Online Planning to Control a Packaging Infeed System

Minh Do and **Lawrence Lee** and **Rong Zhou**
Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, CA 94304
`minh.do`, `lawrence.lee`, `rzhou` at `parc.com`

Abstract

After successfully developing a model-based online planner for the multi-engine hyper-modular prototype printer at PARC, we began investigating other applications with similar characteristics. One such application is an automated packaging line for food and consumer packaged goods, specifically the infeed system, where products arrive continuously from the end of the production line and need to be arranged into a specific configuration for downstream primary and secondary packaging machines. In collaboration with a domain expert from the packaging industry, we developed an innovative design for a reconfigurable parallel infeed system using a matrix of interchangeable smart belts. We also developed an online model-based planner that can control this type of infeed system through simulation in both nominal planning and when runtime failures occur.

1 Introduction

The Tightly Integrated Production Printer (TIPP) project at PARC [Ruml *et al.*, 2005; Do *et al.*, 2008] shows that general purpose online planning algorithms can be used to successfully control a complex production manufacturing system. Like other general purpose planners, our model-based planner can control various printer configurations, as long as they can be modeled in a PDDL-like language. The plans are proven to be correct and the planner is guaranteed to find a plan (guaranteed to be optimal under certain assumptions) if one exists. Our planner is also very flexible with regard to offline reconfiguration (changing the machine/manufacturing-plant when not running) and online reconfiguration (routing around component failures/servicing and take into account repairs in real-time). These advantages are refreshing when compared to the traditional approach in many industries of customizing software for each design, which can lead to a very long development time. The Embedded Reasoning Area (ERA) at PARC has been investigating new applications that share characteristics with TIPP so that we can apply our automated planning technology (more specifically, the *Plantrol* concept that combines *planning* and low-level machine *control*). In this paper, we describe our recent effort in one such application: high-speed packaging machines. Specifically, we address the problem of controlling the infeed system of the flow-wrapper machines.

In the food industry, an automated packaging line normally consists of several main components:

- *The conveyor from the production line:* The items that need to be packaged come out from the end of a production line or a storage system such as a freezer. When the items are transported on the conveyor, they can often shift out of alignment, both transversely in rows and inline in lanes.
- *The infeed system:* The items are then transferred into an infeed system that can collate them into a well-aligned pattern that is synchronized with a downstream packaging machine such as a flow wrapper.
- *The packaging line:* The flow wrapper wraps the items in a specified configuration to create the packaged product that can be transferred to cartoning, case-packing, and other secondary and tertiary packaging machines to complete the packaging process.

Depending on how the output of the production machine and the input of the flow wrapper are specified, there can be different designs for the infeed system. The typical design in the food industry for extruded food products is a *cross-feeding* system in which the products are fed one row at a time using a 90-degree angle transfer into a single lane smart-belt driven infeed to the wrapper. There are several problems with the existing cross-feeding method: (1) Cross feeders tend to jam when products are not aligned well in rows for right-angle transfer; (2) Cross feeders do not have active control over the products such that temporary increases in volume from the production line are handled by keeping an extra cross feeder and packaging line connected to the conveyor from the production line as spare or backup capacity; (3) Cross feeders are not easily reconfigurable to handle different wrapping configurations; (4) Cross feeders have a large footprint necessary for the right-angle transfer, which takes up valuable floor space.

To overcome these drawbacks, we have developed a new infeed system design that assimilates TIPP's hypermodular structure. More specifically, a new infeed system is made up of a matrix of interchangeable smart belt modules that can be controlled individually. Figure 1 shows one example of the

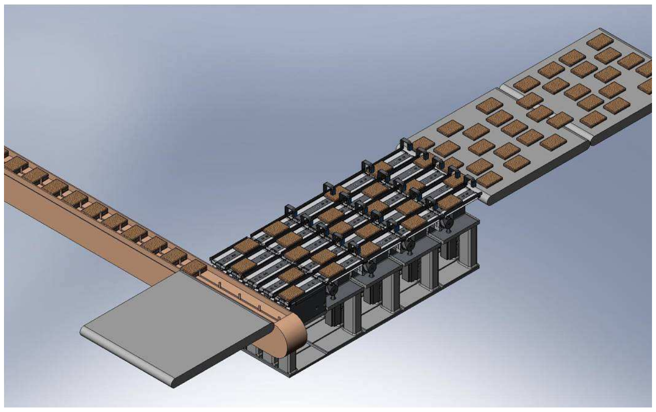
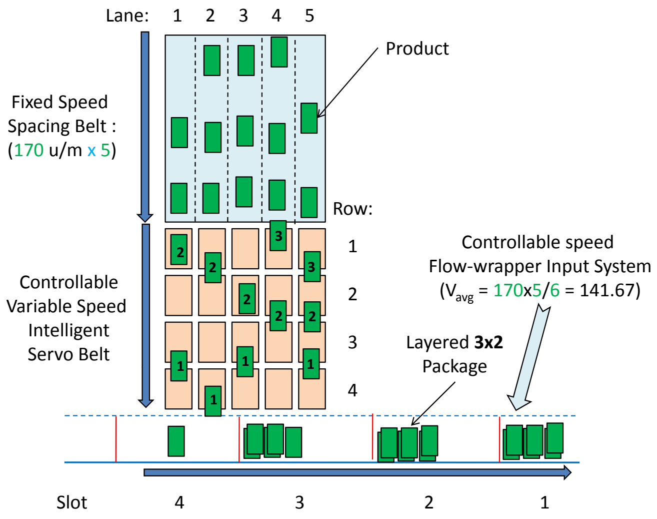


Figure 1: An exemplary modular smart-belt infeed system



Lane: 1 2 3 4 5
Product
Fixed Speed Spacing Belt : (170 u/m x 5)
Row:
1
2
3
4
Controllable Variable Speed Intelligent Servo Belt
Controllable speed Flow-wrapper Input System (V_{avg} = 170x5/6 = 141.67)
Layered **3x2** Package
Slot 4 3 2 1

Figure 2: More detailed description of a modular smart-belt infeed system

design. In this figure, products come out of the production line or storage system from the right in an unordered manner and enter the infeed system, which in this case is made up of a 5×5 matrix of smart belt modules. By controlling the speed and acceleration of each smart belt when a given product is on it, the matrix is able to collate and synchronize multiple parallel lanes of output to achieve a desired configuration for the flow wrapper input on the left. For example, we can have three stacks of products with two in each stack to create a wrapped package in each segment on the wrapper-input line. Some advantages of the new system design are: (1) higher flexibility to create many different modular versions of the infeed system because a new system is defined simply by deciding the dimensions of the matrix and the specification of the smart belt modules; (2) higher operational efficiency because any lane can be taken offline (for cleaning or service) while other lanes continue operating instead of requiring a complete shutdown of the packaging line; (3) higher reconfigurability because products can be collated for any possible wrapping configuration without needing additional programming; (4) smaller footprint due to an inline design that not only reduces the space needed for a right-angle transfer but also eliminates the need to keep a spare infeed and packaging line. Upon visiting large trade-shows in this domain and discussions with domain experts, we believe that our design is novel and there is no current infeed system using this design.

One of the keys to the success of this new design is the ability to control the matrix composed of individual modules. This is a challenging problem given that the incoming rate of products can be very high, normally several hundred per minute per lane and the products are not well-aligned in rows. For this task, we have adapted our fast online planner used in the TIPP project to this domain (from now on, we will refer to it as a Plantrol planner as it is the formal name for PARC's online planning efforts after the TIPP project). Our preliminary results show in simulation that our online planner can indeed successfully control various modular infeed system designs. We are also building a hardware prototype in the ERA's Lab to test our software with a real physical system.

The rest of the paper is organized as follows: we continue in Section 2 with a more detailed description of our initial system design and how controlling this infeed system can be modeled as an online planning problem. We then describe in Section 3 how we adapt our current online planner in our *Plantrol* concept to this application. We describe some preliminary results in Section 4 and finish the paper with a conclusion and outline of our future work in Section 5.

2 Intelligent Infeed System Design and the Control Problem

Figure 1 shows one example of the infeed system using our modular design. A more complex system may have several infeeds of the same design sharing the input from the production machine. Other designs may have different end-effectors that can collate specific types and quantities of products for different packaging configurations. Even though we have simulated several different designs using our online planner, to illustrate our approach, we will use the design outlined in Figure 1 as the leading example throughout the rest of the paper.

Figure 2 shows a more detailed view of our running example system[1]. Products enter the infeed system from a fixed-speed *spacing belt*. While products are well aligned in different lanes and the average incoming rate for each lane is known (170 units/minute/lane in our example), the exact timing when each product starts to enter the smart belt system is unknown. In the result section, we will describe how we simulate this uncertainty. As shown in Figure 1, there are photosensors placed at the entry point of each smart-belt module

[1]One difference between Figure 1 and Figure 2 is that there are 5 rows of belt modules in Figure 1 while there are only 4 rows in Figure 2. This is because the last row operates at a fixed speed and is not controllable (an alert reader may realize that there are no overhead sensors for the last row in Figure 1) so that the moving product can be transferred to the wrapper-input at a constant speed for synchronization with the downstream flow wrapper. We omit this final row from Figure 2 and from all the subsequent figures in this paper.

to recognize the time at which a product enters each module. For the first row of modules (just next to the spacing belt), the sensor feedback provides the timing information when new products enter the system. Sensors on subsequent modules provide feedback on the differences between the scheduled timing (as ordered by the high-level planner) and the actual arriving time. Any difference will require corrections by the low-level controllers of the subsequent modules.

The flow-wrapper input system, which is at the bottom of Figure 2, consists of continuously arriving empty *segments* on a belt. Even though the speed of this input system does not change in normal operation, it is controllable (we will show later in this section how we need to adjust the flow-wrapper input speed when failures occur). The flow wrapper is preconfigured to wrap packages as specified by the operator of the infeed system. In our example, each package is a stack of 3×2 products (three stacks of two units each). Each segment needs to either be filled up to the specification (i.e., in a 3×2 stack) or left empty. Given that each lane only aligns with a given slot in each segment (there are 6 slots in a 3×2 package configuration) at any brief moment in time, the planner needs to speedup or slow down different product units by adjusting the speed of each of the four belt modules that each product will pass through so that it will arrive at the wrapper-input line at the exact moment when that intended segment/slot is aligned with that lane. One feature that is not clear from the figures is the ability to skip a given product by making it "bypass" the wrapper input. This is done by mechanically extending the last belt module over the top of the wrapper input. For example, if there is no physically possible way (given the acceleration and deceleration limits of the servo motor of each belt module and the coefficient of friction for the product on the belt) that product #1 on lane 5 can hit an empty slot in segment 3, 4, or later when that slot aligns with lane 5 then we need to extend the module on row 4 of lane 5 so that product #1 on lane 5 can bypass the wrapper input.

The main objective function is to minimize the total number of empty segments. A summary of the constraints that the planner needs to obey when planning/scheduling routes for different products are:

- There is a limit on the maximum acceleration or deceleration that can be achieved on each module. Thus, there are limits to how much each product can speedup or slowdown on each module.
- For more precise control and tracking, multiple products cannot be on the same module at any given time. Thus, temporal constraints are needed to ensure that the leading edge of each product can not reach the next belt module before the trailing edge of the preceding product on the same lane completely leaves that belt module.
- As mentioned earlier, the leading edge of a given product should leave the last module in each lane (i.e., 4^{th} row) at exactly the same time the intended slot of a given segment on the wrapper-input aligns with the lane that product is on.
- When the planner/scheduler does not issue any speed-changing actions, all products keep moving following the current speed profile[2] of the modules on which they are being transported. Thus, if the planner takes too much time to find a plan, the product may have moved along too far to be able to position properly (note the rate of several hundred products per minute for each lane).

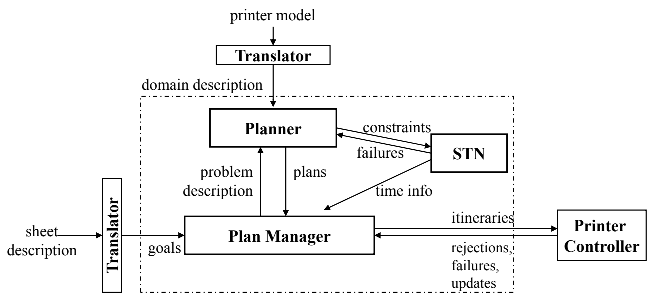


Figure 3: The system architecture of the TIPP planner, with the core planning system indicated by the dashed box.

In summary, the planning/scheduling problem is to control the movement of known products through the smart belt modules so that collectively they are synchronized to fill up every segment of the wrapper input system; while obeying all constraints as listed above. Given that the overall path of each product is well-defined and the only choice is whether or not to "bypass" a product, this problem has more of a scheduling flavor than planning.

3 Adapting the Plantrol Planner

Previously, the Embedded Reasoning Area at PARC applied automated planning techniques to the control of production printing equipment (the TIPP project). Fundamentally, a modular printer resembles any manufacturing plant, with raw materials (blank sheets) entering as the plant's input, getting routed through different machines that can change the properties of the materials, and then collecting final products (printed sheets) at the output. We believe this framework can be applied in a wide range of on-line continual decision-making settings. Specifically, the packaging machine control problem described in this paper is one of several applications that we are investigating to utilize the Plantrol concept developed within the TIPP project.

Figure 3 shows the core architecture of the planner, as used in the TIPP domain, and how it communicates with the physical controller. The major components are outlined below:

Temporal and Plan Management: TIPP is a rich temporal domain with real-time constraints between wall-clock time and the plans for individual sheets, between plans for different sheets, and between the planner and the printer controller. Thus, fast temporal constraint propagation, consistency checking, and querying are extremely important. We maintain the temporal consistency using a Simple Temporal

[2]The speed profile of a given module basically specifies the details of the change starting from the current wall-clock time (e.g., v_1 at current time t_c, change to v_2 at $t_2 > t_c$, then change to v_3 at $t_3 > t_2$).

Network (STN) [Dechter *et al.*, 1991], represented by the box named *STN* in Figure 3.

Planning for Individual Goal: A^* search is used to find the optimal plan for the current goal, in the context of plans for all previous goals. The state contains information both about the current and previous plans. More specifically, the state is a 3-tuple $\langle Literals, STN, R\rangle$ representing: (1) the logical state of the current plan; (2) all known time points and the current constraints among them in STN; and (3) the set of current resource allocations R, representing the commitments made to plans of previous goals and the partial plan of the current goal.

Heuristic Estimation: The overall objective is to minimize the overall wall-clock goal achievement time (planning time + makespan). To estimate the duration required to achieve a given set of goals G from the initial state, we build a structure similar to the Temporal GraphPlan system [Smith and Weld, 1999], improved with resource contention reasoning.

More details are described in [Ruml *et al.*, 2005; Do *et al.*, 2008]. For the rest of this section, we outline how we have adapted this planner for the packaging domain.

3.1 Nominal Planning

For the nominal planning case, we keep the basic structure of the planner as shown in Figure 3. However, adaptations are made as follows:

Input: We use a modeling language that is simpler than the more expressive PDDL-like language to model the smart-belt matrix. The modeling language specifies the key characteristics of different module types such as length, maximum and minimum speeds, maximum acceleration and deceleration. The problem instance part then specifies the size of the matrix and the module type at each location in the matrix. The online product description contains information about time instances at which incoming products enter the matrix at different lanes.

Planning State: Figure 4 shows a stage in the continual planning process. At this stage, we have finished planning for segments 1-4 and are planning to fill up the next arriving segment #5. At any given time, the planner knows about all products that are currently on the smart-belt matrix (i.e., all numbered products with background in black or white). Among those, all white-background products are already planned to be put in some slots in segments #3 and #4 and so are not available to be candidates to be put in segment #5 that is currently being planned for. Our planning problem is to find a subset of 6 candidate products such that they can fill out all slots in the wrapper-input segment #5, and satisfying all the constraints listed in the previous section. We will set it up as a planning problem using search. As such, we need to define the root node, the expansion function governing how children states are generated, and the goal satisfying conditions to terminate the search.

Root State: Let t_c be the time at which we start the planning process and t_p is the expected duration to find a successful plan. Thus, the plans are expected to be found and can start being executed at $t = t_c + t_p$. During the planning time,

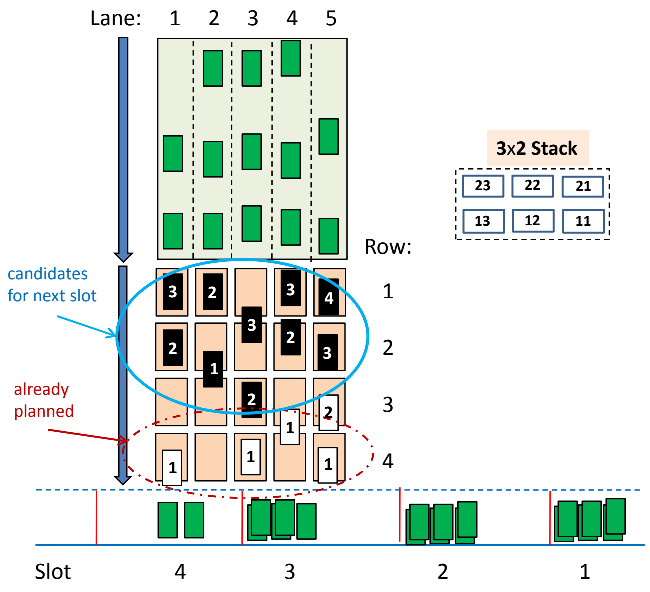


Figure 4: A planning stage example.

all products p currently on the infeed system are expected to move according to the current speed profiles of all modules to the new locations l_p on the matrix[3]. The initial planning state represents the root search node contains information on: (1) all candidate products and their locations at $t_c + t_p$; (2) all slots 11-23 (refers to the right side of Figure 4) are empty; (3) STN represents all the temporal constraints (more details on the STN are discussed later.)

Expansion: There are two choices that we considered for the expansion function: (1) select a given empty slot in the wrapper input segment and branch over the products that can fill in that slot; (2) select a product and branch over which slot that it can fill in (or if we want/have-to make the product bypassing the wrapper input). We use the second approach and schedule one product at a time according to a product ordering function. Specifically, with regard to all the constraints listed in the previous sections (e.g., maximum one product on a single smart-belt module, speed/acceleration/deceleration limits), let $d_1 = [t_s, t_e]$ be the time window bounding the time instance at which a given product p can arrive at the wrapper-input, and t_1, t_2, t_3 be the time instances at which slots 11 (and 21), 12 (and 22), 13 (and 23) align with the lane at which p was on. We select the next product to branch on based on the orderings: (i) between products in the same lane: the earlier one is selected first (e.g., on lane 1 in Figure 4, product #2 is selected before #3); (ii) between different lanes: smaller number of candidate slots (i.e., smaller number of cases in which $t_s \leq t_i \leq t_e$ with $1 \leq i \leq 3$). The second criterion is inspired by CSP's most-constrained-variable-first ordering heuristic. For each $t_i \in d_i$, we create one child plan-state with product p

[3]Note that we can never stop moving products on the smart-belt matrix to give more planning time because there is a continuously arriving stream of products to each lane that will jam the system.

assigned the slot $1i$ or $2i$ (if they are not already assigned to another product). We also create one additional child plan-state where p bypasses the wrapper-input. Nodes are put in the A^* open list implemented as a priority queue ordering by: (1) smaller number of bypass product; (2) break-tie over the higher number of filled slots.

Goal Satisfaction & Final Plan/Schedule: A goal state is reached when all slots in a given segment are filled. When this happens, the search is terminated and the plan is returned. At this point, the final destination for the planned products are decided: a particular slot in the arriving segment or bypassing the wrapper-input. However, the actual duration each product spends on each belt module is not decided yet and thus some scheduling decisions should be made. Simply speaking, we choose to run each planned product as quickly as allowed by the various temporal constraints on the earlier modules and as slowly as possible in the later modules. For example, the product #1 in lane 2 will run as quickly as allowed through row 3 and slow down in row 4. The purpose is to increase the inter-product gap and thus allow more control for the subsequent unscheduled products.

Temporal Constraints: The temporal management using Simple Temporal Network (STN) manages various time-points and constraints between them. Time points represent the instances at which each product enters or leaves a given smart-belt module. Thus, in our ongoing example, for each product p we create 8 time-points representing the trajectory of p. There are temporal constraints (as outlined near the end of Section 2) between: (1) consecutive time-points on a trajectory of each product; (2) time-points at which consecutive products enter or leave a given smart-belt module to ensure that only a single product will be on that belt module at any time. For example, if a given product p enters a module s running at speed v at time point t_1, given that we know the minimum speed v_m (normally it's not zero), maximum speed, and the maximum acceleration/deceleration values a_1, a_2 of s, we can calculate the upper and lower-bound values u, l on the duration p spends going through s. We then add the constraint $l \leq t_2 - t_1 \leq u$ with t_2 being the time-point when p leaves s. The temporal network is continuously monitored and cleaned before each planning stage. The cleaning process removes any time-point that occurs before the wall-clock time at which the cleaning process is run. If there are constraints between a future time-point t_1 with the removed time point t_2, then re replace it with a constraint between t_1 and the universal start time point 0, which represents the wall-clock time at which the planning process started.

Pruning: There are several rules to prune potential children nodes and reduce the branching factor. For example, for a given product p from the first lane (e.g., #2 on lane 1 in Figure 4), then the candidate slot can only be 11, 12, or 13 because to be able to get into 21, 22, or 23, it needs to have some other product residing on 11, 12, or 13 at the time of arrival for p. Similarly, products from the last lane (lane 5) can not be allocated to lower slots 11, 12, or 13 and the candidate slots can only be the upper ones 21, 22, or 23. Another rule narrows the candidate set by limiting the number of products from a single lane by the number of horizontal slots. In our example, it means no more than 3 products from a single lane can fill in a single segment. Two products from a single lane can not also be stacked on top of each other. Two products stacked on top of each other should also arrive in the order that is physically possible: lower slot should be filled before the top slot. For example, if in one search branch the product #2 on lane 3 (product p_1) is planned to fill in a slot 22. Then we know that slot 12 should be filled by some product from lane 1 or lane 2. Thus, if product #2 from lane 4 (product p_2) is planned next, we will not consider slot 12 because it's not possible to fill in p_2 before p_1. Depending on the particular configuration in each search branch, rules as outlined above can significantly reduce the branching factor and speedup search.

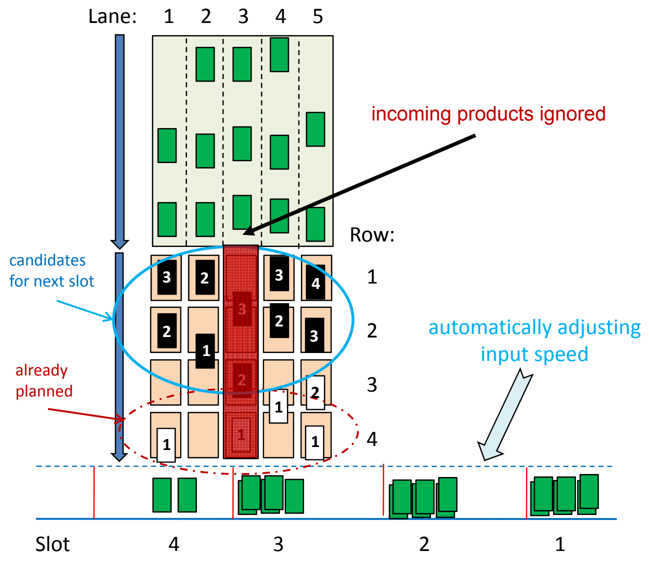


Figure 5: An example of failure.

As shown later in the empirical evaluation, our planning search settings as outlined above allow us to find a plan/schedule very quickly and enable us to keep up with packaging lines with very high throughput rates.

3.2 Failure Handling

Any smart-belt module can fail. When this happens, physically the module can be controlled to divert products to a storage buffer or route back into the start of the infeed system. From the logical point of view, the entire lane is taken offline and thus the planner will only consider products arriving at the remaining operational lanes. Given that the overall incoming rate decreases, the wrapper-input speed also needs to be reduced so that the input and output throughput match and we can avoid having empty segments. Figure 5 shows an example where the third lane is taken offline due to failure in one of its modules.

When failure happens, products on the infeed system from both *already planned* and *candidates for next segment* groups are affected. This potentially leads to: (1) partially filled segments; and (2) not enough candidates to plan for the next empty segment. In our running example, if product #1 in the failed lane 3 was scheduled to fill in segment #4, then if we don't find a replacement, segment #4 will end up as a partially-filled segment. On the other hand, the removal of products #2 and #3 on lane 5 may lead to not having enough products to fill in the upcoming segment #5 if we do not slow down the speed of the wrapper-input. Following are steps taken in our planner to cope with the failures:

Adjust the wrapper-input speed: Given that the total incoming rate of product decreases as lanes fail, the outgoing rate of product into the wrapper needs to be adjusted accordingly to avoid empty segments. The question is when and how we adjust the wrapper-input speed to accommodate the throughput change. As shown in Figure 5, four already planned products (in white background) are scheduled to fill in segments #3 and #4. If we adjust the output speed right when the failure occurs, then those products need to be replanned/scheduled. We do not use this approach because as seen in the running example, many of the already planned products are close to the wrapper-input and it is too late to reschedule them. Instead, we:

- If the failure happens at wall-clock time t_f, we identify the wall-clock time $t \geq t_f$ at which the last planned product get into the intended segment.
- Let n be the total number of operational lanes before the failure, $m < n$ be the number of operational lanes after the failure (normally $m = n - 1$), and s be the current speed of the wrapper-input system, we schedule to adjust at t the input speed to $s' = s \times m \div n$.
- Using the speed profile of the wrapper input such that it will go at speed s until t and continue with s' after t, we can calculate the time instance at which each slot in the unfinished segments (#3, #4, and later) align with each of the five lanes. We use that information to replan the remaining candidate products.

Replace the planned products that are removed due to failure: In our example, product #1 on lane 3 was planned but now removed[4]. We need to find another product from the remaining unplanned products to replace it. We then continue with the subsequent segments. This process is done exactly like the nominal planning case, with the only exception of the recalculated alignment time as described in the previous paragraph.

Rebalance for the incoming segments: even if we adjust the speed perfectly and find the replacement for the lost planned products, during the transition period caused by the failure, there is normally a short period where incoming and outgoing product rates do not match. This makes sense intuitively because the failure caused the loss of three products that were on the infeed system at the time of failure. Therefore, one behavior that we see with our planner in many failure cases is that it will "bypass" some products and leave one empty segment during the transition period and then fill in all subsequent segments. For example, segment #5 is left empty while it bypasses three products from the candidate product set, then fills in all segments starting from #6).

[4]For easier discussion, we assume that all products on the failed lane are lost due to the failure. In actual simulation, product #1 on lane 3 may still be valid if, for example, the module on row 2 failed.

```
(domain SmartBeltFeeder
    (:values
        (spacingBeltSpeed 400)
        (productArrivalRate 0.25)
        (productLength 100)
        (interProductGap 1)
        (servoWidth 50)
        (feederSegmentLength 150)
        (feederSlotLength 50)
        (numFeederStack 2))
    (:servotypes
      (:servo Type1
        (minLength 120)
        (maxLength 120)
        (width 50)
        (minSpeed 10)
        (maxSpeed 5000)
        (maxAccel 5000)
        (maxDecel 5000))
      (:servo Type2
        (minLength 150)
        (maxLength 250)
        (width 50)
        (minSpeed 1500)
        (maxSpeed 1500)
        (maxAccel 0)
        (maxDecel 0)))
    (:instances
        (Type1 Type1 Type1 Type1 Type2)
        (Type1 Type1 Type1 Type1 Type2)
        (Type1 Type1 Type1 Type1 Type2)
        (Type1 Type1 Type1 Type1 Type2)
        (Type1 Type1 Type1 Type1 Type2)))
```

Figure 6: Model of the running example.

Our planner operates in similar fashion when a lane is *repaired*, but some steps are done in a reversed manner. For example, the wrapper-input speed is increased instead of reduced. Given that the incoming rate will be higher than the outgoing rate during the transition period, Some extra products may also be selected to bypass the wrapper-input to rebalance the input/output throughput rate during the transition period without creating empty segments.

4 Preliminary Results

We have modeled different configurations using the modular smart-belt infeed system design described in this paper. Tested configurations include variations of the design shown in our running example and also another design in which the products need first to be collated into a bigger pack in each lane (e.g., collate into package of 30 units) before the synchronized transfer of packages from different lanes into the wrapper-input. The second type of design has been tested less than the first design because it was designed to handle one particular product while the first type has been designed as a more general case. Nevertheless, the planner's operation is basically the same.

Figure 6 shows the full model of our running example configuration (with the exception of the different product throughput). All distance numbers are in millimeter and time values are in seconds. The first part of the model file specifies values such as product arrival rate (240 products/minute/lane in this case), length, minimum product gap, and dimensions of each segment, and lane-width. The final configuration is 2 row (*numFeederStack*) and 3 columns (as decided by $feederSegnmentLength \div feederSlotLength = 3$). In this configuration, there are two types of modules: *Type1* which is variable speed and is controllable while *Type2* is a fixed speed. As mentioned in the second paragraph of Section 2, this type of module is needed for the last row before going into the wrapper-input. Specifications for each module type include length, minimum and maximum speeds, maximum acceleration and deceleration. The last part of the model file specifies the smart-belt module matrix; in this case 5×5 with four rows of controllable modules and the last with fixed-speed modules.

All values in the tested configurations have been confirmed by a packaging industry expert to be reasonable; actually, many tested designs/configurations were developed by our domain expert based on his understanding of challenges in the packaging industry.

We have tested various configurations with sizes range up to 8×6 matrix and product throughput rate from 60-240 products/minute for each lane (overall of up to $240 \times 5 = 1200$ products/minute). The final stacking configuration ranging from 2×1 to 3×3. The planner can run either in Linux or Windows machines, which do not necessarily need to be very powerful (for example, one machine that was used is a Dual Core 2.4GHz with 2GB RAM). The planner is able to handle both nominal planning and replanning for the configuration specified above and tested with simulation of a few hundred to few thousand products entering the infeed system.

For generating the incoming product stream, we have written a random test generation program that injects uncertainty into the product arrival time. For example, if the average coming rate is 240 products/min/lane then if there is no uncertainty, the arriving time of a n^{th} product in a given lane should be $t = n \times 0.25$. Our test generation program randomly injects up to 40% randomness in arriving time. Thus, the n^{th} product will arrive randomly between $(n - 0.4) \times 0.25$ and $(n + 0.4) \times 0.25$.

We also developed a visualization tool that works on-line with the planner/scheduler. It can:

- Display in real-time the trajectory of all products according to the plan/schedule found by our planner.
- Through the visualizer, we can "fail" or "repair" different lanes in real-time. When failure or repair happens, we can also see the adjusted wrapper-input speed, product bypass and transition as described in the previous section.

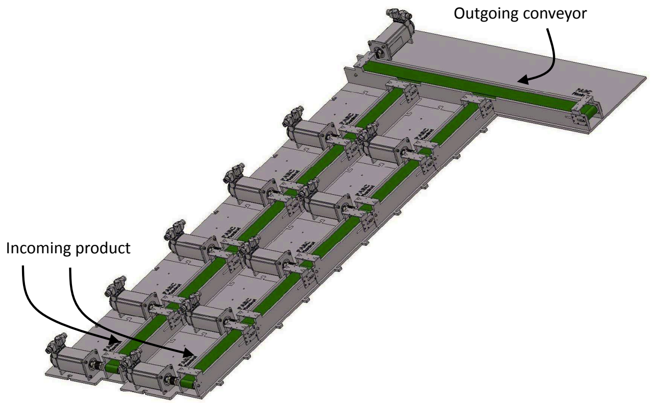


Figure 7: ERA Lab's packaging prototype.

While our simulation results look good so far: we are able to simulate various configurations and handle high throughput rates in both nominal planning and replanning, products can behave differently in a real physical system. There are issues that were not addressed in simulation such as friction, low-level control and sensor accuracy, and other integration/network communication issues. Therefore, we are currently building a two-lane modular infeed prototype, with the initial design shown in Figure 7. We hope to report our result on this prototype in the future.

5 Conclusion and Future Work

In this paper, we have described an online planning/scheduling application in the food packaging industry. Our new hardware design has several key advantages over existing systems. We have also developed a fast online planner/scheduler adapted from our previous work on the TIPP multi-engine production printer platform. The simulation results have shown that the planner can control effectively various configurations with high incoming product rates with uncertain arrival times. In the future, we hope to accomplish the same success on physical systems, with one prototype being built in our lab to prove the concept.

References

[Dechter *et al.*, 1991] Rina Dechter, Itay Meiri, and Judea Pearl. Temporal constraint networks. *Artificial Intelligence*, 49:61–95, 1991.

[Do *et al.*, 2008] Minh Do, Wheeler Ruml, and Rong Zhou. On-line planning and scheduling: An application to controlling modular printers. In *Proc. of AAAI08*, 2008.

[Ruml *et al.*, 2005] Wheeler Ruml, Minh Binh Do, and Markus Fromherz. On-line planning and scheduling for high-speed manufacturing. In *Proc. of ICAPS-05*, pages 30–39, 2005.

[Smith and Weld, 1999] David E. Smith and Daniel S. Weld. Temporal planning with mutual exclusion reasoning. In *Proc. of IJCAI-99*, pages 326–333, 1999.