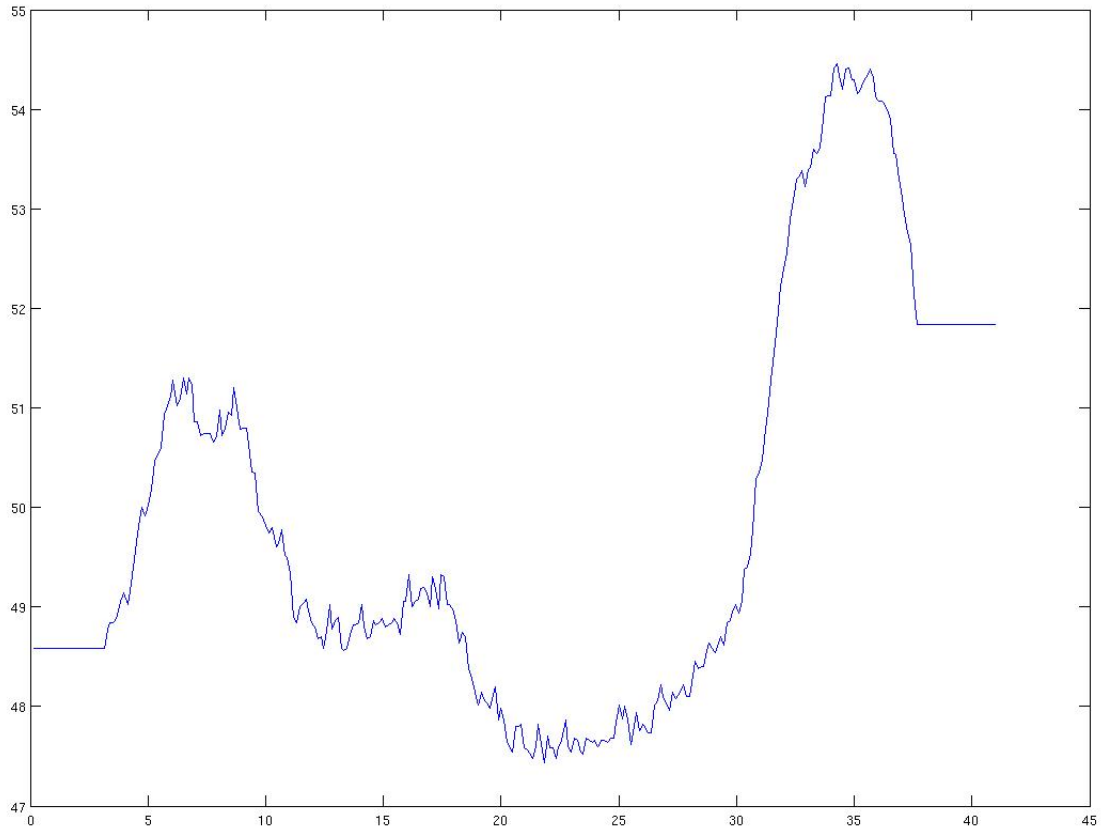


Figure 10: Zoomed-in light sensor data

The results from this concept were very noisy but there was still a noticeable rise when the pallet arrived. An attempt to smooth the data made the pallet detection more obvious. However, it was not known how large a smoothing window could to used or where the threshold should be set.

Concept 4: Main sensor next to feed sensor

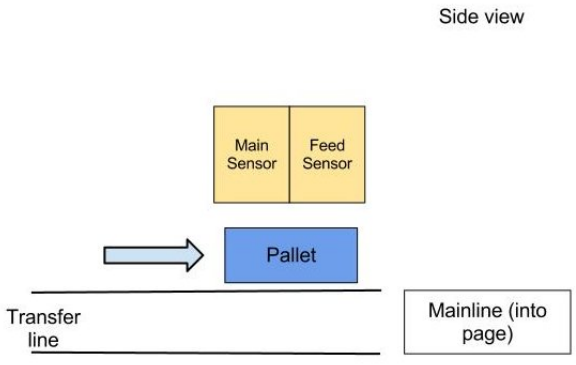


Figure 11: Concept 4: Back to back sensors

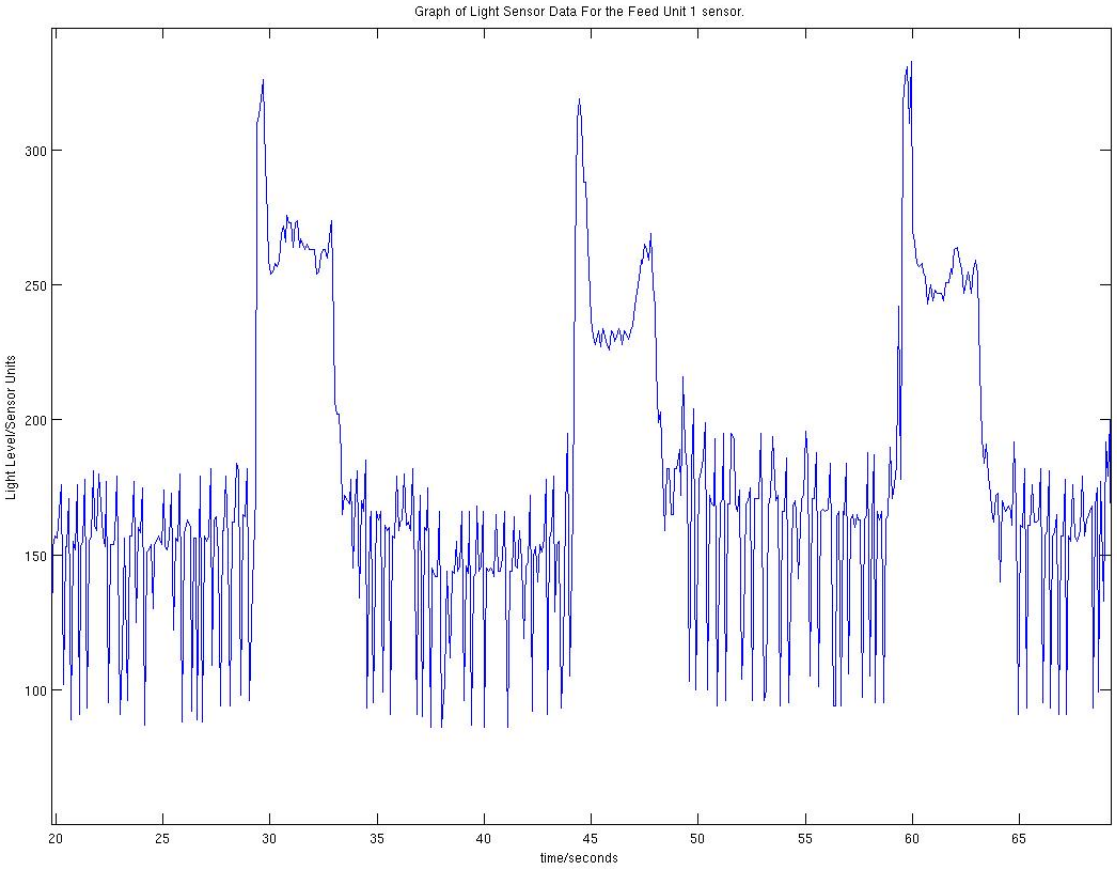


Figure 12: Feed Light sensor data - Concept 4

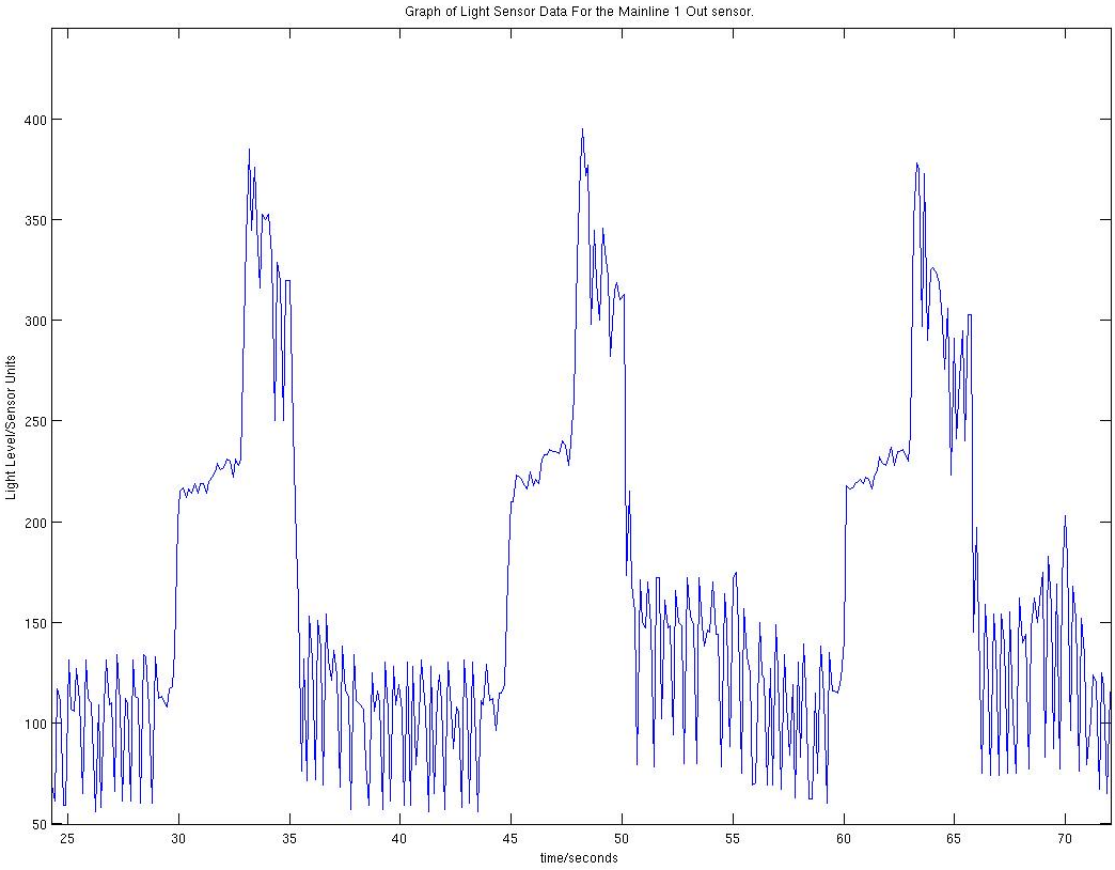


Figure 13: Main Light sensor data - Concept 4

In the case of concept 4, both the feed and main sensors successfully detected the pallet. The main light sensor exhibited readings similar to the one from the feed sensor which was known to be a reliable way of detecting the presence of a pallet.

Conclusions

Given the limited space available, the light sensors were placed as close together as possible. The feed sensor would pick up the front edge while the mainline sensor detected the back edge of the pallet. Therefore, both sensors gave reliable detection of the pallet.

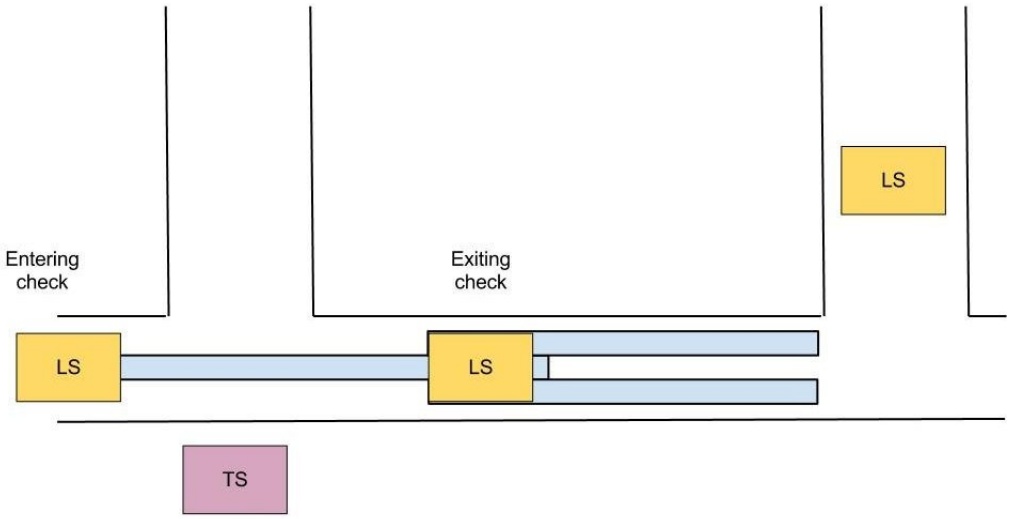


Figure 14: Final light sensor layout for each section of mainline

3.3 Mainline Buffering Strategies

Each section of the mainline includes a transfer platform. The significant amount of space taken up by the platform led to having little space available for buffering. Therefore, only 3 buffer positions could be implemented with an additional pallet on the transfer platform. The
The figures below show the differet buffer positions used for the various buffer sizes. The flowcharts in Appendix ?? show the state-by-state progression of the pallets through the system for each buffer state.

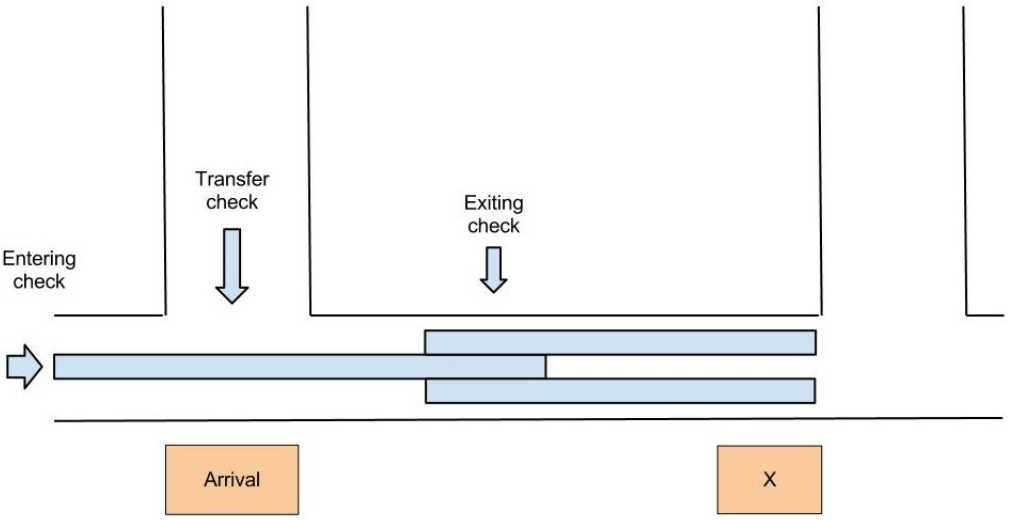


Figure 15: Positions for buffer size = 1

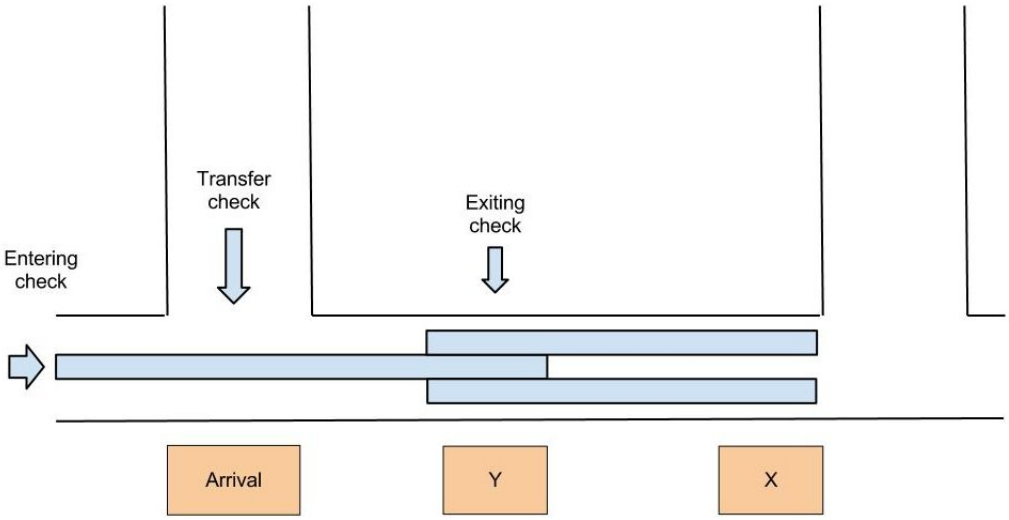


Figure 16: Positions for buffer size = 2

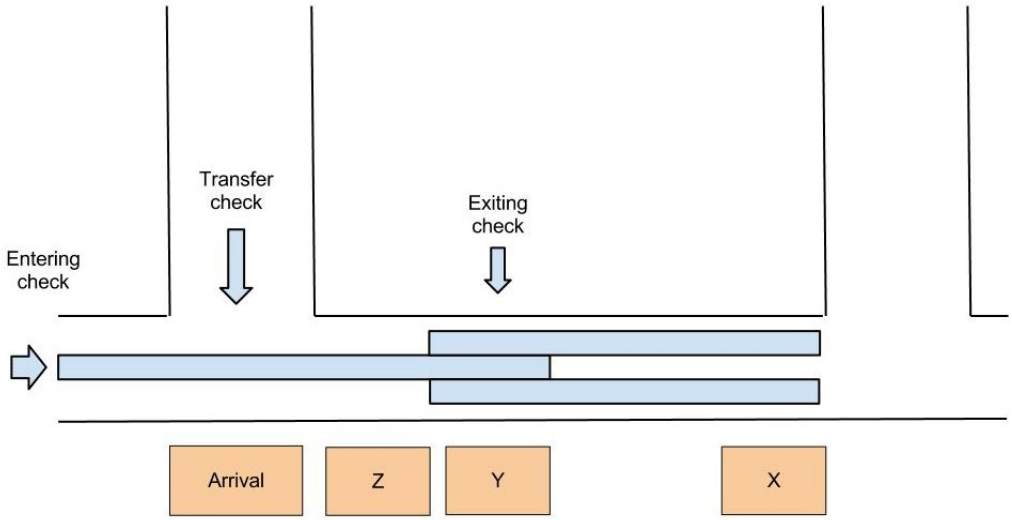


Figure 17: Positions for buffer size = 3

4 Correcting and Implementing the Splitter Unit

4.1 Overview of Problems

Although the original project by Konrad Newton had developed a splitter unit for the model, it was incomplete, and had a number of failings that prevented it from being effectively deployed as part of the model. These errors had to be addressed and corrected before the unit was suitable for any practical modelling purposes. The majority of the corrections concerned the operating algorithm and later the mechanics of the system.

Once the unit was working satisfactorily a number of changes and additions were made to the unit. These added the ability to perform colour splitting of the pallets and a form of quality control whereby pallets with mismatching code and colour would be separated. Finally the structure of the units conveyor belt was adapted such that it could be placed at any point on the mainline. This change would allow future developments whereby parallel mainlines may be employed using the splitter to divide pallets between the two mainlines.

4.2 Mechanical Corrections

Firstly the main separating arm design was not standard and both existing designs had their disadvantages. A new arm design was created incorporating the best features of both existing designs and the arm design was

standardised. This also resolved the issue of the arm head catching on the elastic bands of the conveyor and displacing them; this avoiding system damage.

In response to this change the siding was changed and new guide rails were added to reduce interference with the pushing arm head. These also gave better alignment of the pallets on the side siding and prevented the pallets not being pushed far enough or protruding into the mainline and causing jams.

To prevent the pallets catching on the cables which connected the sensors and motors to the NXT a series of cable runs and ties were deployed to restrain the cables. This resolved the error of cables interfering with mainline operations.

Finally the new fail safe initialisation procedure required a mean of establishing the position of the pusher head. A touch sensor was incorporated into the design at the top of the pusher arm to signal when the arm was fully retracted and not protruding onto the mainline, more reliably than a fixed angle rotation.

4.3 Correcting the Code Reading System

As received the splitter was unable to accurately recognise the codes on the side of the pallets, and thus would behave in a very erratic manner. This was unsuitable for reliable modelling units. This was addressed by a two stage process:

4.3.1 Changing The Pallet Code System

The first change made was to redesign and standardise the pallet.Previously the pallets were all of unique designs and the code was implemented by forcing balls of screwed up paper into the holes in the pallet. This made the system hard to read by the user and unreliable to the software. Balls of paper were not of consistent size or density and could not be guaranteed to fully block the hole; hence the light passing through each blockage was different.

The Lego system offered a better alternative, the pallets were redesigned to use open Lego connectors in places where a hole is required and a closed connector where a blockage is required. This had the advantage of the components being different colours; this did not affect code reading by the computer but made it much easier to read the code by eye. Finally the ease of changing the code on a given pallet was increased as the Lego could be separated easily by hand and reconfigured.

This also had the advantage of allowing the type of code to be specified to being a 4 bit code- three data bits and an initial parity bit. The code was read in standard left to right fashion with convention that a 1 was represented by a blocked hole and a 0 by an open hole. This made the logic of reading the code from the pallet much simpler when it came to realising the concepts.

4.3.2 Updating the Pallet Reading Code

Initially the system employed to read the code was to gather data along a pallet, differentiate the data ,and search for regions of high gradient within certain time windows. When graphed manually the results were inconclusive. A preliminary investigation revealed that the code was difficult to work with or read and so a new system was developed from scratch to avoid inheriting any of the fault of the old code.

To read the pallet it was noted that, due to the constant running speed of the belt, the time after the leading edge of a pallet of each of the code bits was well defined within a range of 0.2 sec. Hence all systems would start reading a snapshot of light gate data on the leading edge of a pallet for 3 seconds; the time it took for pallet transit.

The first concept was to smooth the dataset, differentiate and set a gradient threshold for a one or zero bit. If the gradient was below a threshold within the time intervals of the bit than it was declared a 1 bit, and if a gradient 'spike' was observed a zero bit would be declared. This system was tested with a pair of pallets of differing colours: the results are shown below.

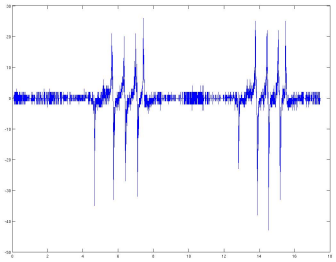


Figure 18: Use of differential data for code detection

Here it is shown that in some cases the static detection threshold is ill suited to the problem. A dynamic threshold was also tested but similarly did not function.

Once new, dynamic light sensor code was developed to suit the mainline sensors, a new option was explored. A more simple approach was taken to use the readings directly; previously the readings were too sensitive to noise and ambient lighting levels to set a one/zero threshold on the data directly. This option was explored, using a dynamic threshold as set by the new code. The graphical results below show successful operation and bit identification.

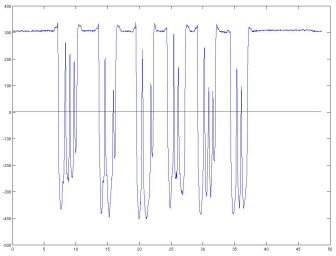


Figure 19: Direct use of data with threshold for detection

This shows that the unit operates successfully in code detection mode. The identification of codes was then simple to enact by performing a series of logic operations on the results to determine if the codes matched, and if so to add the pallet to a queue of pallets which must be separated. Thus the unit was deemed to be successful in meetings it original specification and being far more robust to ambient environment changes.

4.4 Implementing a Colour Detection System

In the process of investigating the code detection system it was noted that a colour detection system may be easier to implement and more reliable than the code system. It was decided to investigate the deployment of such a system.

The first concept to be explored was to use the existing light gate for performing colour readings. The results were poor, with all coloured pallets appearing identical to the graph. This was thought to be due to the small coloured area presented to the light gate due to the view angle. It was hence decided to suspend a light sensor above the pallet and reattempt to determine light levels corresponding to different coloured pallets. This also failed as there were only three distinct signals from five colours of pallet.

An alternative concept was then investigated. A Hi-Technic colour sensor was purchased, which offered the ability to read either r-g-b colour data or to use an inbuilt index to determine the colour. The sensor was tested on the three pallet types and the index was noted to be sufficient to make the broad distinction of red,blue,yellow, grey in all light levels, however the light and dark grey pallets were indistinguishable by this method. Consulting the RBG data retrieved revealed that the rgb code for light grey was far higher value than the dark grey in all light levels. It was also noted that the dark grey levels could be established from a reading when no pallet was present as the structure is predominantly dark grey. Thus a means of differentiating grey pallets into the subcategories was obtained.

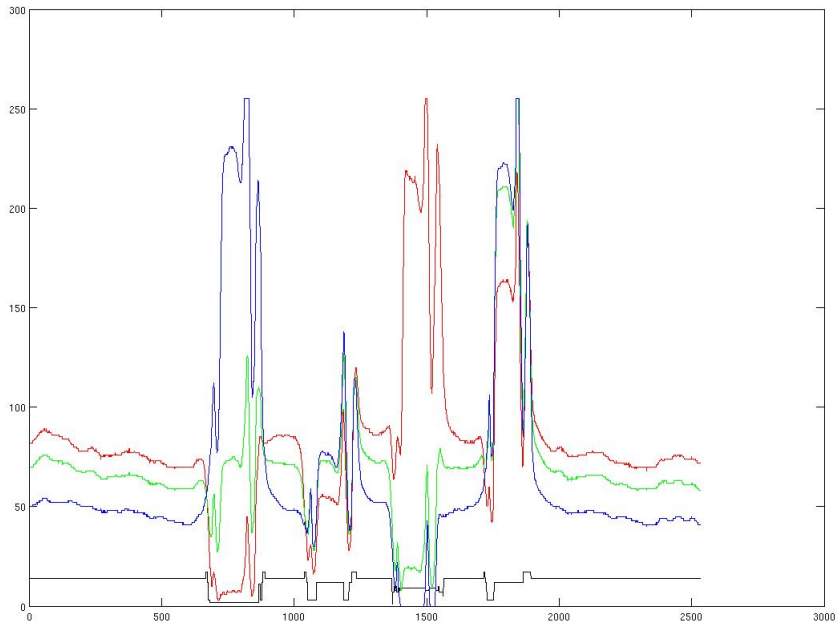


Figure 20: RGB and index values for different colour pallets

Having obtained a reliable method of determining colour a script to read the sensor data just after the pallets leading edge was designed - this prevented the pallet from being mislabelled by reading a section which is part pallet and part line and improved the working reliability of the unit.

Finally a colour column was added to the existing data array generated, and a new case script was written to separate the pallets based on colour. The script was adapted from the code script of:
If Colour = separated colour
add to push list
else
just log pallets arrival
end
if push list has new pallet
push

This script was deployed and tested. It was shown to work reliably and hence was deemed successful as an improvement. It was shown to be more reliable than the code reading script by a small margin This reliability was demonstrated over a range of operating conditions and lighting levels, over a number of weeks.

4.5 Quality Control Mode

Based on the above work in obtaining colour and code recognition capabilities integrated into the splitter, it was noted that a process of quality control could be modelled by the unit. This could be captured by comparing a pallets colour and code to a reference table provided by the user: if the pallet id not match a table entry it was declared defective and separated. This was very simply implemented by taking the existing scripts if conditions and combining them into a decision tree. This tree decided first the colour of the pallet, and then for each colour a code match section was employed. If these checks showed the pallet to be different from specification then it was added to the ejection list.

The system was implemented and shown to function. The configuration file was updated to allow each splitter unit to have a quality control table or a colour or a code to operate on, and a user defined switch added to allow this choice to be made.

4.6 Improving the Modularity of the Unit Mechanically

The layout was rearranged to allow the unit to fit successfully with the mainlines in any configuration. This was achieved in two ways:

- The conveyor system was reconstructed to be consistent with the mainline conveyor system. The single belt and ramp approach was replaced with a single belt and dual belt system, of the same design as the mainline system, allowing the splitter to be placed anywhere along the mainline without interference.
- The component were rearranged such that there was no overhang from the board, allowing more compact storage and the unit to be placed in more advance configurations in future if this design concept is adhered to.

Hence the flexibility of the unit is improved, and the unit is a contained module which could be used alone or as part of a larger more modular set-up which may be developed in future.

5 Implementing an Upstream Feed Unit

In the SIMIO model of the production line there exists and option to have pallets arriving at the end of the mainline to simulate arrivals from lines further upstream. This can be useful in allowing large systems to be simplified; if there is a single section of line in which the problem exists the model can be simplified to using an upstream unit and then the current kit to replicate the area of interest for visualisation and study. This would increase the versatility of the kit, as it would be rare to have to model all 10 lines in a factory, but attention could be focussed on a single case e.e. where one line is producing units erratically as a result of mechanical error further upstream.

It was noted that the process of feeding from the upstream would be similar to any other feed unit; hence the modular nature of the lien could be exploited. The current feed unit was studied; its was noted that in its current form the unit could not simply be placed upstream and run normally. However the hardware involved was all approximately correct and established, only the layout on the base needed changing to interface with a mainline rather than with a transfer unit. Similarly the current feed buffering system would be unnecessary as pallets arriving from upstream will undergo all buffering elsewhere; the arrival is all that is important.

Thus it was decided to adapt the unit to suit the new purpose.

5.1 Mechanical Implications

The first adaptation that was made was to construct a new feed unit tower structure and feed mechanism from scratch to experiment on.It was found that the tower and base feed section required no additional modification to function correctly.

Next a conveyor section was constructed. It was noted that the interface between main line and feed unit would need to be different than the existing interface (that between feed and transfer units). This was resolved as the splitter unit had a similar structure and interface, and so the conveyor design was borrowed from the splitter to allow successful interface with the upstream end of a mainline unit. Hence all of the components of the new feed unit were now constructed.

Finally the layout on the board was designed from scratch.The new components were assembled into a complete feed unit, and the board was positioned such that it could successfully align with the mainline units. The design was such that the unit was fully modular, and could be removed and positioned adjacent to the splitter unit at the other end of line for a more compact storage arrangement.

5.2 Software Implications

Having designed a new structure for the upstream feed unit the existing code for a standard feed unit was examined, with a view to adapting it to suit the new unit. It was noted that all of the existing code for initialisation would be compatible with the new unit type and so this process was copied to the new process. The existing start/stop mechanism was maintained and implemented in the unit, and the safe shut down process was appended to the end of the flowchart.

The next stage was to design a process flow for operations. The existing feed units operated on a buffer state system, as defined by the user, which had some elements in similarity with the transfer unit and included

predictions of the transfer units behaviour. This was not suitable for the upstream unit. It was noted that there were two cases of operation:

- Mainline priority: the unit must always feed on time regardless of the processes going on at the proceeding transfer unit
- Transfer Line Priority: the unit must hold the pallets if the transfer line is attempting to transfer a pallet at the time.

Otherwise it was noted that the operations must be the same with regards to checking time, and then following the existing feed process. Hence a two stage buffer process was designed,the first stage shows the pallet has arrived from the tower onto the line. It is then moved to a hold position at the end of the line. If the priority is for the mainline it is then transferred directly to the mainline. Appropriate decisions are made as to the state of a downstream blockage if the transfer line has priority.In which case the pallet may be held at the buffer state until the downstream line has cleared.

The feed process was copied directly from the existing feed process, with a minor adaptation to incorporate the new state system. The code was now fully assembled to incorporate all processes. Debugging was carried out and then the code was deployed to control the fully assembled unit. Further testing and debugging was then carried out until operations were satisfactory. .

5.3 Review

- The Unit was found to operate successfully as a standalone system, demonstrating local control was deployed.
- The unit was successfully integrated into the main system as an upstream feed unit
- The unit was found to function acceptably in both operating modes when integrated into the mainline.

Hence it was concluded that this new module had been successfully created and integrated into the main system. This also demonstrated that the inclusion of new units was possible and reasonably simple provided a local control system was adhered to when designing the unit.

6 Robustness testing & improvements

It was noted that, in its original condition, the run time of the model was limited by the time between critical failures, which often did not allow for the actual failure to occur. From the perspective of using the model of teaching this is a bad thing, as often the time until failure and throughput are the best quantifiers for the performance of the line. It was hence specified that the robustness, and fault detection capabilities of the line be improved.

In order to rectify these faults the line was repeatedly run until a failure occurred. The failures were logged and then resolved, allowing the running time of the line to be gradually extended as common errors were eliminated.

6.1 Robustness Improvements

6.1.1 Upstream Pallet Jamming

Problem:
Pallets arriving from the upstream unit onto the first mainline section arrived at a slight angle; this caused them to catch on the platform guard rail joint. This cause the pallets to be lifted up on the platform and to cause errors hen the transfer arm delivered the next pallet.

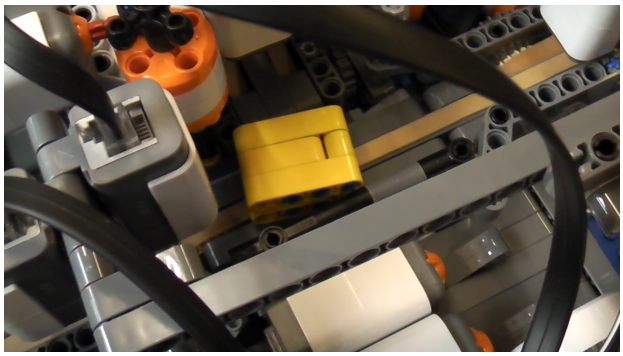


Figure 21: Upstream pallet becoming jammed on central column of barrier

Solution:
The barrier was redesigned such that the joint in the middle was made smoother. Hence the area on which the pallets got caught was removed. Guide rails were added to attempt to correct the alignment of the incoming pallets such that they did not have the orientation which caused jamming. This resolved this problem.

6.1.2 Mechanical Jam of Pallets

Problem:
Mechanical Jams at various points on the unit. Pallets would jam on the mainline in a variety of locations across the model. This could lead to pile ups of pallets if they occurred in a location that underwent t constant motion such as the mainline with fewer sensors to detect errors, or resulted in misaligned pallets at the transfer platforms.

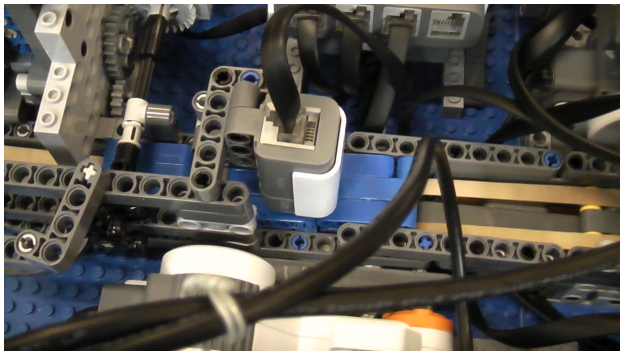


Figure 22: Collision of three pallets due to a mechanical jam holding the lead pallet in place

Solution:
In most cases the line was widened by moving the side barrier slightly further apart and the problems were alleviated. This type error seems to re occur with reasonable frequency, depending on how the unit is handled. Hence a permanent solution was investigated, but could not be found.

6.1.3 Feed Unit Reliability Improvements

Problem:
The upstream feed and occasionally other feed units deliver the pallets onto the line such that they fall off the back of the line rather than being carried in the correct direction. This had the effect of cheating the system of pallets and causing the feed unit to incorrectly calculate the buffer sate as the pallets never arrived at the transfer sensor, such that the state was always estimated as having had the pallet, causing an inconsistent operation of the feed line.

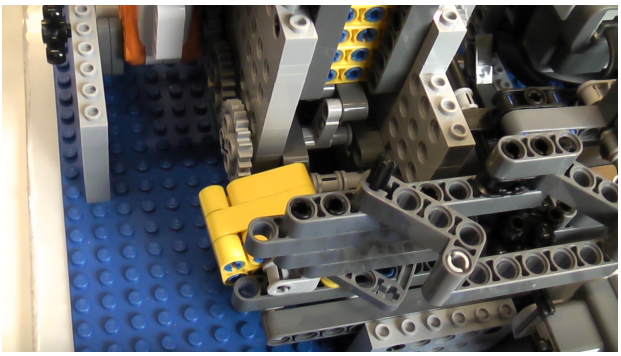


Figure 23: A pallet falling backwards from the feed point

Solution:
The feed units were observed to have two designs. One of these designs had a better reliability with regards to this; this reliability was investigated and improved upon. The new unit with the problem eliminated was then copied to all other feed units as part of the unit standardisation section of the project. This also eliminated a few reliability problems associated with some of the other feed units such as excessive shuddering of one which could lead to structural failures.

6.1.4 Robust MATLAB Instance Handling

Problem:
In general the instances of MATLAB opened by the initialise script were staying open after the end of the run; this caused them to remain in the processor memory until they were deleted by the processor; this was an inefficient use of resources. Also it was noted that where the program hit an error, most frequently failing to establish communications with an NXT that the instance of MATLAB remained open. Sometimes these instances held open the link to a particular NXT such that when it was next called upon by a new instance the program would fail to find it and create more erroneous instances. This caused inconsistent behaviour which could only be resolved by complete system shut down and user restarting of the computer.

Solution:
At the end of all scripts code was added to ensure the safe shut-down of the unit by disabling the sensors, closing the NXT connection formally and finally closing the instance of MATLAB. This code was designed to run even in failure, causing immediate shut down of the unit, assisting with the prevention of structural damage. The RWTH tool-kit was modified such that if the identification of a particular device failed the link was shut and the instance of MATLAB was closed immediately after a message was printed to the error logs.

The result of these software robustness improvements was that the frequency of restarting of units was reduced to zero and the number of user resets was reduced to zero, thus allowing experiment modes which performed a series of runs and then compiled results together could be developed. This made using the line for teaching far more achievable as a restart would cause work currently in progress to be lost.

6.2 Intelligent fault detection and response

6.2.1 Transfer Arm Error Correction

Problem:
It was noticed that when pallets were delivered to the front end of the feed line they did not always arrive in the correct location due to mechanical jamming. The result of this was that when the transfer arm attempted to transfer the pallet it forced it against the side of the line for a few seconds and then failed to operate correctly. This then caused incorrect operation of that section thereafter, with some pallets being simply ejected sidewards onto the mainline causing future jams.
Solution:
Adapting an existing solution which used the timer method to detect if an action had been executed correctly to diagnose the failure was relatively simple to implement reliably. Next a simple algorithm to retract the arm by the same angle executed, advance the pallet by an increment and then reattempt the transfer was implemented. It was realised that this occasionally advanced other pallet s from their buffer state locations and so a method of moving back these pallets by the distance moved forward was implemented to allow correct subsequent operations.

The system was tested, with a klaxon sounding to identify occurrences of this problem. Jams were simulated and the transfer mechanism was show to cope with all common misalignments of the pallet due to jamming. Thus the adaptation and implementation of this safety features was deemed successful.

6.2.2 Use of Intelligent Logic for Safer Shut Down

Problem:
In the course of developing the mainline speed control it was noted that when the line failed due to error the feed and transfer units could remain in operation after the finish command or failure of another section to complete a process. If the cause of failure was causing damage this is a non-fail safe option. The cause of this was the use of while loops in the control system, against the use of if logic in the new Mainline code.

Solution:
The transfer unit if logic was re-written to use the equivalent if logic, conditional on the unit not failing. A failure flag was added to alert the unit to its own failures, and all sub sections were conditional on this flag being in the no error state. A single while loop was used to hold the software in a loop until the STOP file was created either by failure of another section or by user issuing a finish command.

The feed unit was too reliant on the while loops to rewrite from scratch and such the local failure flag and Stop conditions had to be added to each while loop, to form a kind of while/if loop. This method reduced the instances of failures causing structural damage e.g. by a pallet being ejected from the feed lines and destroying the barriers on the main line platform and hence was deemed successful.

6.2.3 Protecting a Damaged Unit During Initialisation

Problem:
During the Initialisation procedure the feed unit and transfer units move motors to a given start position; determined by the pressing of a switch when the position is correct. Occasionally the hammer which struck the switch would fall off and hence the unit would never complete initialisation and leave motors running, causing large scale structural damage to the unit.

Solution:
It was noted that it usually took the same amount of time for the initialisation to complete once the movement of the motor had begun. Thus if the time taken for the switch to be pressed was greater than this fixed time it was likely that damage had occurred to the hammer and hence the unit should stop. Hence for each such section of the initialisation procedures a timer was initiated at the start of movement, and if the timer was elapsed failure would be declared.

The system was tested and implemented and was found to prevent additional structural damage above the failure of the hammer. The cause of the failure was always hammer failure and so a failure dialogue was provided to allow the user to intervene to correct the exiting damage.

6.2.4 Alerting The User to the Cause of Failure- Intelligent Error Diagnosis

Problem:
In general it can be observed where a failure occurs, however this has required a practised eye in some cases. For a new user it can be more difficult to understand the difference between an expected buffer overload error and having to track down a more unique sensor failure error. The former can usually be determined by eye, however sometimes it is hard to distinguish the two cases.

Solution:
Using the failure flag introduced previously, an error type variable was employed. At any event which raised the failure flag an error type was changed to reflect the error. At the end of the operation a unit which had failed wrote a small error file which contained the unit identifier and cause of error. Upon the finish command being issued the file was read and the errors reported.

6.2.5 Further Problems with the Transfer Arm

Problem:
Transfer arms were no longer placing pallets on the line properly after the code base was edited to correct

developments made since the original publishing of the code. In so doing the error case had been modified; it had not been noted that the timer on the arm swing time was too short and hence the transfer always entered the error case and was using this to retract.

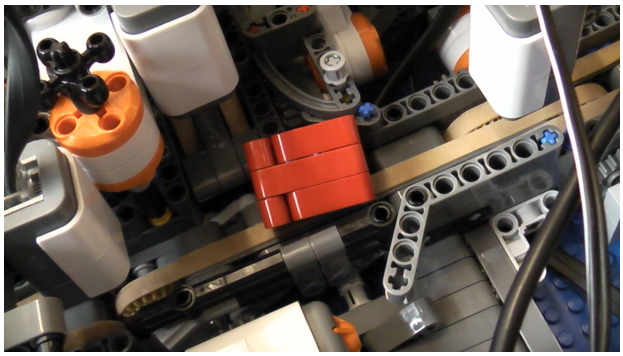


Figure 24: A pallet being placed at an incorrect angle by transfer unit arm

Solution:
A new time was estimated by calculation and considering the costs: if the time was too short the line would never be able to complete or would misplace the pallet; too long and structural damage to the barrier would result. The new time allowed the original transfer procedure to be resumed and left the error case for the true errors without inducing any new problems.

6.3 Mechanical

All elastic bands in the system were changed as of 28 August 2012.

7 Improvements for teaching purposes

7.1 Legoline GUI Design

Problem Statement:

- Existing user interface involved typing commands at the matlab prompt to operate; this was not very user friendly
- No proper logging of errors or failure making the cause of the line stopping, or otherwise failing, hard to detect and hence harder to correct.
- Feed Time Data existed, but was buried in the program files and not easily accessible unless its location was known initially.

The above factors limit the suitability of the unit for its purpose as a teaching system. Students using the unit would spend longer attempting to become familiar with the interface, and attempting to locate the output data required than performing the experiments the unit was intended for. In so doing they would fail to grasp some of the reasons why the unit is being used and what it can demonstrate about operations management and/or control system design.

Solution:
Design a simple GUI which can be run from MATLAB and contains familiar buttons and menus which hide all of the scripts and commands behind a simple to use interface of buttons and drop down menus familiar to users of windows, Linux and Mac OS environments, in a manner which is consistent with the OS on which the program is being run.

Features of the New Interface

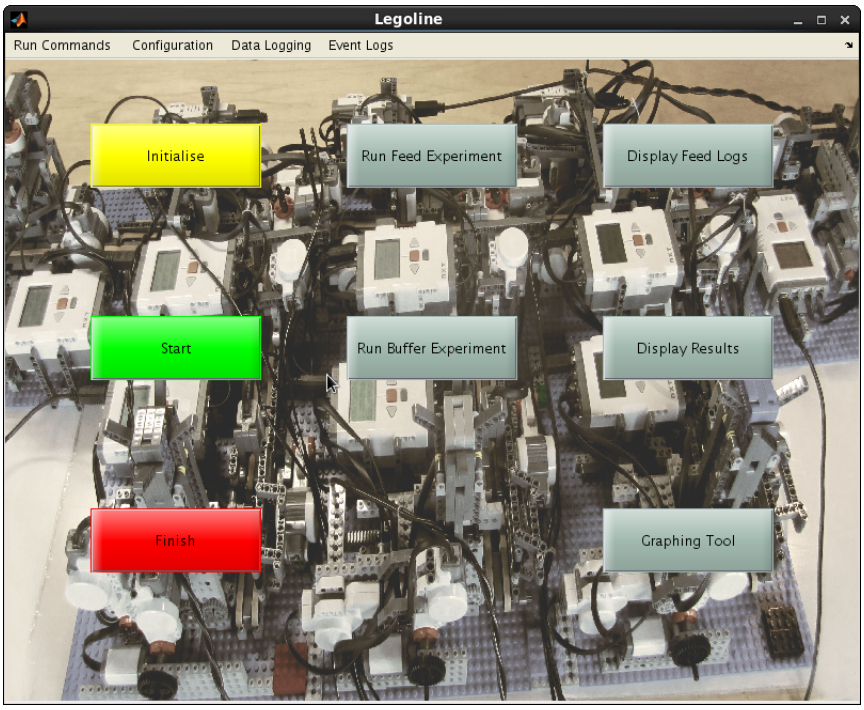


Figure 25: The Mainline Interface Window

- Command bar housing all possible running commands and data viewing commands including opening individual or groups of log files
- Dashboard of buttons for common commands created for more user friendly operations.
- Log files generation was changed to generate more readable log files which were saved in a well-marked folder for ease of access and editing.
- A failure diagnostic pop-up system- the reason for failure was written to a file, which was read by the finish command such that a cause of error could be displayed, allowing the source of error to be pinpointed and the appropriate log file to be reviewed.
- Log file generation was changed to include step by step process guide and timings, complete with headings and date/time of creation.

The software could be further improved by allowing the path to where the log files would be stored to be selected by the user. This was later superseded by the development of the archiving interface, which allowed all of the log files and data files to be compiled into one location.

User Review:

The software then underwent end user review by Dr A Komashie out of which a number of further corrections were made:

Firstly it was observed that accidental selection of experiment mode during user exploration or by clicking them in error would lock the user into experiment modes. Each of which would take either a full shut down of MATLAB and a restart of the line or half an hour to complete. This was rectified, as suggested in the review, by including dialogue boxes which the user must confirm to begin an experiment.

Secondly there was scope for error when using the open log files features: sequential runs overwrote old log files, but if a unit was not used in the second run the original log file would be left. Thus when all log files were opened or saved some were out of date or not relevant to that particular run. This could only be determined by the user opening the log file and observing the date stamp at the top of the file.

This fault was overcome by purging the files as part of the initialisation procedure. Hence when the log files were generated at the end of the run, only those corresponding to the units which were involved would be created. This left no old log files in existence; however the GUI needed to be periodically updated to capture these changes, as when it generated the list of available log files it did not take into account the existence of the files. Thus an update function was built into the GUI which allowed it to poll this data periodically such that it only offered the most up to date log files to the user.

Finally since data was overwritten an archive function would be required to allow students to save and load data conveniently. This requirement was satisfied with a new interface, the archive interface section 7.3

Overall the response to the interface was positive, with the simple layout, and ease of sue being praised, but the allowance for user error needed improvement. These modifications were made and demonstrated, and the interface was approved. This section of the project was deemed to have concluded successfully.

7.2 Graphing Tool

Aim:
To improve the experiment mode by allowing generation of figures from existing data without requiring user knowledge of the Matlab plot commands etc. and to make the output figures suitable for inclusion in reports.

Requirements:

- Ability to plot light sensor history, and threshold where appropriate, in order to assist in failure diagnosis and possibly in outreach activities
- Ability to graph 3D failure surfaces to give a simple visualisation of the performance of the unit.

Solution:
Design a GUI for performing graphing of the various data sets in a user friendly manner. The problem is split into two areas:

1- GUI design and 2D dataset plotting
2- 3D surface rendering to allow plotting of the failure surface

2D graphing was achieved using standard matlab plotting commands- a list of all available sensors was com-

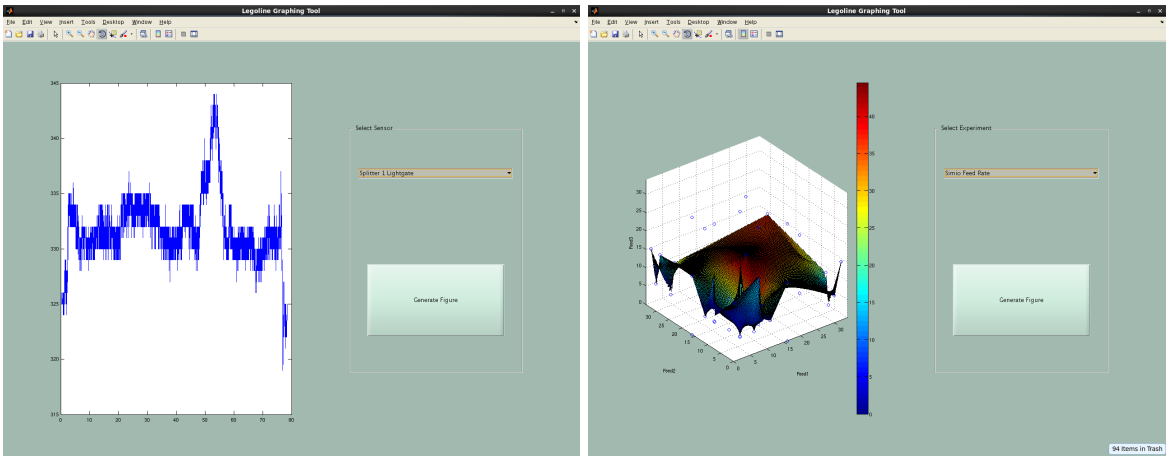


Figure 26: A-The Graphing Tool for Light Sensor Data(left) B- The Graphing Tool for Failure Surface Data(right)

piled and a menu which filtered by availability was constructed. A means of previewing the graphs was created in the GUI window. Finally a script was written which could generate the graph as a saveable matlab figure, complete with title and axis labelling. This enabled users inexperienced in matlab to swiftly produce report quality graphical items kin a simple and friendly manner.

The 3D plotting functionality was more difficult to accomplish. A range of 3D surface rendering techniques was investigated, and the graphical interface adapted slightly to accommodate a 3D preview axis. The graphs themselves were acceptable, however the surface fitting software proved more problematic.

3D-Plotting

The data to be plotted could come from two different sources: experiments run on the physical Lego model or simulations run on Simio. In both cases, the data produced would be in the form of a spreadsheet. The spreadsheet would have 3 columns for the feed rates, 1 column for overall throughput of the system, 3 for buffer values, and 1 for a failure flag.
On the failure surface, each point that we plot has 4 values: 3 defining its position in 3D space (corresponds to the 3 feed rates) and one corresponding to colour intensity (which represents the throughput value at failure). These points have to be extracted from the spreadsheet; the failure flag is used to filter out these data sets.

The result is a set of scattered data points, not practical for plotting a surface. To be able to plot in Matlab, it was necessary to generate a uniformly distributed data set in the form of a grid. Throughput values (represented by a colour scale on the graph) were then 3D-interpolated over this grid.

The uniform grid of points was generated using the meshgrid function for a mesh in the x-y plane. z values were then interpolated over this grid using the TriScatteredInterp function. Colour values were similarly interpolated. The next step would be to generate a surface using these points.

Delaunay triangulation was briefly considered; however, it gave rise to tetrahedrons given the data was in 3D. This meant that we were generating a volume rather than a surface. Backface culling or hidden surface removal might have been a solution to converting the volume to a surface; however, it was thought that Matlab was not the most suitable software to be using for such graphical manipulations. Delaunay triangulation combined with convex hull was attempted but gave rise to a volume that enclosed all the data points rather than a surface passing through the points.

Other attempts to generate the surface included the use of the patch function. The patch function gave good results for a small dataset but it was found to generate twisted shapes for the data set we had. This was because the patch function requires the points to be given to the function in the a specific order to be able to generate the vertices between each set of points. For such a large number of data points, it would be unrealistic to manually reorder the points. This approach was therefore abandoned.

The final version of the graphing tool used the surf function to generate the surface. However, given how sparsely distributed the points were, it only gave a very distorted shape which often was between data points. Ultimately no method of successfully rendering the surface from the sparse data points generated by SIMIO was found. It was decided, therefore, to review the SIMIO model to attempt to generate a more evenly distributed set of points from which to plot the failure surface. A script file which may be edited was generated and placed in the GUI folder. This was done such that if a solution became available in future integration could be achieved simply by editing this script file, and not having to trawl through the graphing interface code. However despite this minor shortcoming the GUI was approved of and deemed successful.

7.3 Archiving Tool

As requested by Alex Komashie during his review of the Legoline GUI, a means for students to save work or results would be very useful - particularly with regards to fourth year projects and gathering sufficient good quality, presentable evidence of performance.

Requirements:

- A place for user to generate a meaningful name for an archive
- User friendly method of selecting the relevant data for storing in the archive
- User friendly method of selecting a place for the archive to reside

A provisional layout was drawn up and agreed upon; code was written and tested to perform the various functions. The user had the ability to freely type the archive name into a labelled text box; otherwise a default name of archive, with a time/date stamp would be created. MATLAB's interaction with various operating systems was exploited to allow the user to select the folder in which to create the archive in the same manner as normal navigation i.e. a pop-up file browser window. A set of check boxes clearly displayed the data to be included in the archive in a simple to understand manner, and was set apart from the other check boxes to keep the screen uncluttered and easy to understand.

The new GUI was then debugged and reviewed by A Komashie .After this review; as per request; a new

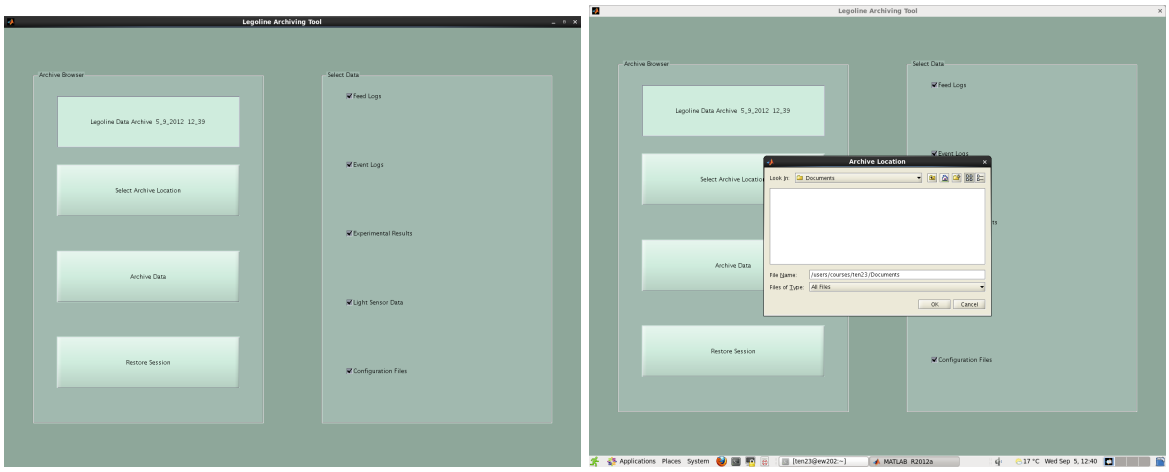


Figure 27: A-The Archive Tool Interface(left) B- Archive Location Browser(right)

'Load' feature was added - this was to allow students to reload old data if they had not finished processing the data at the end of an experiment, and it had somehow been erased in the intervening time period. This was implemented with a load button which allowed the user to graphically navigate to their save using the file browser window; this kept the software interface familiar to the user.

7.4 Creating Experiment Mode

Aims:

- To generate a script which can take a series of parameters from the configuration file as inputs and perform a sequence of events to find points on the failure surface of the unit in the same manner as the SIMIO discrete event simulation model.
- To extend these scripts to allow feed rate or buffer size experiments to be performed to quantify the behaviour of the real system, and highlight the differences in performance.
- To deliver an output results table from the line as a complete digital copy of the current teaching sheet.

Implementation:
The first step in implementation was to copy the process flow from the SIMIO model and adapt it to suit the capabilities and limitations of the physical model. This was done and an algorithm was then developed to perform this flow.

The algorithm was tested and debugged, and then a series of user instructions which would display to the screen the interactions the users must make with the line were added. These were again tested and shown to be quite helpful in following the process flow through as well as reminding the user to restock the unit and check for blockages etc.

In order to decrease the time taken for experiments to be performed a physical limitation on feed rate was noted. This allowed three data points and all associated data to be provided immediately to the user, without them having to search for the points. This reduced the number of experiments by approximately 8 or about 20 minutes work time.

The data table generated was made to look more professional such that it could be printed out and appended to

a report. Data tables which interacted with the graphing software were also generated, allowing the theoretical and practical versions of the same failure surface to be plotted together and allowed for comparisons to be drawn.

Evaluation:
A successfully series of program enhancements were made which allowed a user friendly approach to performing the experiments required in teaching. Flexibility was included to allow development of experiments for other areas, such as varying mainline buffer size by copying one of the existing scripts and performing minor adaptations to allow for a different range of experiments to be performed once new capabilities are added to the line.

8 Experiments

8.1 Extending the line to 5 units

Aims:
To Investigate how 'modular' the line is: I.E. how easy it is to allow more complex systems or new unit types to be added.

Currently the only new units which may be added are the existing ones. This experiment will aim to extend the maximum line size to five units.

Methodology:

- Start at High Level software i.e. control files such as initialise and move down until at unit level software (i.e. not sensor reading code or buffering code). Initialise had to be changed to open more instances of matlab for each of the new NXT hubs added, this allowed the system to remain under strictly local control.

- Add new Start-up files for each new unit added (6 files), these setup the new instances of matlab with the correct workspace variables for new units and read the correct data files in from the configuration files.

- All sections of code which relied on three units were changed to allow at least 5

- All similar references in the setup file were altered accordingly to allow for five units to exist with all the appropriate sensors.

Debugging of the new code was swift due to a lot of the changes being effectively iterative: sections of code could be copied and the appropriate variables renamed to allow operations. This type of coding is swift and minimises the number of errors generated in writing. Having got the code working to allow the units to operate extension of the logging and data gathering were added at a high level. New files to be generated were added in the Setup script and the sensor archive was expanded to accommodate the additional sensor files which the new units required.

Finally the configuration and master configuration files were changed to accommodate the new lines in a user friendly manner. A series of simple changes by extending existing tables were made to allow for new units to be added.

After all modifications were completed the line was tested to ensure that no functionality that already existed was lost by the modifications.

8.2 Cross Platform Software

Aims:
Initially the Lego control software used hard coded links to various folders in DGD24's user account, and relied on having this included in the MATLAB path data by the system. This user account was only available on two computers. Currently the location of future work is unknown and so it was decided to ensure that the unit could be moved, and that any computers in the vicinity of the new location could be used to run the unit.

This would also be useful if the line is to be used for teaching, as sometimes it is impractical to use the same computers, or if people want to use the teaching system rather than the dedicated PC's a degree of flexibility in the software would be a great advantage.

Implementation:
Firstly the code base was analysed and all areas which needed hard coded links were identified; a list of all files and their locations and requirements was drawn up. A new single file system was designed in concept and a copy of all files placed within this folder.

A path master file was created mapping all of the sub directories that the program files remained in, excluding those the user may modify. Two matlab scripts were developed, one which located the current directory, and added all of the new paths to it, and one which generated the paths as variables within the matlab workspace and appended them to the matlab path on request. This allowed all of the existing hard coded links to be deleted and new ones using the variable names to be inserted. Hence the computer generated its own paths independently of the OS used or the location of the new Legoline folder. This data was saved in a path master file on start-up, from which all other matlab could load the new path. Each instance uses the default matlab path and so would not have access to the new scripts- hence the path generating script was used to alleviate this problem.

Debugging was then carried out in-situ on the old machines until the Legoline folder could stand alone and the control system functioned satisfactorily.

Next the program was tested in the DPO. Under a normal instance of matlab the program failed to communicate with the NXT's with errors relating to the tool kit. This was due to the tool kit not being included on the matlab path. A new path to the tool kit was added, superseding any other edition of the tool kit if it already existed. The program still generated error messages. This was a standard feature of the tool kit and MATALB and so warning suppression was employed to eliminate the warnings.

Having updated and included a modified version of the tool kit with one or two refinements to the Legoline folder the software was re-tested in both the DPO and EIETL. In both cases communication with multiple uses was successful. However when running the Legoline program the version of matlab installed had issues with the copy file command of matlab. Shell and DOS commands executed from within matlab were employed on the advice of Tim Love to compensate for this shortcoming and allow continued cross platform operations. So far both platforms investigated had already had some work done to accommodate the Legoline, and the requirements of the RWTH toolkit were already met. It was hence decided to investigate the process of installing the toolkit from scratch under both windows and Linux.

The installation on both platforms was investigated and a series of installation files and scripts was generated and compiled into a folder to assist future users in setting up the line swiftly and with a minimum of additional effort.

Conclusions:
The Software was modified such that, providing the RWTH toolkits requirements were met the software could run on any machine; increasing flexibility of use.

To aid with this an Installation guide was put together including information about updating and preparing new NXT units from scratch and how to use the software. To make some more intricate processes simpler a series of Windows Batch files and shells scripts for performing the installations were written. The Legoline control system became a standalone folder which incorporated the toolkit and all mapping software; it was designed to be self-contained within matlab and its own directories: Hence the program could be run from a USB memory drive or similar small storage device rather than requiring a specific PC or system. This is dependent on system settings as some set-ups automatically prevent the running of programs stored on such drives, which may be problematic when it comes to the NXT update software. However the main program does in fact run on several different Linux distributions and on both 32 bit and 64 bit versions of Microsoft windows.

8.3 Multiple PC Operations

Aims:

- To investigate the robustness of the local control system
- To investigate the changes in performance with changing processor power of computer

Currently the unit employs a local control system with each unit interacting independently of each other; however all units act on a single PC, despite being separate instances of matlab. This can cause the processor to max out, creating conflicts over which command get processed in what order and potentially allowing pallets to bypass sensors between two readings even though the sensor and program would react faster. Since the control is local and in separate instances it should not matter if they run on the same PC or not hence no degradation in performance was expected.

Methods and Tests:
This behaviour was investigated by splitting the line into two sections and running them on separate PC's but at the same time.

- Divide the line into two subsections of feed lines
- Attach each to a separate PC by USB cables
- Monitor behaviour under different running conditions

Observations and Results:

- No degradation in performance under standard operating conditions
- Experiment mode did not generate sensible data tables
- Three arrangements of the line and sub configurations were deployed.
- Both PC's were still saturating all processors on processor usage throughout the operation

Conclusions:
The Line was shown to be fully locally controlled, as if required each unit could operate from a separate processor at the extreme case of this argument. The only problem would be ensuring all of the units were started at approximately the same time to allow the feed schedule specified in the configuration to be followed reliably.

It was noted that this type of running was not suited to experiment mode as the data in this mode is pooled from all units, hence by having the units unable to present data for external analysis the results of any experiments are meaningless. A single PC running all instances is still a requirement of the system.

From the observation made about the processor performance it was noted that the current computer system did not have a sufficiently fast or efficient processor to cope with even half of the normal nine units without limitations being hit. This suggests that to get reliable results for the performance of any control system deployed a more powerful processor is required.

It was also noted that several configuration files had to be set up for each run and were a liability for introducing errors into the system such that the behaviour of the line was not as expected. This is due to the doubling up of the configuration data into two files; also both lines are treated as shorter lines, and act independently by naively copying the files between lines the desired behaviour would not be observed.

A Flowcharts

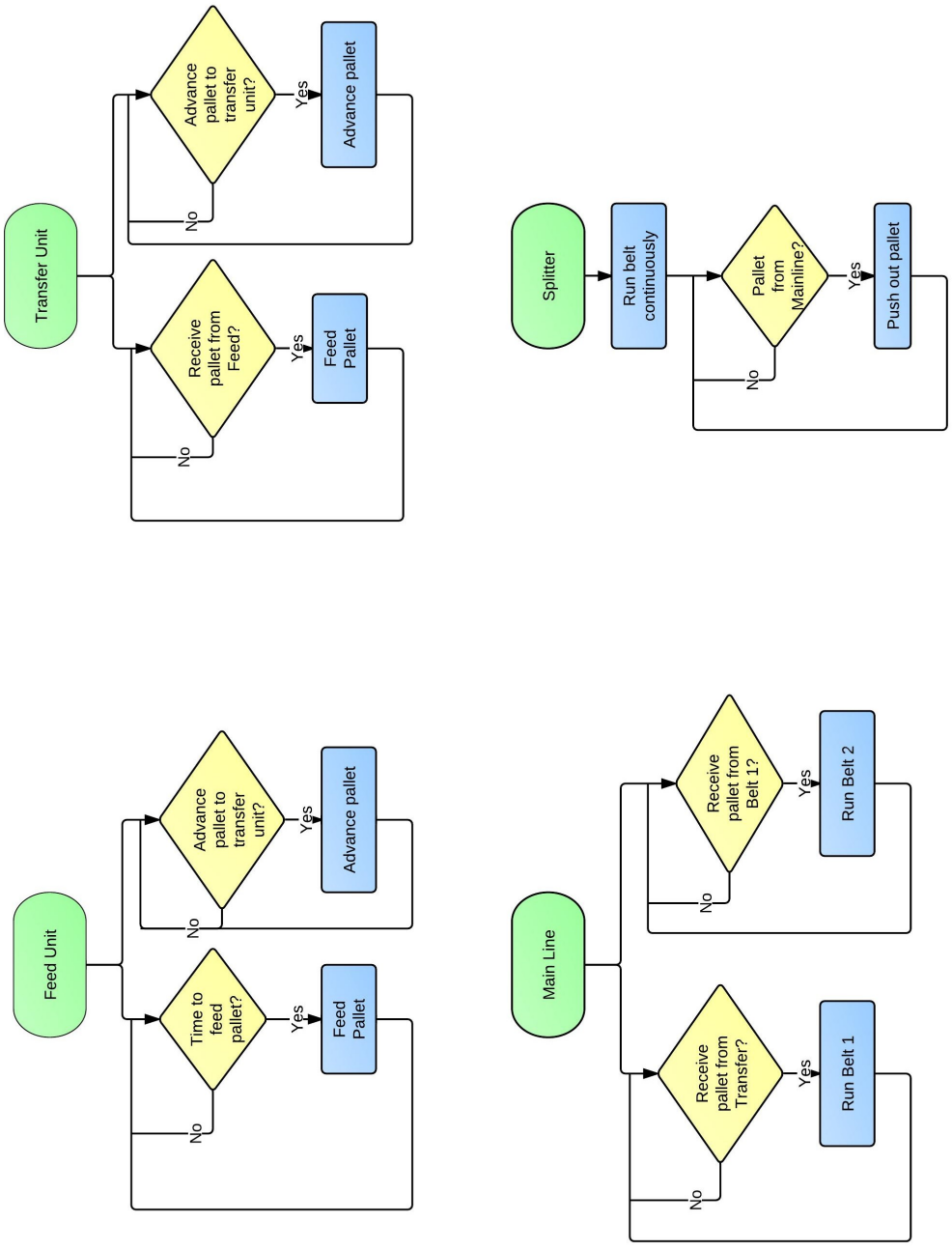


Figure 28: General Flow of Pallets in System

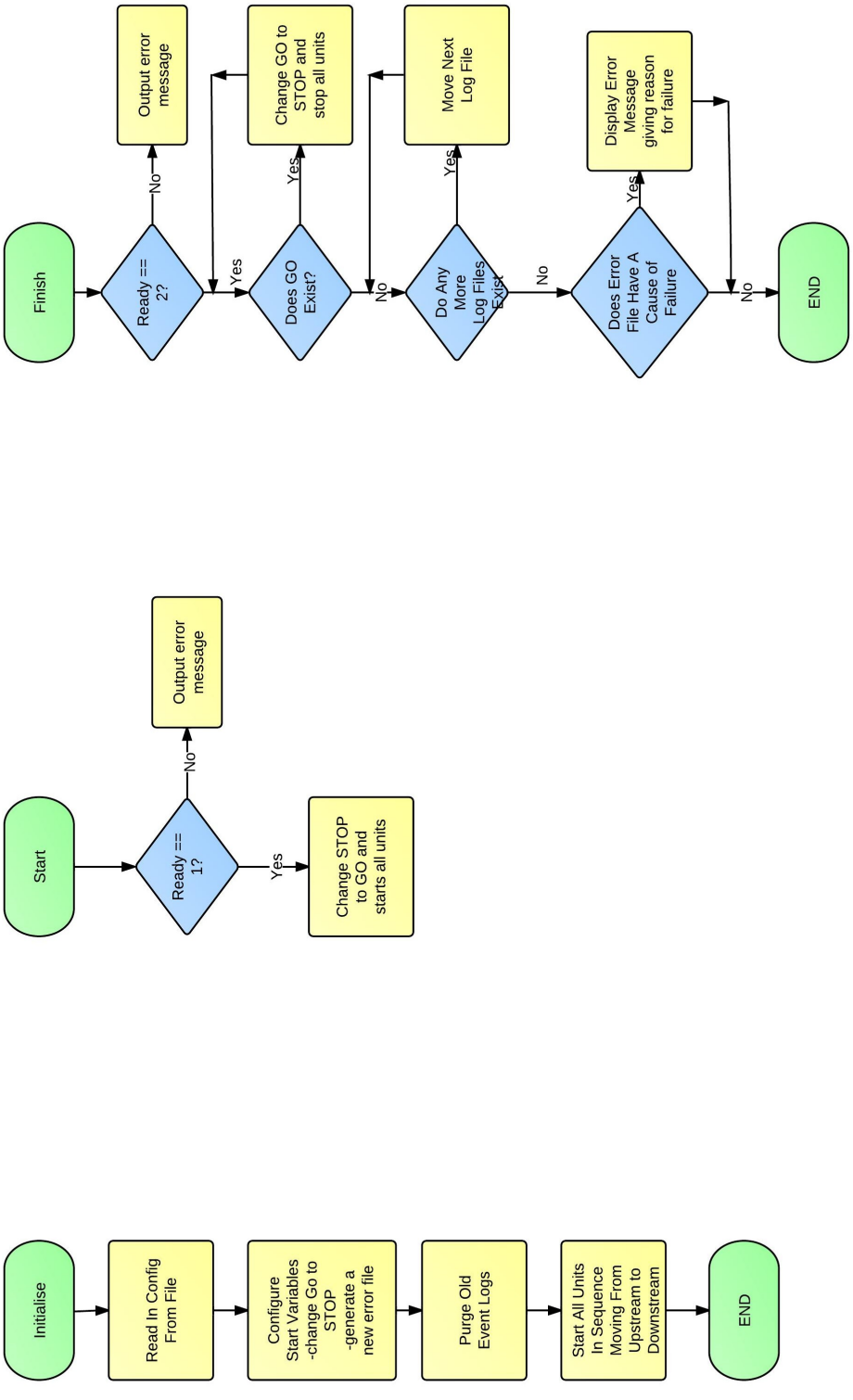


Figure 29: Initialise, Start and Finish processes

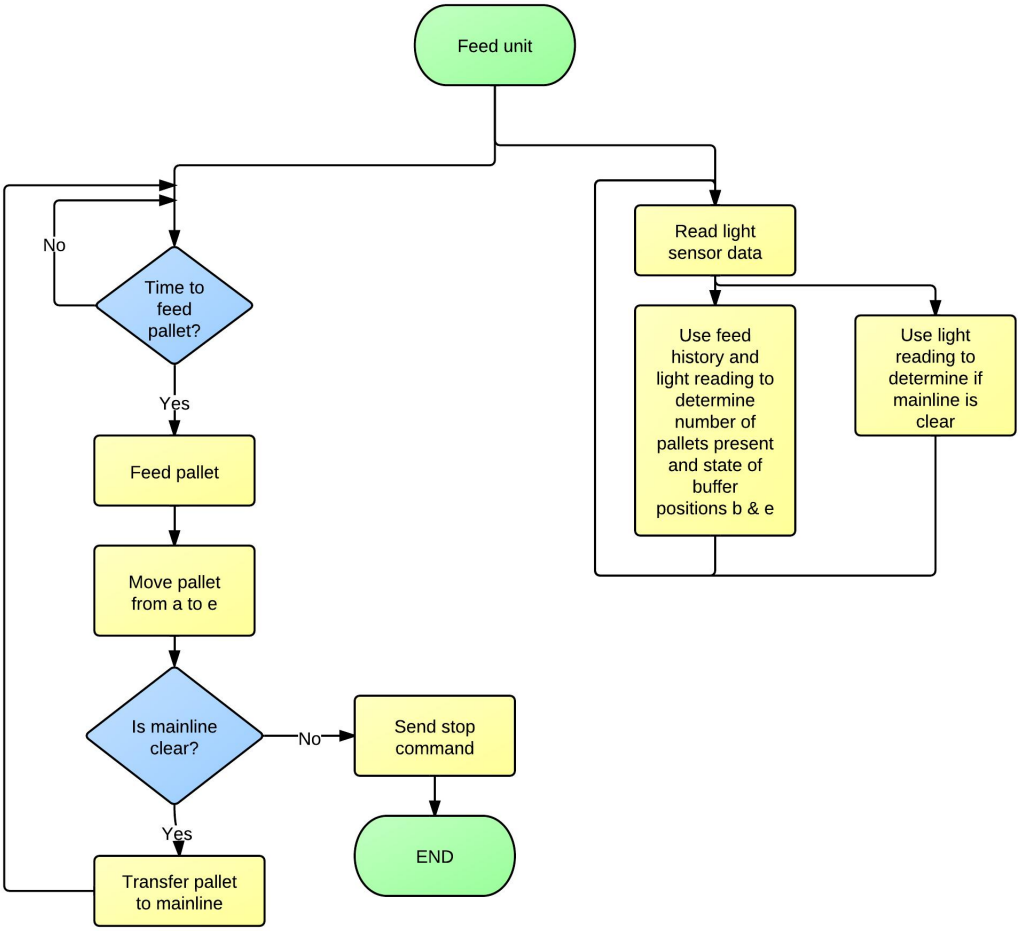
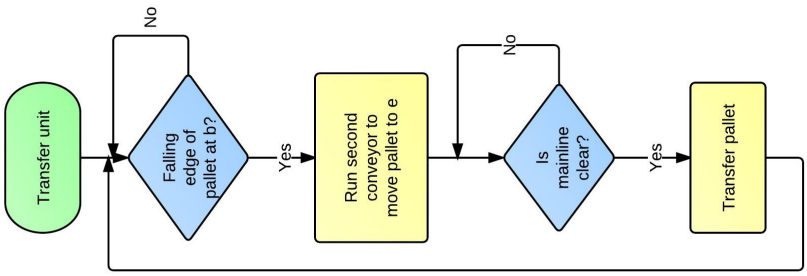


Figure 30: Feed and Transfer Units - Buffer 0



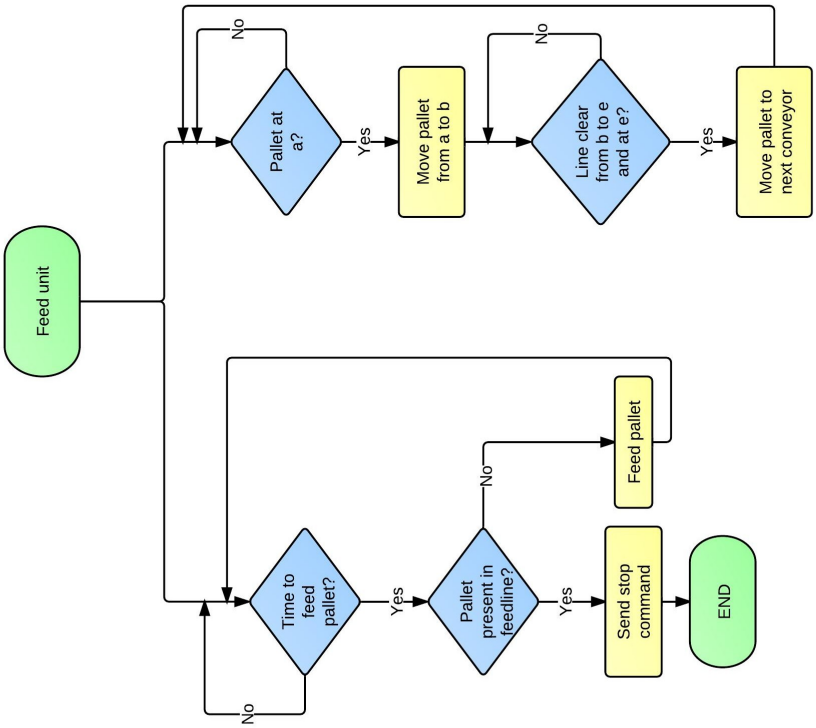
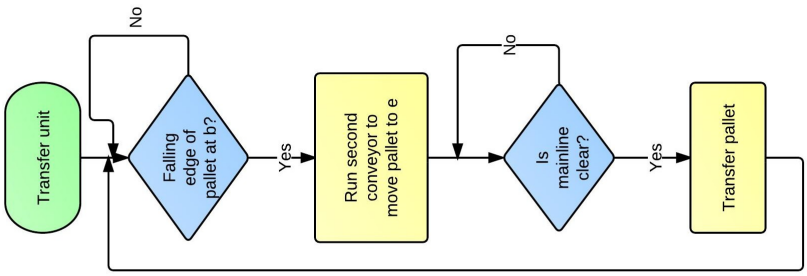


Figure 31: Feed and Transfer Units - Buffer 1



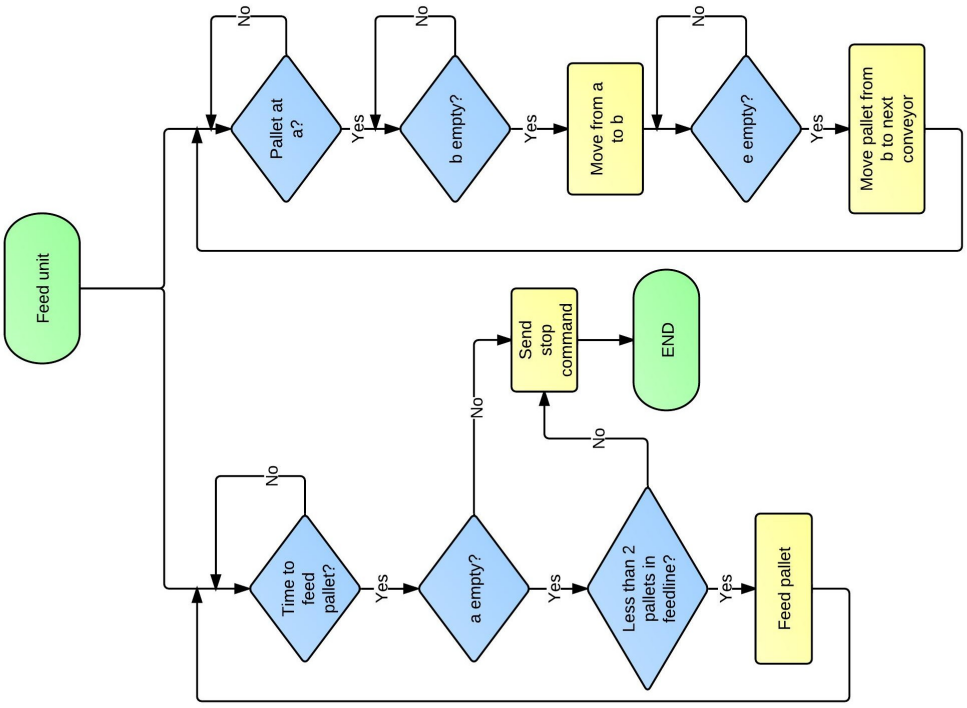


Figure 32: Feed and Transfer Units - Buffer 2

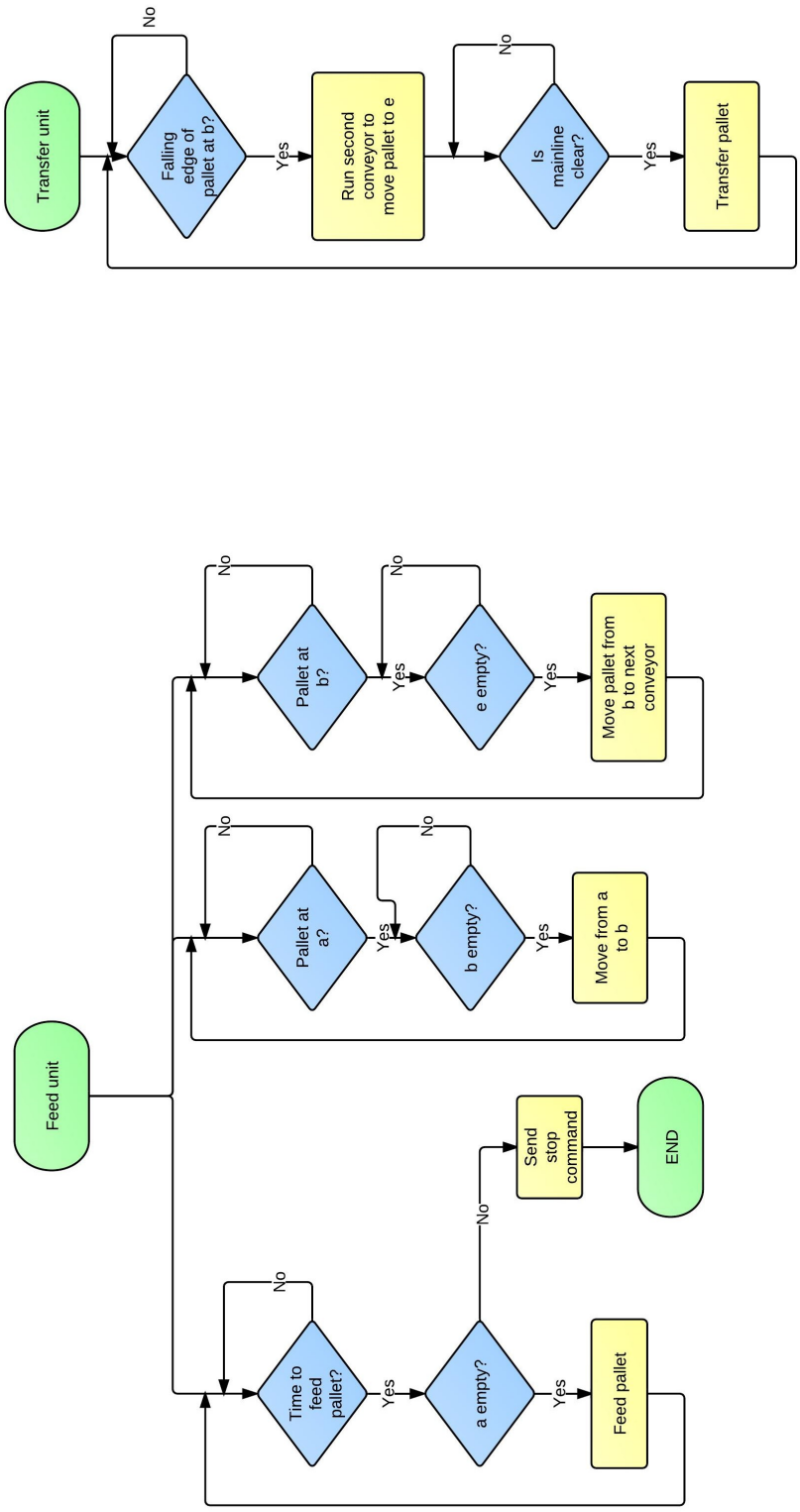


Figure 33: Feed and Transfer Units - Buffer 3

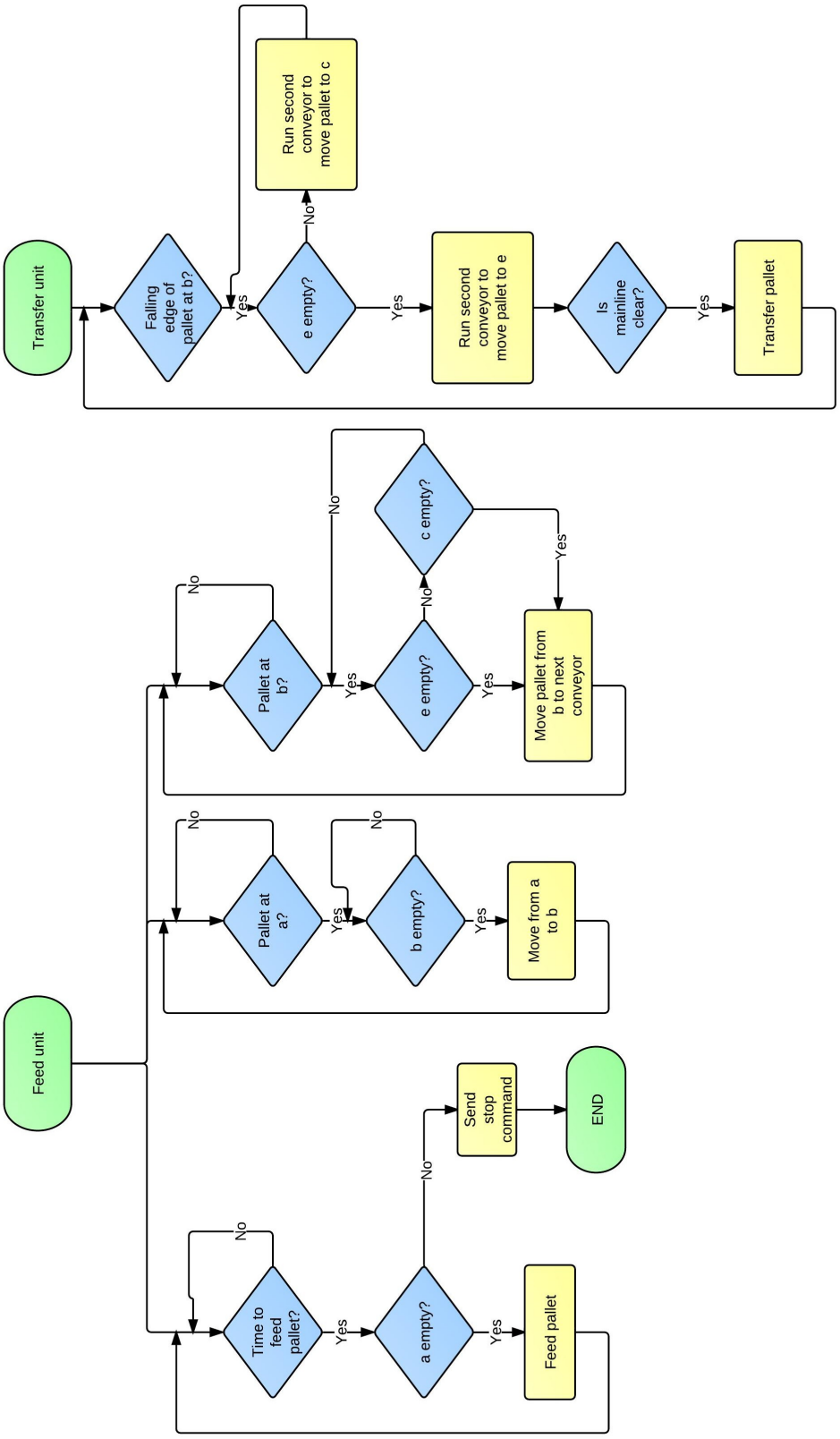


Figure 34: Feed and Transfer Units - Buffer 4

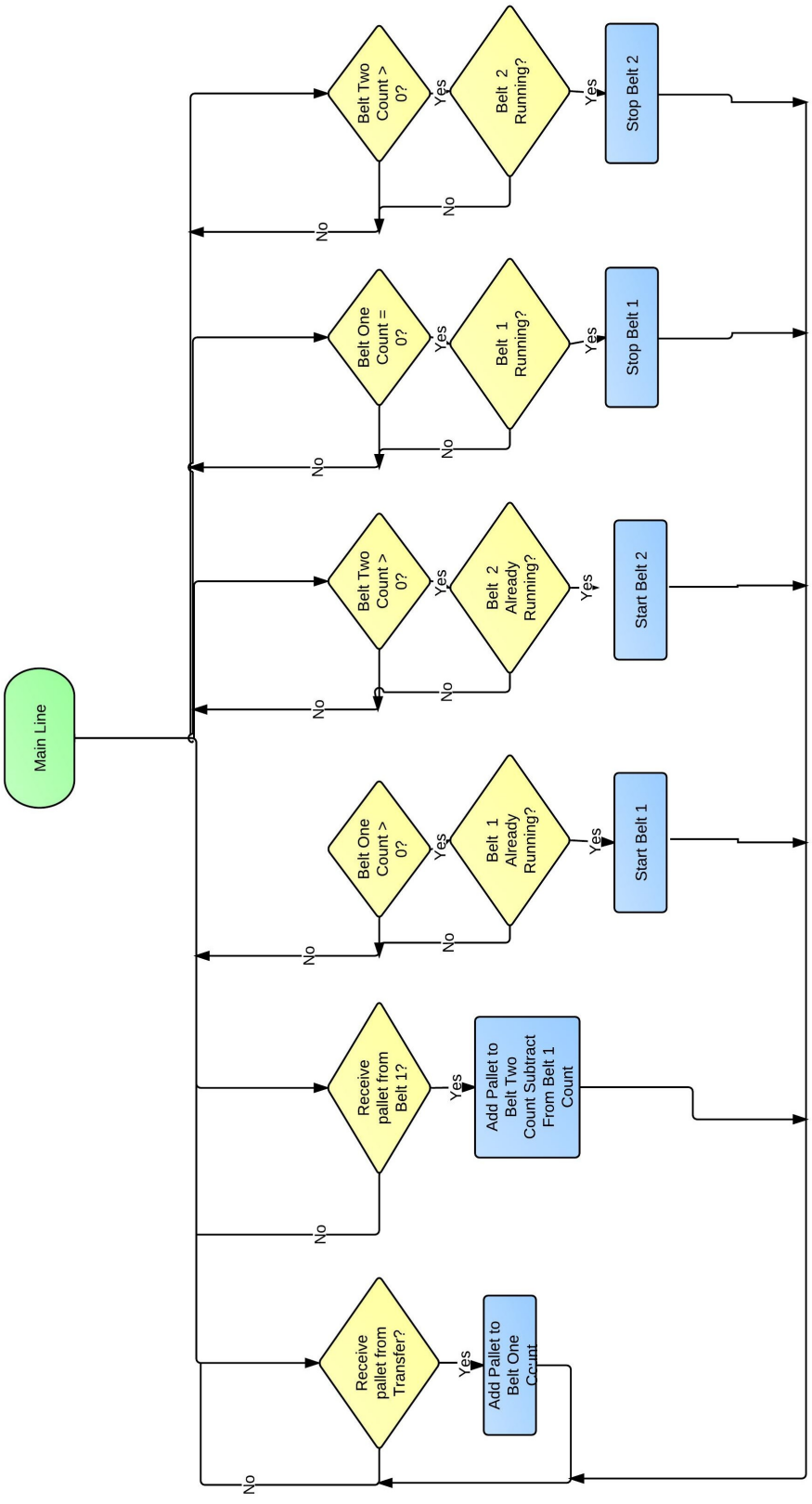


Figure 35: Mainline Buffering - Buffer size 0

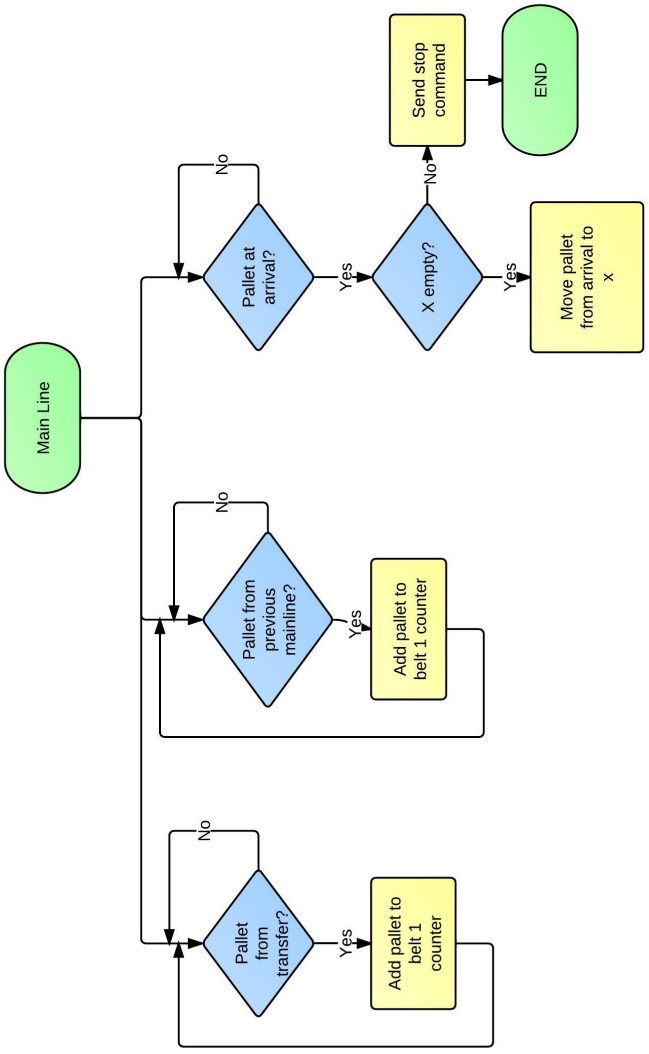


Figure 36: Mainline Buffering - Buffer size 1

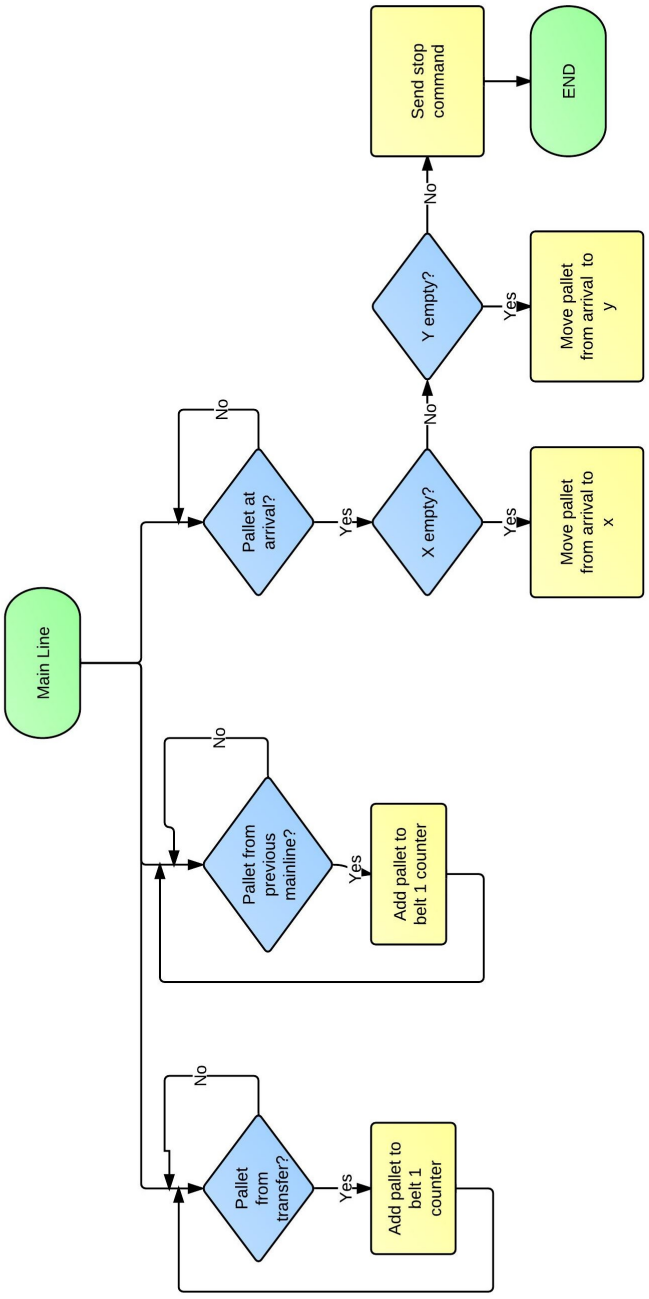


Figure 37: Mainline Buffering - Buffer size 2

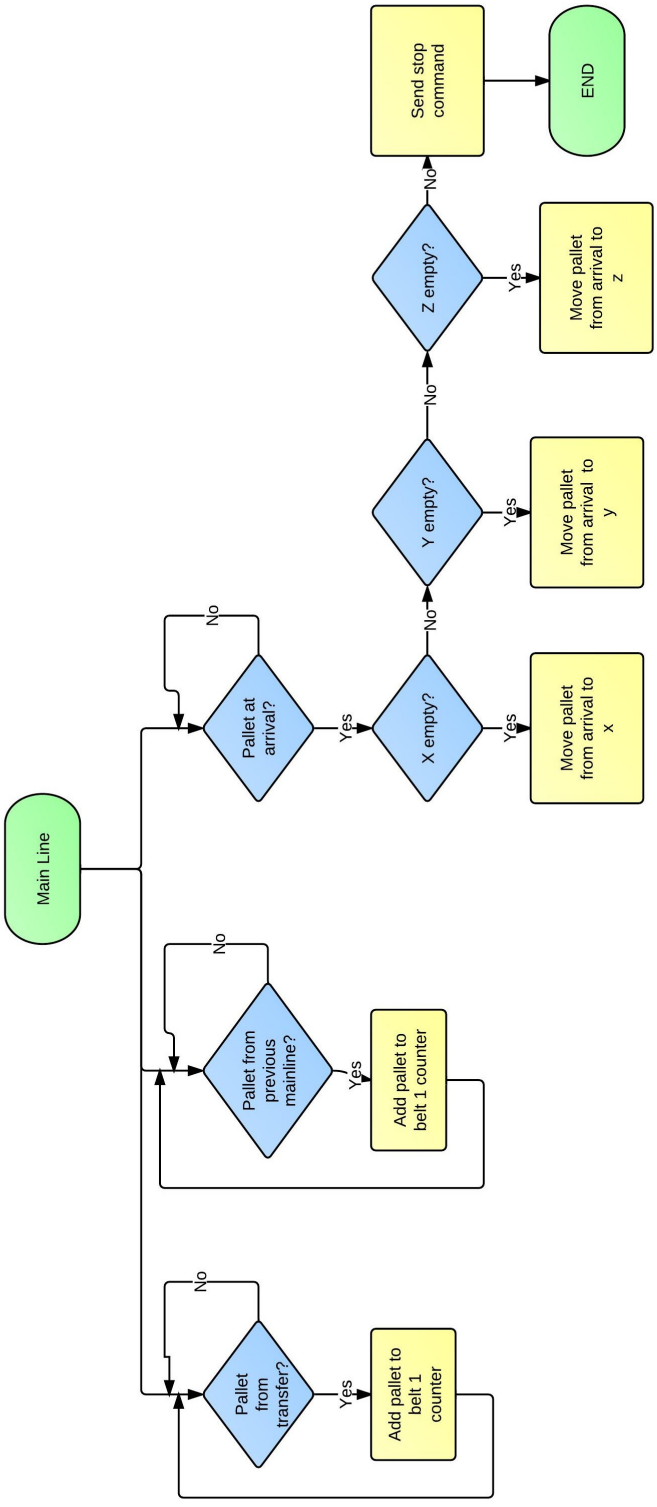
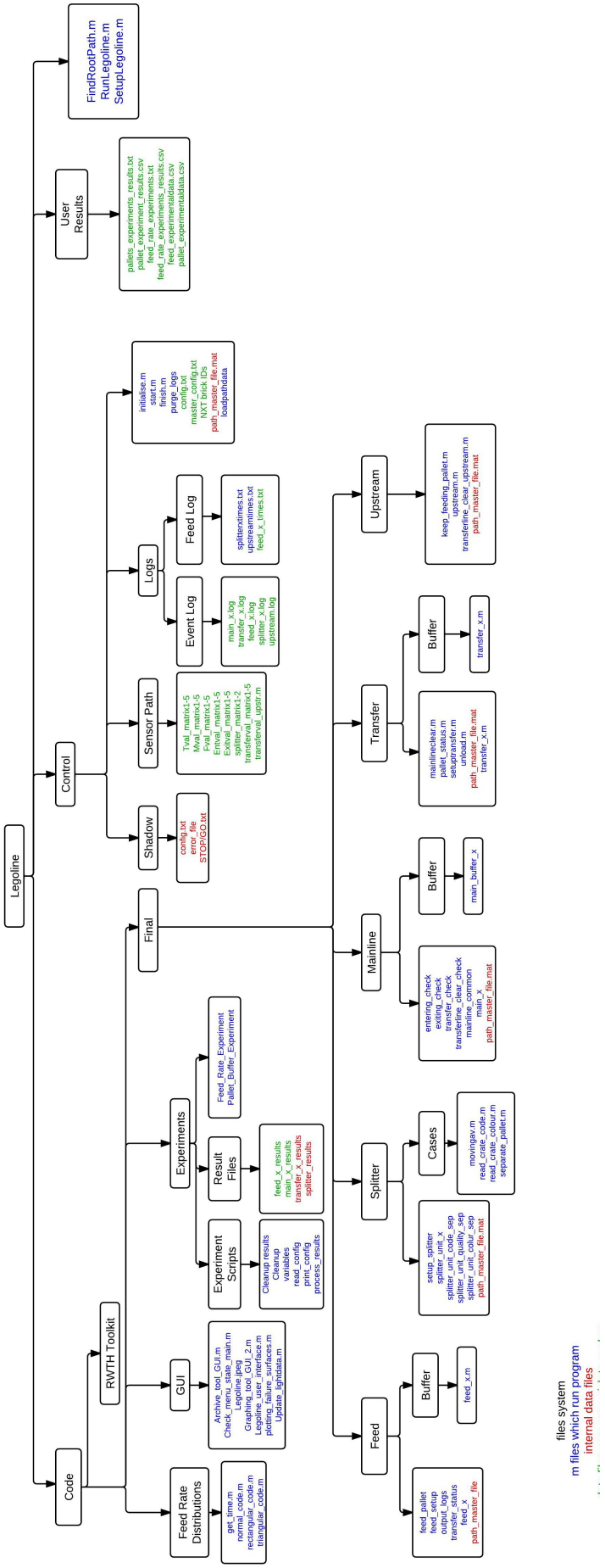
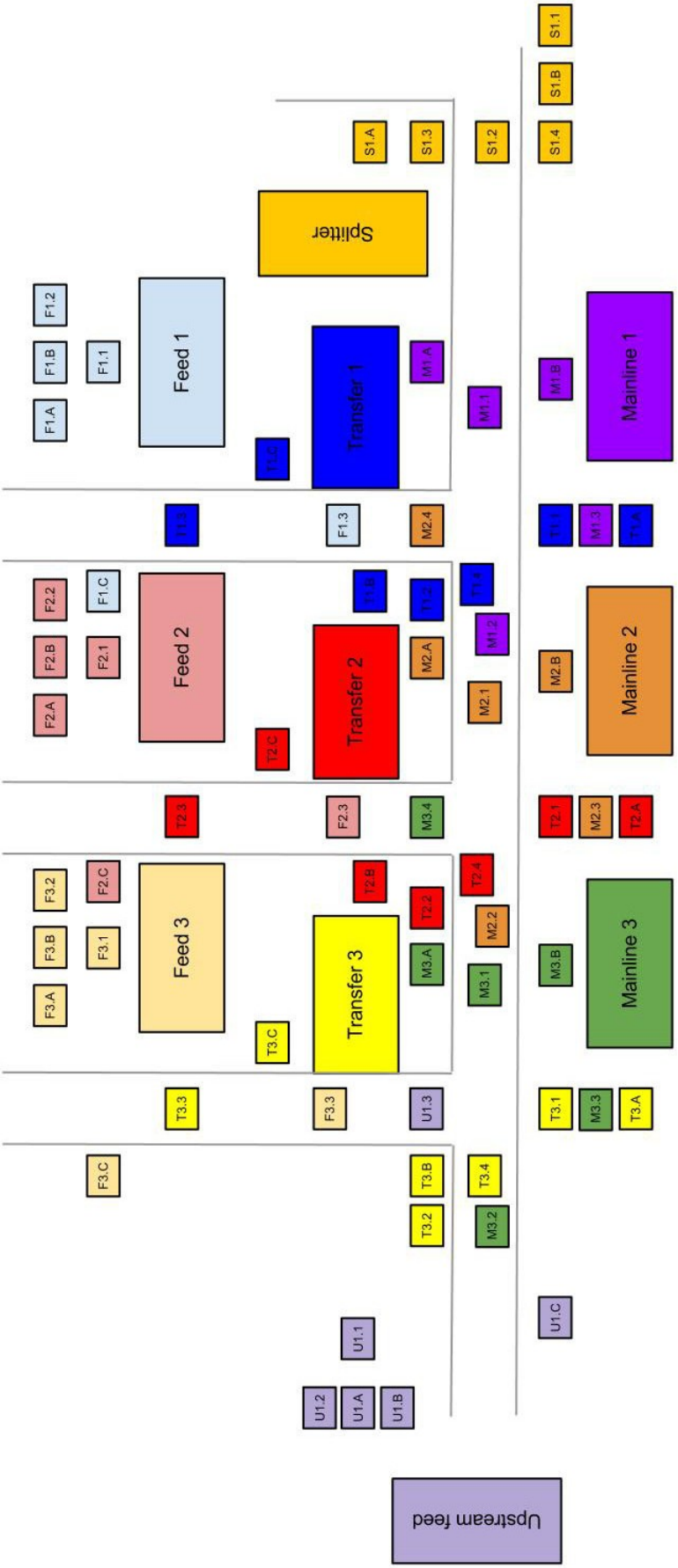


Figure 38: Mainline Buffering - Buffer size 3

B File Hierarchy



C Sensor layout map



D Lego RWTH Toolkit in Matlab Syntax Guide

D.1 Initialisation

General Guidance - it is sensible to use the tab auto complete command to finish filling out the name of the commands when typing the commands have a lot of capitals in them and so it is best to let Matlab do the hard work - errors can be avoided by this method. and time is saved in debugging simple spelling mistakes. This also works for variables- just type the start of the command or variable then hit tab Matlab will fill out the rest for you or offer a choice of possible auto completes- select the correct one and hit enter to fill in.

where specified sensor ports are identified by the numeric terminals on the NXT brick into which the sensors may be plugged and motor ports are identified by a letter.

D.1.1 Establishing a Link

Set the code as a string as follows and include at top of file so can change NXT easily

Call the following where handle is a descriptive name for the NXT - Will use the wrod handle here to mean this handle the handle is a name given to the NXT link object and is used in the functions as an argument so that matlab knows which NXT it is communicating with

will set the default NXT such that if a command is given and no NXT handle is specified then it will default to your NXT - this saves lots of time debugging when you have forgot to include the handle argument. In this case if a default NXT has been set then all subsequent loines of code which require a handle input can have that input ignored.

D.1.2 Connection Code

Code = 'XXXXXXXXXXX'
Handle = COM_ OpenNXTEx('USB',code) if using USB, 'BT' if bluetooth
COM_ SetDefaultNXT(Handle)

D.1.3 Setting Up Sensors

It is a good idea to open the sensors you wish to use in an initialisation section to ensure that they provide correct readings sensors are initialised as follows where the handle argument is optional - if only using one nxt and have set default nxt as in D.1.1 just call the code D.1.4

D.1.4 Sensor Opening Code

OpenLight(SENSOR,'Mode',handle)
where mode is 'ACTIVE' If the internal red light source is to be turned on or 'INACTIVE'if off.
OpenSound(SENSOR,handle)
OpenSwitch(SENSOR,handle)
OpenUltrasound(SENSOR,mode,handle)
where mode is either blank - can take distance reading or 'snapshot' which can make the device operate as an ultrasound scanner. These two modes preclude the use of each others operating commands and so the mode must be specified correctly in initialisation to prevent errors occurring. by recording several echoes for one output port on request.

D.2 Retrieving Data

D.2.1 Light Sensors

light sensors can be read by executing the command X = GetLight(Port,handle) where handle may be ignored if a default NXT handle has been set and port specifies which sensor it is you want to use. If using the light sensor to detect presence of an onject it is best to use a differential threshold to negate the effects of ambient light levels. This means taking an on and off reading and then taking the difference of the two readings: ambient light changes will affect both equally and so will be removed by the difference operation. This is acheievd by:
OpenLight(SensorPort,'Active'),handle)
Val_ on = GetLight(Port,handle)
OpenLight(SensorPort,'INACTIVE',handle)
val_ off = GetLight(Port,handle)
diffval=on-off;
Then a Threshold can be set on the diffval variable. Light History can be easily logged by declaring an empty

matrix as well as a tic command at the start of code and for each reading use the following command
LightHistory = [LightHistory;toc,diffval];
This generates a two column matrix that can be plotted as a graph to determine where an appropriate threshold value can be set.

D.2.2 Touch Sensors

Read Data from Touch sensors by using the command data = GetSwitch(sensorport,handle). This will return a value of 1 if the sensor is pressed and 0 if it is not pressed. Be aware that if the sensor has not been opened then it will always return a value of 1.

D.2.3 Ultrasound Sensors

Ultrasound sensors may be operated in one of two modes. The first mode is distance based - chosen by not entering a mode when opening the sensor. In this mode the sensor emits an ultrasound pulse and then measures the time to the echo and thus calculates the distance away from the sensor the object which caused the echo is. This data may be gathered by performing
distdata=GetUltrasonic(port, handle)
The data may then be processed or logged as above.

The second way in which the ultrasound may be operate is in snapshot mode - much like a medical ultrasound unit. This operation has two stages- the emitting stage and the data presentation stage. The emitting stage requires the command
USMakeSnapshot(port, handle)
which causes the emission of a single pulse. Up to eight echoes are then recorded in the sensors memory. These echoes then need to be retrieved for processing
echos = USGetSnapshotResults(port, handle)
the echoes are sorted in time order, and so should represent increasing distances. Unfortunately the intensity of echo data is not recorded, somewhat limiting the usefulness of this command. Also the sensor must be opened in this mode for these two commands to work.

D.3 Motor Commands

D.3.1 Setting Up Motor Commands

The NXT Motor may be operated directly using the direct motor commands or a series of predfined commands may be set up and then called when required.These allow far more precise operations to be performed include fixed angle rotations etc and so have been found to be of greater use. The first step is to create the motor commands that you will use. These are given by:
commandname = NXTMotor('Motorport','Power',(val),'Tacholimit',(val),'ActionAtTacholimit',(action),'Smoothstart',(true or false),'SpeedRegulation',(true or false))

Power specifies the value of the motors speed/torque. Usually speed regulation is on and hence pwoer specifies the rotational speed on a scale of -100 to 100 with the sign specifying roation direction.

Tacholimit - this value in degrees is the angle which the motor will attempt to rotate by, assuming an initial position of 0. If this is set to zero or not specified then the motor will run continously until stopped by a motor stop command.

For the action At Tacholimit Property the following are available In COAST mode, the motor(s) will simply be turned of when the TachoLimit is reached, leading to free movement until slowly stopping (called coasting). The TachoLimit won't be met, the motor(s) move way too far (overshooting), depending on their angular momentum.

Use BRAKE mode (default) to let the motor(s) automatically slow down nice and smoothly shortly before the TachoLimit. This leads to a very high precision, usually the TachoLimit is met within +/- 1 degree (depending on the motor load and speed of course). After this braking, power to the motor(s) is turned off when they are at rest.

HOLDBRAKE is similar to BRAKE, but in this case, the active brake of the motors stays enabled (careful, this consumes a lot of battery power), causing the motor(s) to actively keep holding their position. SMoothstart allows a smooth roll on rather than a step input to allow less transient responses and hence better angle searching

D.3.2 Running Motor Commands

In order to run a particualr motor command the following code should be used
commandname.SendToNXT(handle)
This will execute the command. The motors themselves will react badly to commands being issued before teh previous command sent to that motor has completed (commands sent sequentially to different motors will be accepted) and will fail to implement the second command. The NXT will beep to signal that this situation has occurred.

Whilst a command is running the command
commandname.ReadFromNXT(handle)
will read back the data from the motor. This includes the current speed of the motor, an .IsRunning flag and the current angle. This can be useful if checking whether a command is actually being implemented.

D.3.3 Motor Waiting Commands

commandname.Waitfor(0,handle)
will wait for the command named command name to be completed before issuing the next command. It is advisable not to rely on this for long commands when sensor readings require periodic updating as the program will wait at such commands until the wait for criteria has expired. Providing a state machine approach is used whereby new commands are issued in each state and the state advances when the command has finished a wait for command before each motor command will not waste much time and will be reliable in avoiding commands failing to execute.

D.4 Shutdown

D.4.1 Shutdown Instructions

The Following section ensures a safe shut-down by ensuring that all sensors are turned off and that the (NXT) link is closed; as well as displaying a friendly message to tell you this has happened. Calling these commands ensures that if you re-run your program then the NXT is ready to receive commands- not using the close command will cause the Open NXT command to return an error telling you that the link is already open. It may be good to have a script with these commands in as a shut-down script that can be called in case things go wrong as it will instantly shutdown the unit - preventing damage.

D.4.2 Shutdown Code

```
StopMotor('all' , false);
disp('stopped motors')
CloseSensor(SENSOR_ 1)
CloseSensor(SENSOR_ 2)
CloseSensor(SENSOR_ 3)
CloseSensor(SENSOR_ 4)
disp('Closed All Sensors')
COM_ CloseNXT('all')
disp('Closed NXT Connection')
```

D.5 Common Errors

The following errors are very common when using the NXT program and so this section was written to help with debugging these errors.

1)NXT gives error message whilst opening
- Have all NXT links been closed (COM_ CloseNXT('all'))
- IS USB cable connected
- Does NXT Have MotorControl Program On It
-If all Else Fails Restart NXT

2)NXT Beeps Whilst In Operation
- Usually the result of two commands being issued to the same motor simultaneously i.e. motor is told to turn to one rotation before completing the last rotation.
- In which case use the Waitfor Command to wait until one command is completely executed.

- If a motor is simply 'on' or 'off' then with a zero tacholimit this problem is avoided - or if the exact angle is irrelevant and a fixed run time is required try using a zero tacho limit and flags i.e.

Turn Motor On
time on = toc
while toc - time on <time to run
pause 0.1 to save processor power
end while
turn motor off

3) NXT gives warning message

$$CannotRunMotorControlOnNXT$$

By checking the NXT menu :
My Files -Software Files
a program called Motor Control 22 should be present. If it is not then turn off the NXT, connect to the pc by usb cable and turn it on again. The PC has a rules file which detects the arrival of the device and attempts to update the motor control 22 file. Sometimes this does not work and instead the old motor control file is deleted accidentally and no new one imported. This results in the file not being present. It is always good to check if the motor control 22 program has been added correctly when using department network machines - if you have this problem repetitively whilst using your own computer. You may have a dodgy NXT or just need to add the program. Executing the download from the RWTH tool-kit should add the file back toy your device when it is necessary to do so. See the Readme in the tool-kit for details on how to do this.

E Outreach Documentation

The purpose of the Lego model of a production line currently is to assist in the teaching of control and operations management engineering to PhD students. It was noted that the educational opportunities presented by the line went far beyond the scope of its current use. Particularly, the Dyson foundation encourages projects which are interesting, involve innovative problem solving and have the potential to inspire younger students to take up careers in education. To further this aim conceptual design process was pursued allowing more education opportunities to be studied.

Firstly based on personal experiences of both team members during the first years of the degree a number of opportunities for the line to be deployed in undergraduate teaching were noted.

- Could be used to demonstrate 1B MATLAB, particular interfacing matlab with external systems.
- Could be deployed to inspire part 1A students during the Lego Modelling Project
- Linked to other similar control systems demonstrated during the part IB control systems lecture series, as a large scale example of a real control system problem.
- Could be deployed as a physical example in 3E10 Operations management module to accompany video demonstrations
- May find uses in the IfM and MET courses in a similar manner.

Secondly a Range of Outreach experiments were considered which would inspire younger students to pursue careers in engineering. These were broadly designed to teach a specific engineering concept in a simple to understand manner, using the highly interactive line, and known interest of Lego to make the activities more fun and interesting.

For children of about 10 years a simple experiment could be run, allowing user participation which changed the relative speeds of various sections of the line, under a competition to get the highest throughput. Children of this age could be taught about the advantages of modelling a situation in engineering in order to test changes to a system without affecting the real world adversely.

For older students a range of possible experiments allowing interest in either mechanical or software aspects of engineering to be explored, as well as how multidisciplinary projects are conducted. For those interested in mechanical design an experiment introducing the idea of conceptual design could be deployed. In this experiment a number of concepts of a mechanical system, such as a section of conveyor belt with different drive systems (gear box ratios, belt driven systems, etc.) or a different splitter arm system could be evaluated theoretically or even designed, and then tested in practice to allow ideas to be demonstrated and concepts to be evaluated. Students could interact with dummy sections of the line to show how what they design must integrate with the available system if it to be successful, and also that software must be designed to control the system showing the multidisciplinary design.

For students interested in software design an experiment involving developing and coding a simple algorithm to control one unit using MATLAB was investigated. Students would learn how to construct an algorithm for performing tasks such as the pusher sequence by performing the task by manually typing in commands at the prompt, then developing this into a script with some automation. This may integrate well with the pusher design tasks for the mechanical students, increasing the awareness of both multidisciplinary design and algorithm design and coding.

Finally for students already considering a career in science or engineering activities exploring the area of system management could be used, illustrating a less well known consideration of engineering. This could be done by performing simple experiments showcasing the failure surface, e.g. by getting them to use simple mathematical analysis to get a theoretical failure surface, and then generating a few data points and their own surface to show how they should be guided by the theory to manage the operations of the line and increase throughput.

These experiments should fulfil the requirements of the Dyson foundation towards the award of their bursary. If the bursary is awarded some or all of these experiments and outreach materials associated with them should be produced.