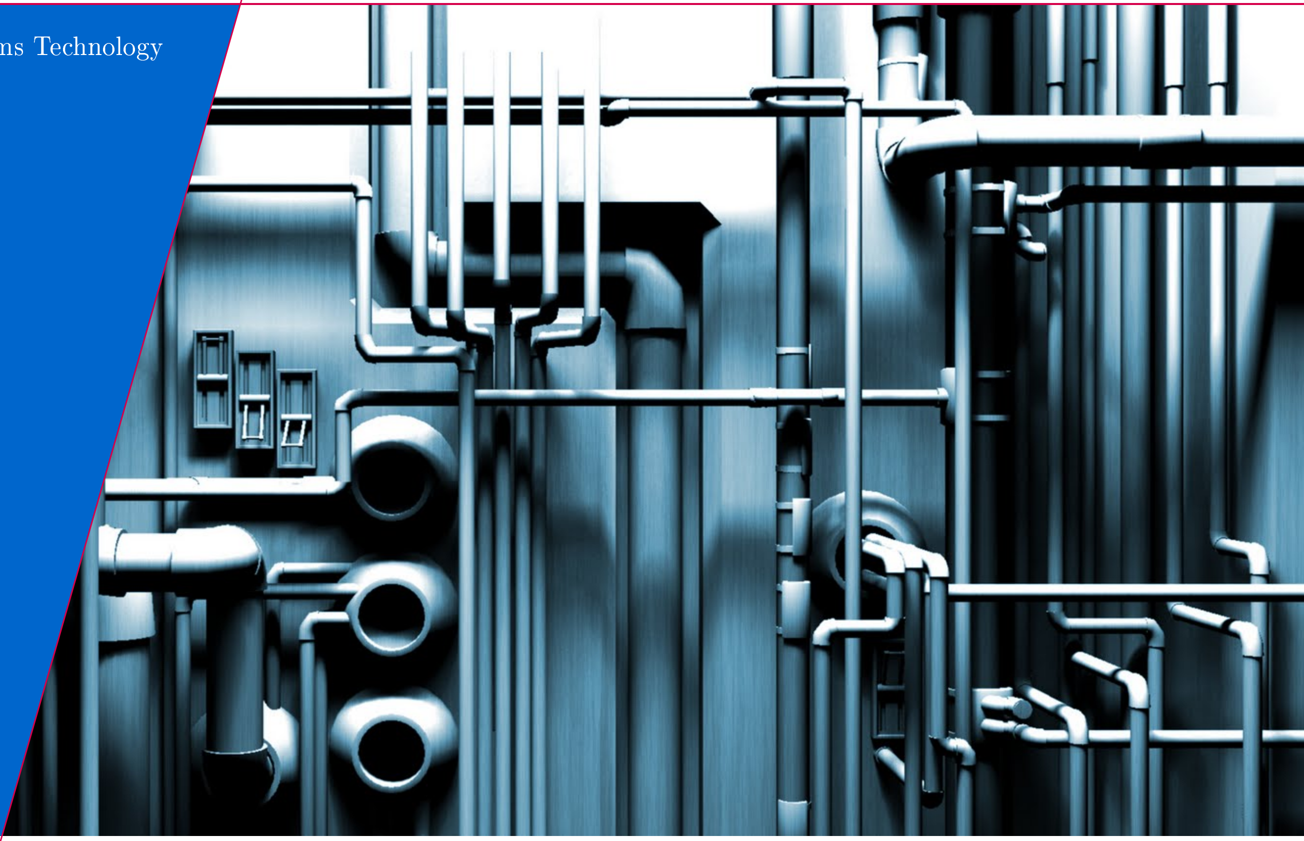
Technische Universiteit
Eindhoven
University of Technology

/ Department of
Mechanical Engineering

Control Systems Technology



Concurrent simulation of the control, scheduling and physical processes of an experimental pipeless plant using PTOLEMY

B.C.G. Vercoulen

Where innovation starts

Control Systems Technology

Concurrent simulation of the control, scheduling and physical processes of an experimental pipeless plant

using PTOLEMY

May 10, 2017

B.C.G. (Bart) Vercoulen
0747283
CST 2017.054

Supervisor: M.Sc. M. (Marina) Rantanen-Modéer
Supervisor TU/e: dr.ir. M.A. (Michel) Reniers

Contents

1 Summary

In the past years researchers and students of the Process Dynamics and Operations group at the Technical University of Dortmund developed a scaled version of a pipeless plant. A pipeless plant is a chemical production process based on batch production. This provides an alternative to the traditionally used continuous process which uses pipes to transfer the liquid products to other tanks. This pipeless plant uses Automated Guided Vehicles (AGV) to transport vessels from one process station to another. Each station performs its own process, for example filling or mixing the product in the vessel.

Development of systems such as the pipeless plant is an iterative, costly and slow process. To reduce the time span and costs of the development process, model driven development is introduced. A model can predict design flaws in early stage of development which would otherwise be encountered after implementing the system. Since the pipeless plant setup has already been build, this software implementation is analyzed first. The previous implementation of the pipeless plant software is built within a C-sharp framework, within which simulation and execution are build into one framework. For this project only the simulation parts are of interest including the physical processes of the AGV, the scheduler and the controller. The other parts included in the C-sharp framework are the camera processing, PLC management to communicate with the stations and communication to the AGV's, these parts are for implementation on the real plant.
The physical part for simulation is built in Modelica, which includes the AGV dynamics, station dynamics and arena restrictions such as the border of the field. The scheduling makes use of TAOpt, this is an optimizer for timed automata. The input to the TAOpt scheduler is a specific recipe file for TAOpt for all the containers that have to be processed. The output contains all the tasks in the optimal order which can be executed by the pipeless plant or the simulation. The controller which is developed by [Alvi, 2015] as graduation project, is implemented in Python with the use of the Casadi toolbox and is a Non Linear Model Predictive Controller with different configurations.

To develop one comprehensive model with these three aspects a suitable environment is needed. PTOLEMY II has potential to be the right environment for this. This environment is based on the java language, is developed for heterogeneous modeling, has concurrent execution of models and is compatible with Python, Matlab and other tools. It has an actor oriented design, where actors are the building blocks of a model. The actors communicate through messages via interconnected ports, resembling Simulink models.
After analyzing the previous simulation model follows the implementation of the physical part, scheduling and controller into a simulation model. First the physical AGV implementation is realized. Where the velocity and rotation speed are the input variables and the position and angle of the robot the output variables. The physical processes of the stations and containers are not included because those are considered secondary.
After the implementation of the AGV, the already developed controller is implemented to navigate the AGV to a specific location. Since the controller is developed in python it should be easy to implement it because PTOLEMY has a Python actor. But this Python actor can not be used due the fact that it is Java Python which is not compatible with the scientific toolboxes such as numpy and Casadi. Therefore a workaround has been developed to execute the actual C Python code from PTOLEMY with the use of the commandline actor and a socket communication module. An extra loop is running in a background process which starts an controller instance if it gets a new destination message. During the guidance of the AGV, the velocity and position updates are sent from the controller instance to the PTOLEMY simulation via socket communication until the destination is reached.
The scheduler has to be rebuilt because it has to schedule the tasks in real time. One task will consist of picking up the container, filling according to recipe, mixing the content and store it. The resulting scheduler consist of two queues one with new recipes and one with recipes where already one or more layers has been processed. This last queue has priority before new incoming recipes. When an AGV is in idle state, it

requests a task and starts processing that layer. When the layer is finished a new task is requested.

The final simulation model itself is working as it should, but the resulting simulation with three AGV's had one big problem. When two AGV's drive to one destination at the same time they are blocking each other because they are others' obstacle. This causes a deadlock because no AGV will reach the destination. The scheduler needs knowledge about the occupation status of the stations before sending an AGV to a destination.

2 Introduction

In the last decades system modeling has become an important part in the design process of a new system. A model represents the major parts of a system to analyze its behavior. It is impossible to model every single detail of a system in a model. That is why a lot of aspects are analyzed individually or even are assumed to be of no interest.

This project aims at investigating the structure of the pipeless plant and ultimately to build a simulation model representing some of its major aspects. The pipeless plant is a system that distributes containers, containing fluids, to processing-stations with the use of Automated Guided Vehicles (AGV). In the past years students and researchers at the Process Dynamics and Operations group at the TU Dortmund have developed this pipeless plant. A physical model in the Modelica language, a scheduling model with TAopt, and also several controllers. These models built in dedicated tools are combined into one framework which is used for simulation and execution on the real plant.

However, this aggregation of submodels is not concurrently executable and multiple tools are needed. A hypothesis of this project is that concurrent simulation can capture behaviours that are otherwise overlooked. One of the environments that is developed for this purpose is PTOLEMY II. This is a java based modeling environment which is able to combine multiple modeling formalisms. PTOLEMY is explained in section 5.

In this project the PTOLEMY environment is used to model the different aspects of the pipeless plant into one comprehensive model. This includes the physical processes of the AGV, the controller to guide the AGV's and the scheduling to direct every task. These implementations are discussed in section 6. Such a comprehensive model can give an insight in the interactions between multiple submodels in an early stage of the design process.

Concluding with the simulated experiments where the interactions between submodels are shown. This is discussed in section 7. Based on the results of the experiments conclusions and recommendations are give in section 8

3 Pipeless Plant

In chemical industry a pipeless plant is used for batch production. Containers are transported between several production stations where a process is executed. This pipeless plant transports the container with the use of Automated Guided Vehicles (AGV). Figure 3.1 shows an overview of the miniature pipeless plant as it is realized at the TU Dortmund. There are four stations, two fill stations, one mix station and a store station. An overhead camera with a fisheye lens is used for position feedback. The Programmable Logic Controller (PLC) is used to actuate the stations. This plant produces colored chalk with different layers based on the input recipes. The scheduler assigns an AGV to pick a container from the storage station, this container is placed on the AGV. This container is filled with colored liquid at the fill stations, followed by the mixing process where plaster is added and the mixture is mixed. Thereafter the container is transported to the storage station where the layer can harden. This sequence can then be repeated for another layer based on the recipe for this particular container.

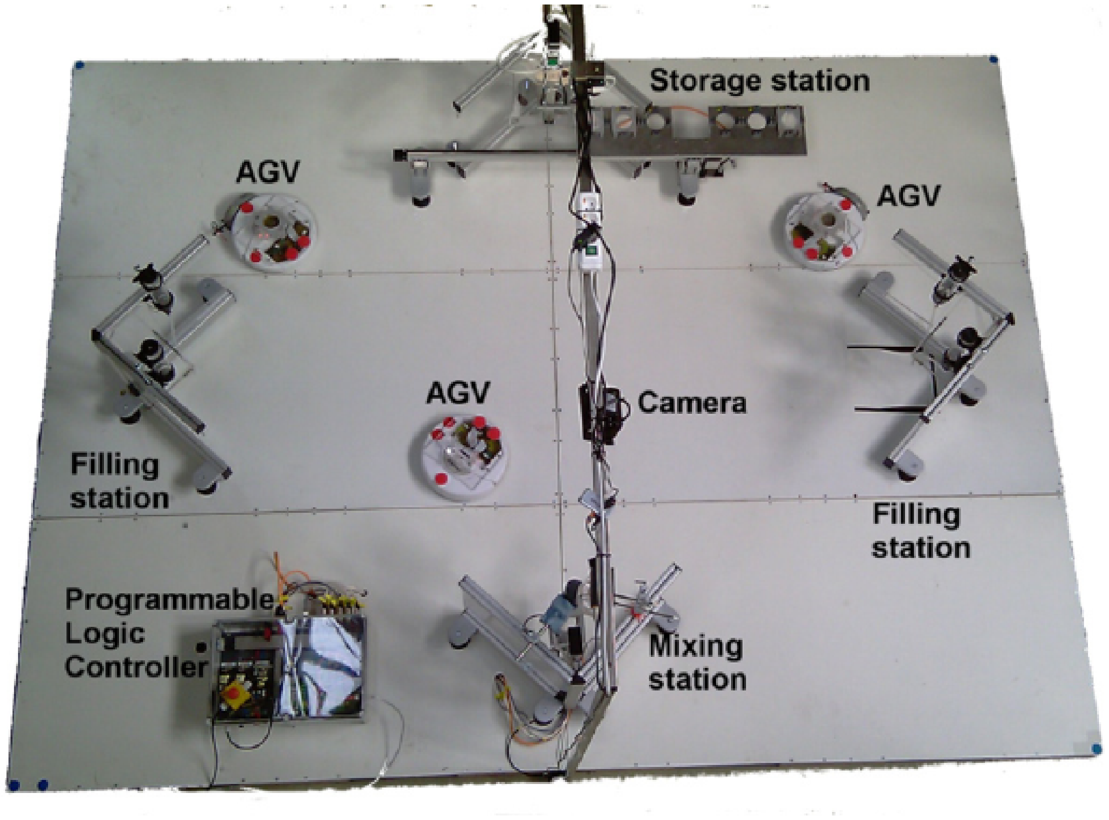
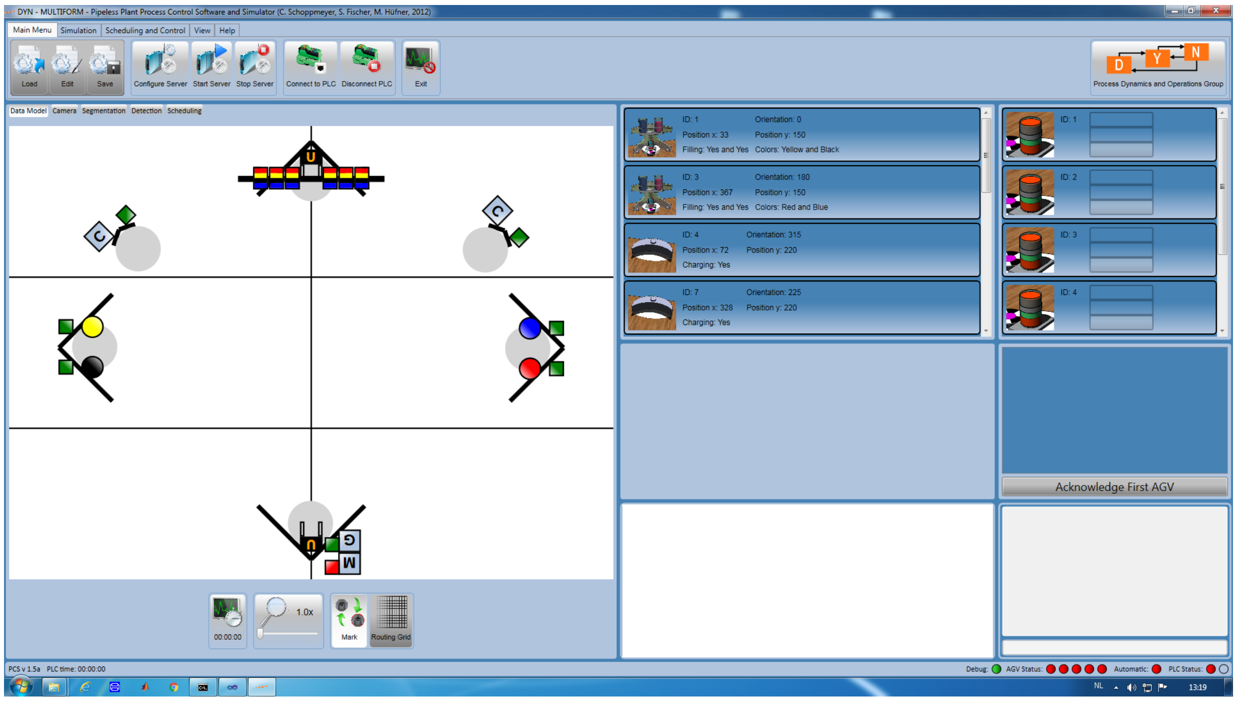


Figure 3.1: Pipeless plant overview.[Schoppmeyer et al., 2012]

3.1 Previous simulation

The already developed simulation exist of a framework built in C sharp language. This framework is custom built for this pipeless plant and connects the physical model, controller, scheduler, PLC communication to the real plant into one interface. Figure 3.2 shows the graphical interface. In the upper left corner there are buttons to execute the individual parts of the simulation. These parts are executed non-concurrently. First the scheduler has to translate the recipes into tasks, thereafter the simulation can execute these tasks.

The physical model of the AGV's and stations is modeled in Modelica. This model is executed via Dymola which can interpret the Modelica model and is connected with the C sharp framework.
The scheduler is TAOpt [Schoppmeyer et al., 2012], which is a timed automata optimizer developed at the TU Dortmund. TAOpt can interpret a recipe file which is explicitly written for this module. The output is afterwards interpret by the C sharp framework where the task sequence is stored.
The controller is implemented in Python with the use of the Casadi toolbox communicates with C sharp via specific text files. One side is writing the text files and the other side reads it. For example communicating the velocity profiles and positions of the AGV.
The framework contains also a PLC communication part and a camera module to communicate with the real plant. However these parts are not discussed as they are not implemented in the simulation model.



ID: 1 Orientation: 0
Position x: 33 Position y: 150
Filling: Yes and Yes Colors: Yellow and Black

ID: 3 Orientation: 180
Position x: 367 Position y: 150
Filling: Yes and Yes Colors: Red and Blue

ID: 4 Orientation: 315
Position x: 72 Position y: 220
Charging: Yes

ID: 7 Orientation: 225
Position x: 328 Position y: 220
Charging: Yes

ID: 1

ID: 2

ID: 3

ID: 4

Figure 3.2: Graphical interface of the previous framework.

4 Project motivation

To analyze a Cyber Physical System (CPS) [Lee, 2008a] it might be useful to combine heterogeneous submodels in an integrated environment to perform a comprehensive simulation. This comprehensive view on the system will detect problems caused by interaction between submodels in early stage of development.

This project aims to develop a comprehensive model with concurrent execution for the pipeless plant. This model will contain the physical processes of the AGV, the controller and the scheduler. These parts are coupled subsystems of the pipeless plant. PTOLEMY II has been identified as a suitable modeling environment. If this comprehensive simulation model can give more insight, it will be an addition to model driven development.

Researchers at the Process Dynamics and Operations group at TU Dortmund work mostly on confined control projects, where they forget about other aspects. This project has the potential to provide a framework that supports modeling other aspects, which adds to the modeling knowledge within the group.

5 Ptolemy II

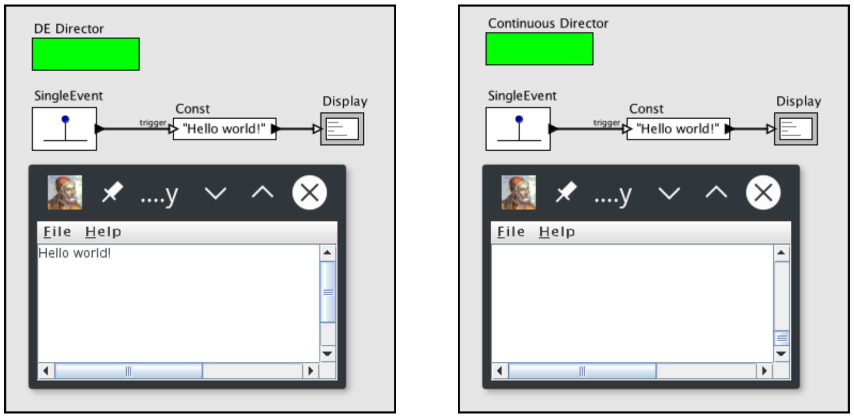
This chapter gives a brief view of PTOLEMY II [Ptolemaeus, 2014]. Examples and important building blocks are explained.

5.1 General

Ptolemy is a modeling language which is built for multiple types of models. For example, state machine, discrete events, continuous time, dataflow and network models are a few of the types available. These different types can be used within each other. This makes Ptolemy a very interesting tool to combine a physical model with a scheduling. Every type has a so called Director with a specific computation behavior.
Ptolemy is built on java language. One big reason for using java is the multi-treadedness [Lee, 2008b]]. This ensures that multiple processes can be executed at the same time.
The building blocks of Ptolemy are called *Actors*. These Actors have input and/or output ports which are connected with other Actors. There is a wide range of Actors available. Self-explaining Actors as Add/Subtract, XYPlotter and Scale. But there are also Actors which need some explanation.

5.2 Directors

A model in PTOLEMY behaves according to the director within that submodel. To illustrate the difference between the used directors in the simulation an example is presented. Figure 5.1 shows the 'Hello World' model. When a discrete event director is used only one line with Hello World will be displayed. This is caused by the single event that triggers the constant which sends its value to the display. Replacing the discrete event director with the continuous time director gives different behavior. The continuous time director is continuously requesting input at the display actor. But only once a value is received in the first iteration. This results in a lot of blank lines in the display. This concludes that the discrete event director only sends messages when there is a trigger in contrast to the continuous time director which requests continuously for messages even if there are none.



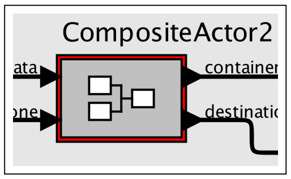
Hello world!

(a) Discrete Event director (b) Continuous Time director

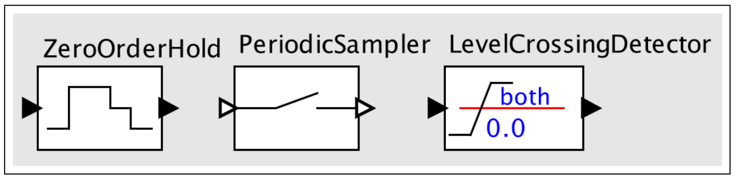
Figure 5.1: Hello World model.

5.3 Actors

In this section frequently used actors are explained briefly. The first and mostly used actor is the Composite Actor shown in figure 5.2a which is nothing more than a submodel. Within this Composite Actor a different director can be used to combine different model behaviors.



(a) Composite Actor.



(b) Continuous/Discrete Actors.

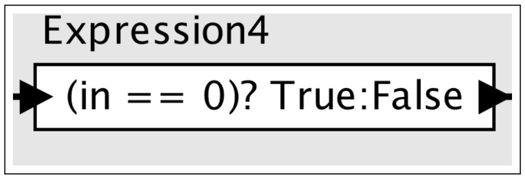
Figure 5.2: Actors.

One combination that is frequently used is a continuous submodel within a discrete event model. To be able to communicate from the discrete environment to the continuous environment specific actors are needed. These actors can be seen in figure 5.2b and are explained in table 5.1.

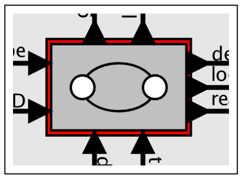
Table 5.1: Actors Continuous/Discrete environment communication.

Name	From	To	Action
ZeroOrderHold	Discrete	Continuous	Holds the last value, which can be used continuously.
PeriodicSampler	Continuous	Discrete	Samples the continuous values at a specific time.
LevelCrossingDetector	Continuous	Discrete	Sends a message when the variable crosses a certain value.

The expression actor can be used to perform an operation using PTOLEMY's expression language. In figure 5.3a an example is showed of an expression. This expression can be build with multiple actors but it is more efficient to make it as an expression. This expression represents an if-statement, when the input port in is equal to 0, the output is True and vice versa. The documentation of the expression language can be found in [Ptolemaeus, 2014].



(a) Expression Actor.



(b) ModalModel Actor.

Figure 5.3: Actors.

The last actor is the ModalModel as can be seen in figure 5.3b. This is a finite state machine where a state can contain a different submodel. The ModalModel behaves according to the current state's submodel.

6 Implementation

As mentioned before, the three main parts of the simulation are the physical behavior of the AGV, the controller and the scheduler. In this section is discussed how the parts are implemented in the comprehensive PTOLEMY model.

6.1 AGV

The AGV in the setup in question is differentially steered. Since the controller is already developed for the previous simulation, and preferably used again in this model, it is chosen to model the AGV in the same way.

This results in a mathematical model as can be seen in equations 6.1 - 6.3. The input variables of the AGV are the velocity v and the angular velocity ω. The output variables are the x and y position and the angle θ.

$$\dot{x} = v \cdot \cos(\theta) \tag{6.1}$$
$$\dot{y} = v \cdot \sin(\theta) \tag{6.2}$$
$$\dot{\theta} = \omega \tag{6.3}$$

This mathematical representation is translated into a PTOLEMY model as can be seen in Figure 6.1. The director of this submodel is the 'Continuous Director'. This director is used because the AGV drives continuously and the integrator actor can be used within this model.

In figure 6.1, the input ports are the `rotationspeed` which is the angular velocity, and the `speed` which is the velocity. For both inputs a limiter is implemented to avoid extreme velocities with manual inputs. (This is the actor with the downwards arrow.) With the use of a controller these limits are never reached. As can be seen the angular velocity is integrated this results in the `angle`. The cosine and sine of this angle times the velocity is integrated to get the `x` and `y` position of the AGV. The initial conditions for the integrator for the `x` and `y` positions are set in the parameters `x0` and `y0`. There is also a `scale` variable to scale the velocities, but this is set to `1.0` when used with the controller.

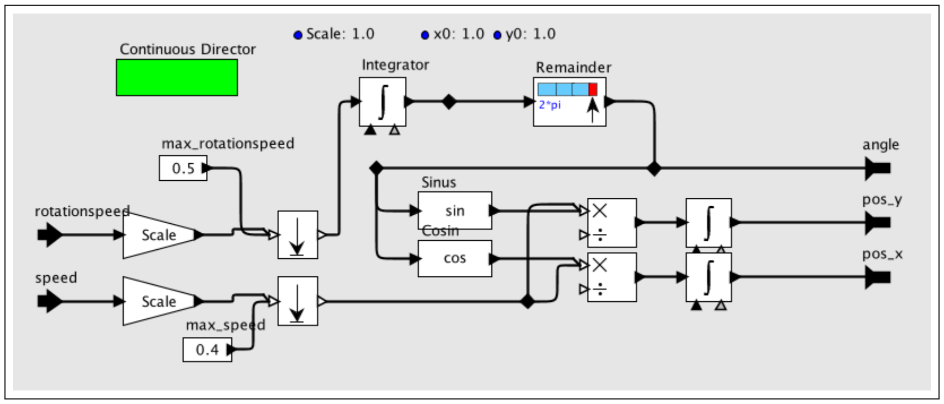


Figure 6.1: AGV model in PTOLEMY

6.2 Controller

The Nonlinear Model Predictive Controller (NMPC) developed by [Alvi, 2015] is built in Python with use of the Casadi toolbox. Casadi is an opensource symobolic framework for algorithmic differentiation and

numeric optimization, usable within Python, Matlab etc.
This controller was implemented in the previous simulation connected to the C sharp framework. The communication was built as writing and reading a text file. This setup was found to be non-compatible with the PTOLEMY model. Therefore a communication module is developed in Python which communicates via sockets to send and receive messages. These messages could be position updates, velocity updates or new destinations for the controller.

The Python controller code is adapted to host two sockets with the use of the communication module. One socket to receive position updates and one to send the new velocity profile. Likewise two Python scripts are developed to receive the velocity profile and send the position update. At this stage there are three Python scripts needed to be implemented in the PTOLEMY model to integrate the controller.
The developers of PTOLEMY claim that it is compatible with Python using the Python actor. However, it is compatible with Java Python. The Casadi toolbox is only available for C Python, which makes the Python actor useless for the controller implementation. Also the Python code within the actor has to be written within a certain PTOLEMY framework to guarantee communication behavior in the PTOLEMY model.
The solution is to use the command line actor, which can execute the C Python scripts via the command line. Three of these actors are needed to execute the scripts as can be seen in figure 6.2 on the PTOLEMY side. After implementation the problem occurred that the command line actor, under a `Discrete Director`, waits until the executed command is terminated before the time in simulation can pass. This will cause a deadlock, because the controller code only terminates when the destination is reached, but the AGV can not move because the simulation time can not pass.

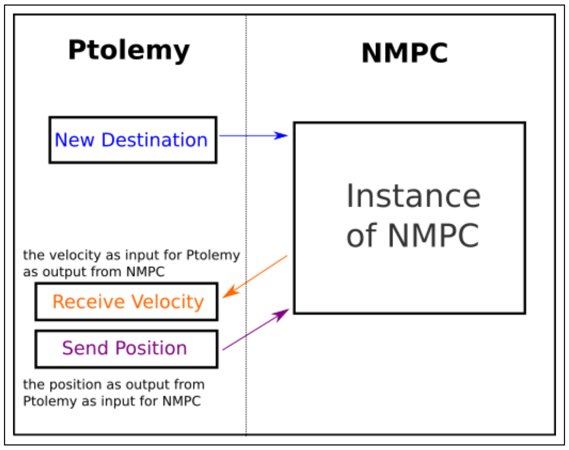

Ptolemy
NMPC
New Destination
Instance of NMPC
the velocity as input for Ptolemy as output from NMPC
Receive Velocity
Send Position
the position as output from Ptolemy as input for NMPC

Figure 6.2: First communication scheme.

To solve this problem the threading actor is used. This actor runs the command line actor in a background process, but only for a specified amount of time, the time that the AGV drives to the destination. Estimating that time delay is hard because there can be obstacles. To avoid wasting any time within the simulation, the hosting of the controller code has to be taken from the PTOLEMY model to another background process. For this purpose an extra loop has been created. This loop will be running in the background for as long the simulation runs, waiting for a new destination to host a controller instance. One extra script is developed to send new destinations to the loop, with the use of the communication module. Resulting in four Python scripts to implement the controller.

The resulting scheme as can be seen in figure 6.3, shows the communication between the scripts. First the NMPC loop is started as a background process for infinite time. The loop is now waiting to receive a new destination. When a new destination arrives the loop starts an instance of the NMPC controller script for that destination. The AGV receives its first velocity profile to drive to the destination, and after sampling time the position is updated to the NMPC instance. This is repeated until the AGV arrives at the destination. The loop will go back in waiting state for a new destination.

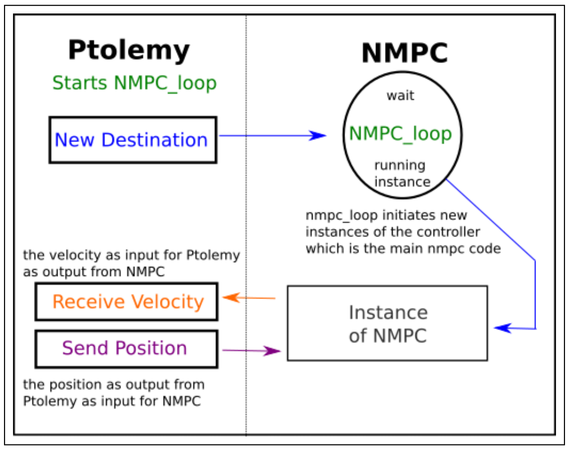

Ptolemy
Starts NMPC_loop
New Destination
NMPC
wait
NMPC_loop
running instance
nmpc_loop initiates new instances of the controller which is the main nmpc code
the velocity as input for Ptolemy as output from NMPC
Receive Velocity
Send Position
Instance of NMPC
the position as output from Ptolemy as input for NMPC

Figure 6.3: Final communication scheme.

The PTOLEMY model of this implementation is displayed in figure 6.4. The blue triangles are the command line actors which execute the python scripts. After starting the simulation the `SingleEvent` starts the NMPC loop. When there is a destination coming in via the `newDestination` port it is first checked if it was the same as the previous destination. That means that the AGV is already at that destination and this one can be skipped. Otherwise it is sent to the NMPC loop which starts the controller instance. And in the meanwhile the position and velocity is constantly updated. Even when de controller is not running, then the velocity profile will be zero and the position update will be unused. The `AGV_ID` parameter is used to choose a socket port, which has to be different for each controller used at the same time.

There are several controller configurations which can be used. These are listed in table 6.1. The 1 robot configuration is only used to guide one AGV at the same time. The 3 AGV configuration is used for guiding 3 AGV's at the same time with one controller to each their destination. When one of the AGV's is at the destination it has to wait for the last AGV to finish. The Multi Shoot configuration is a controller which guides one AGV to a destination with the other AGV's as dynamic obstacle. This implies one controller per AGV.

Table 6.1: Controller configurations

Name	# AGV's	# controllers	# obstacles
1robot	1	1	0
2robot	2	1	0
3robot	3	1	0
MSrobot	i	i	i-1

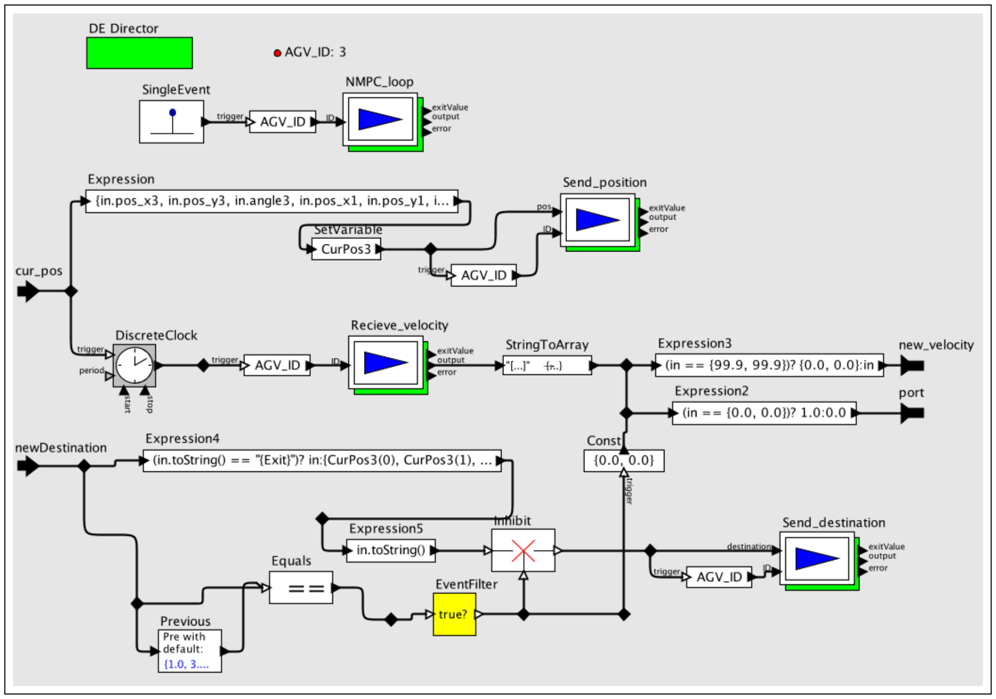


Figure 6.4: Controller model in PTOLEMY

6.3 Scheduler

The scheduler is developed for the Multi Shoot controller configuration for multiple AGV's. The basic setup with two fill stations, one mix station and one storage station is used. Since the scheduler is running concurrently with the rest of the simulation the tasks have to be scheduled real-time. Therefore a new way of scheduling is needed. First a concept is introduced before the implementation is discussed.

A Concept

In general a pipeless plant is introduced to increase flexibility in a production process. If a bottleneck slows down the whole process, an extra station can be added to release the pressure on the bottleneck station. Same holds for number of AGV's. To be able to have this property in the simulation, a configuration file has been introduced. An example configuration file can be found in appendix A. In the configuration file the stations are specified with their respective location. Per fill station the products are specified as an index number, so it can represent any product. The fill rate of a product is the amount per second the fill station can fill. For the mix and storage station a fixed delay for mixing and storing is set. Adding an AGV is not possible via the configuration file. To add one AGV several parts of the model have to be copied and connected.

The basic setup contains four products which can be gathered at two fill stations. For this setup the recipe for one container have the following template: `{{p0,p1,p2,p3},{p0,p1,p2,p3}}` where `px` is the quantity of product `x`. This template contains two layers, but the scheduler can handle infinite number of layers.

To process one layer the following steps are taken:

- Pick up container
- Fill the container according to recipe

- Mix the layer volume
- Store the container

These steps are executed uninterrupted by one specific AGV. This is implemented as a state machine in PTOLEMY which is discussed later.

B PTOLEMY implementation

The overview of the scheduler model is shown in appendix B.3. The individual parts of the scheduler are explained in this section.

In figure 6.5 is shown where the configuration file is read and the variables are set. The rest of the scheduler model can use these variables to determine the position of a station and what the process time will be. The publisher ports at the right send the position of the stations to the plotter to visualize those in the plot.

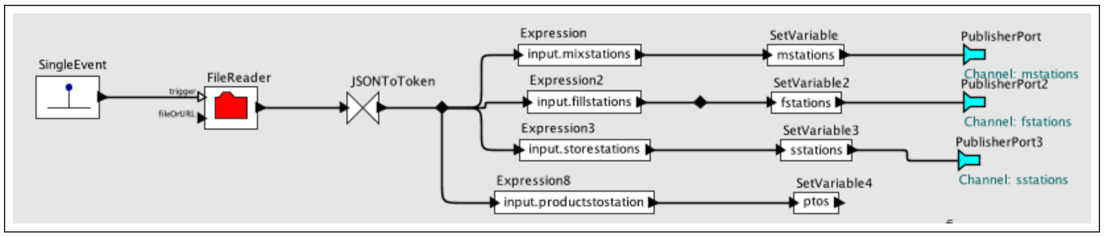


Figure 6.5: Scheduler: configuration reading in PTOLEMY

The recipes shown in figure 6.6 are coming in at a specific time set by the `SingleEvent` actors. The recipes are translated in the `RecipeTranslator` into a dataset which contains the location and process time to fill each product. These datasets are entering the queue `newQ`.

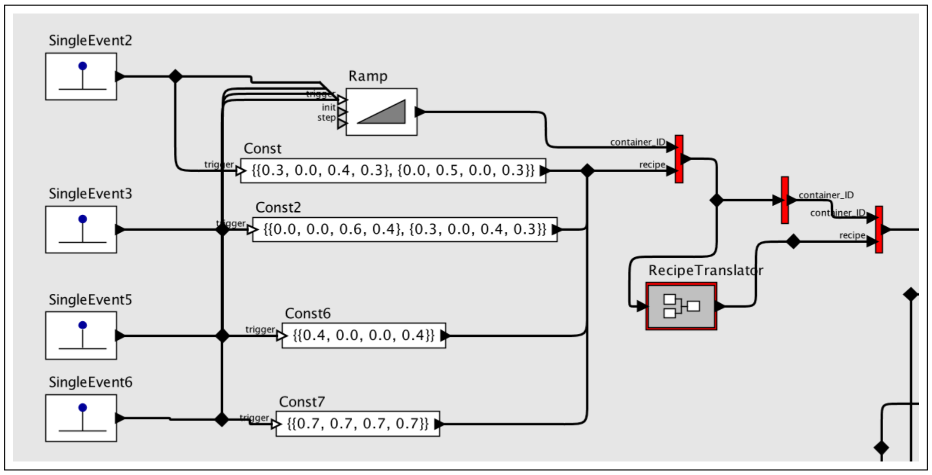


Figure 6.6: Scheduler: recipes input in PTOLEMY

This scheduler makes use of three queues: `newQ`, `busyQ` and `AGV_Q`, as can be seen in figure 6.7. At the `newQ` new recipes are coming in. The `busyQ` contains recipes from containers that have already processed one or more layers. This queue has priority above the new incoming recipes. The `AGV_Q` is a queue where the index of an AGV is coming in to request a new task. When there is a recipe available in the `busyQ` this recipe is send to the layer processor of the AGV with that specific index. If the `busyQ` is empty, the `newQ` is checked for an available recipe.

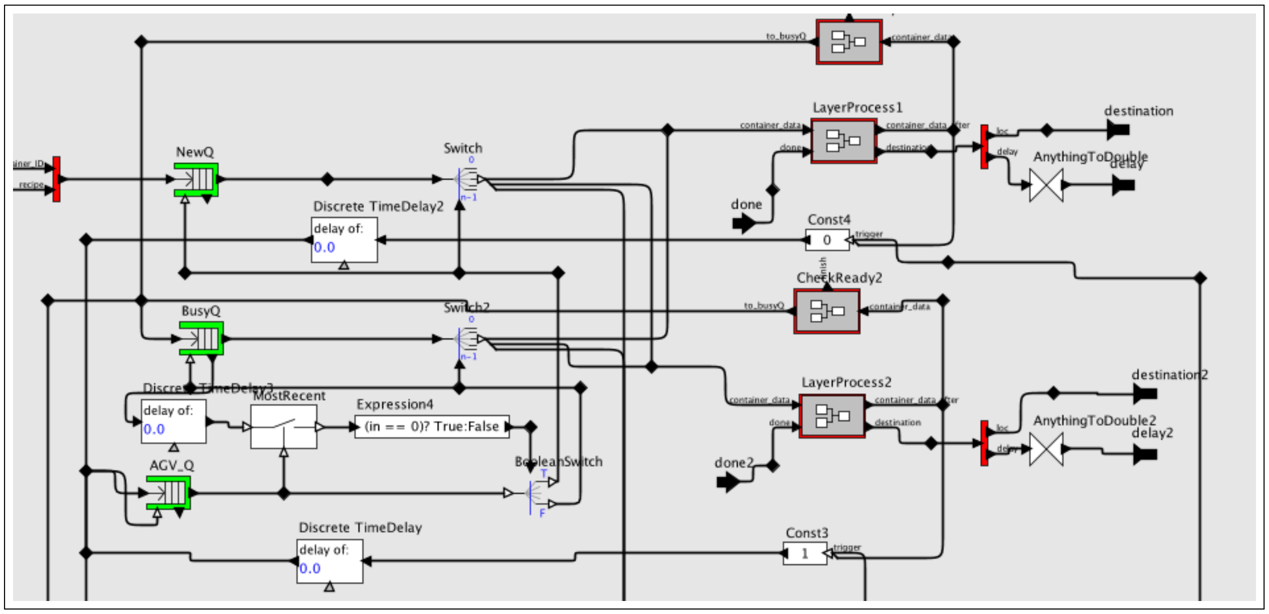


Figure 6.7: Scheduler: distribution of recipes in PTOLEMY

The layer processor contains the state machine described earlier, with data flow around it. When a recipe comes in, the next layer is obtained and sent to the state machine. The state machine will handle this layer. When the layer is processed this layer is removed from the recipe data and the rest is either send to the `busyQ` or it is finished via the `container_data_after`.

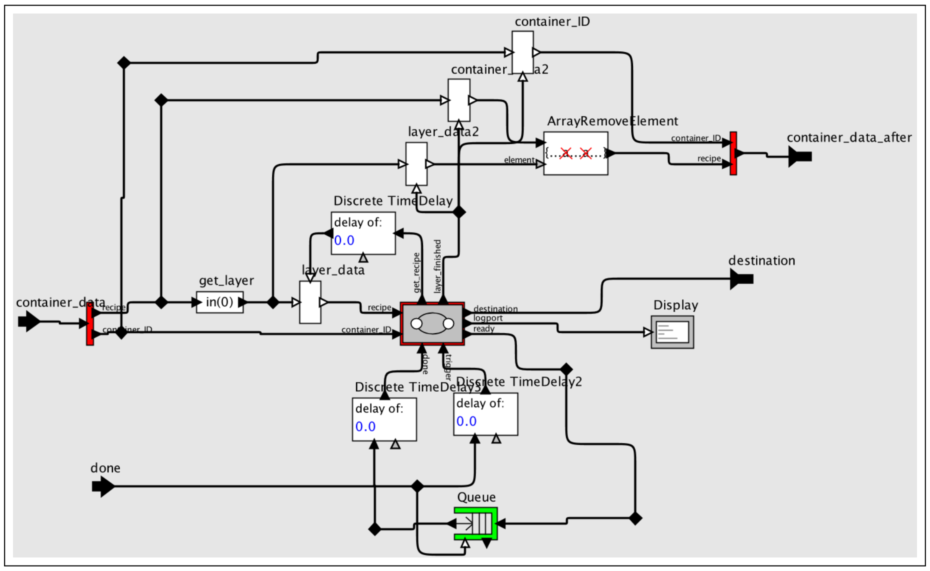


Figure 6.8: Scheduler: layer processor in PTOLEMY

Within the state machines states, the destination and delay of each process is retrieved from the variables set by the configuration file. Except for the fill state, figure 6.9, which uses the recipe data which is translated earlier to destination and delay. The zero delays are filtered out and the rest is processed sequentially. This information is then sent to the controller which drives the AGV to the destination. When the AGV arrives at the destination and is processed, a message arrives at the state machine and the next step can be processed.

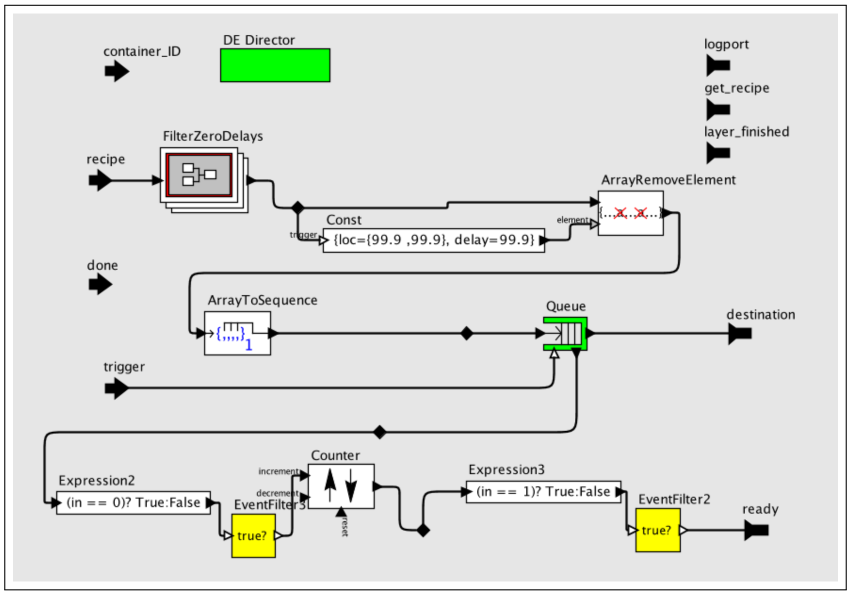


Figure 6.9: Scheduler: fill state in PTOLEMY

6.4 Model overview

In figure 6.10 the overview of the model is shown containing the scheduler, controller and AGV model. The scheduler communicates the destination to the controller. This controller guides the AGV via velocity profile messages to the destination. The AGV is constantly sending position updates to the controller. When the AGV arrives at its destination the controller sends a message back to the scheduler, which can send a new destination.

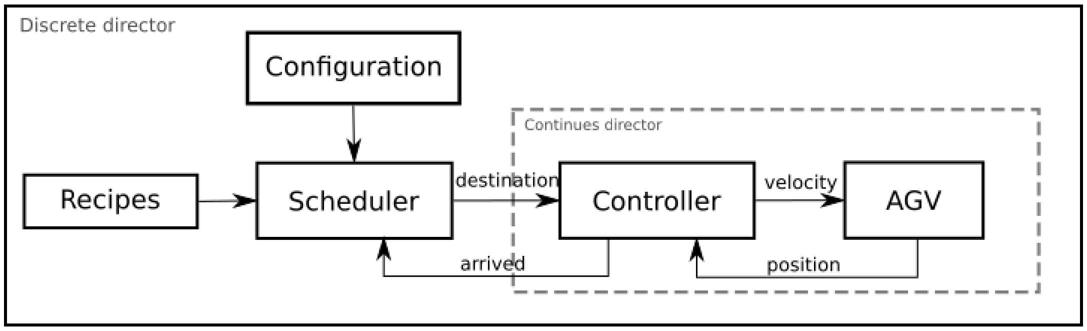


Figure 6.10: Model overview

7 Experiments

To show the behavior of the comprehensive model some simulated experiments are executed. These experiments are discussed to give a general view on the advantages and disadvantages of this new modeling approach.
As mentioned earlier, there are multiple configurations of the controller. The first simulation shows the case where there is one controller, controlling three AGV's at one time. In figure 7.1a can be seen that the AGV's start exactly at the same time to drive to their new destination. This will always be the case. When the first AGV arrives at its destination as can be seen in figure 7.1b, it has to wait for the other AGVs to start driving to the next destination. This controller is not used in the final model with scheduling because of this behavior. Time is wasted waiting on the other AGV's and there has to be always a new destination for every AGV even if there is only one task to fulfill.

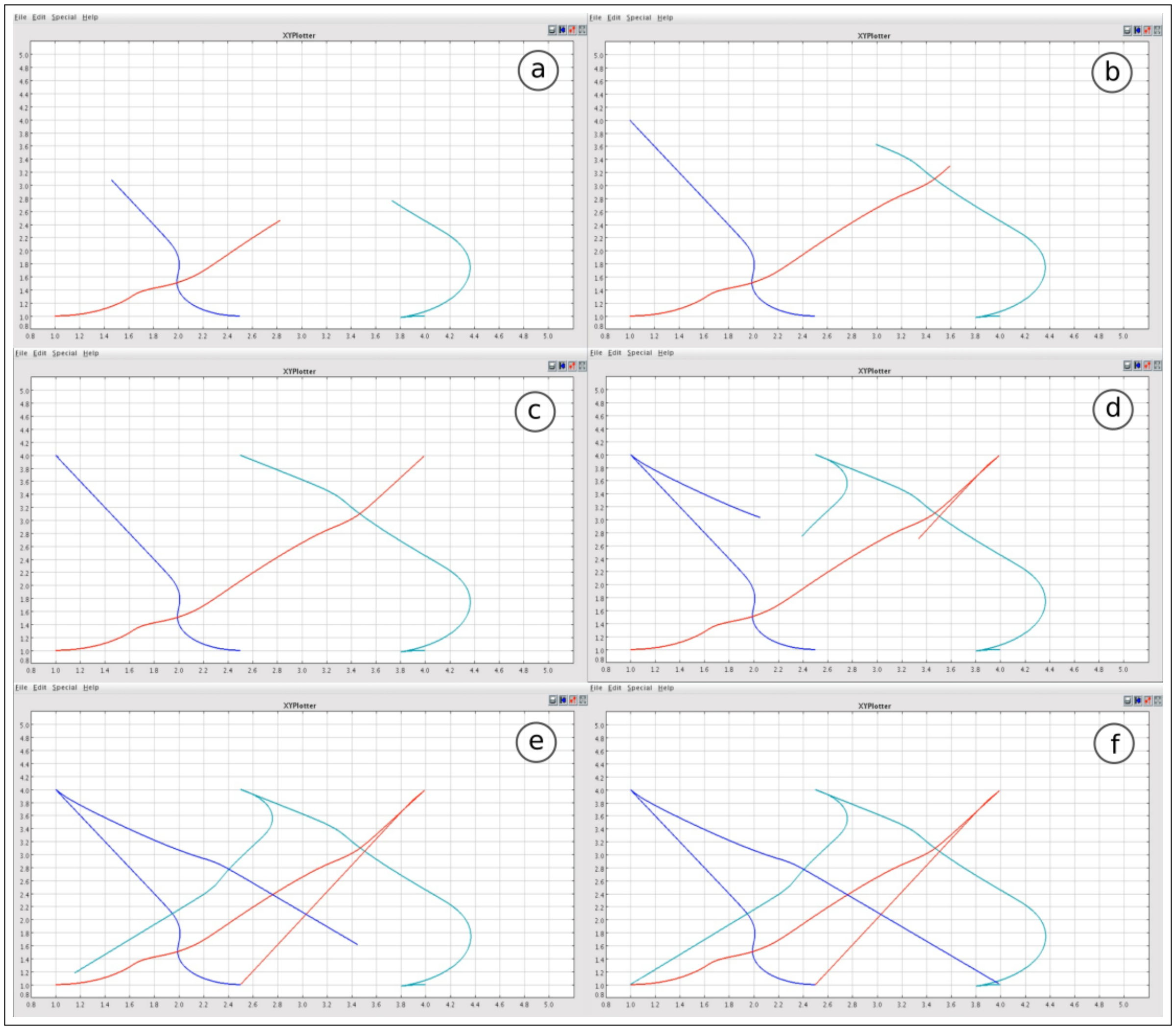


Figure 7.1: Simulation of 3AGV's with one controller

The next simulation shows also three AGV's but with each their own controller. This configuration is called the Multi Shoot controller. Here one AGV is controlled and the other AGV's are dynamic obstacles. In this simulation the scheduler is build-in. Figure 7.2 show the following stations:

- North: mix station
- East: fill station 2
- South: storage station
- West: fill station 1

This simulation shows clearly the influences between submodels. Every AGV goes first to the storage station in the south to pick up a container, afterwards they go to a fill station according to the recipe. After the red AGV leaves the east fill station (7.2c) the other two AGV's want to approach this station. Since the two AGV's are obstacles of each other neither of them can enter the station (7.2d). This causes a problem, because later the red AGV has to go there as well and three AGV's are blocking the station (7.2f). This is caused by the fact that the scheduler does not know whether a station is occupied or not. To make the controller and scheduler working fine together this knowledge has to be implemented.

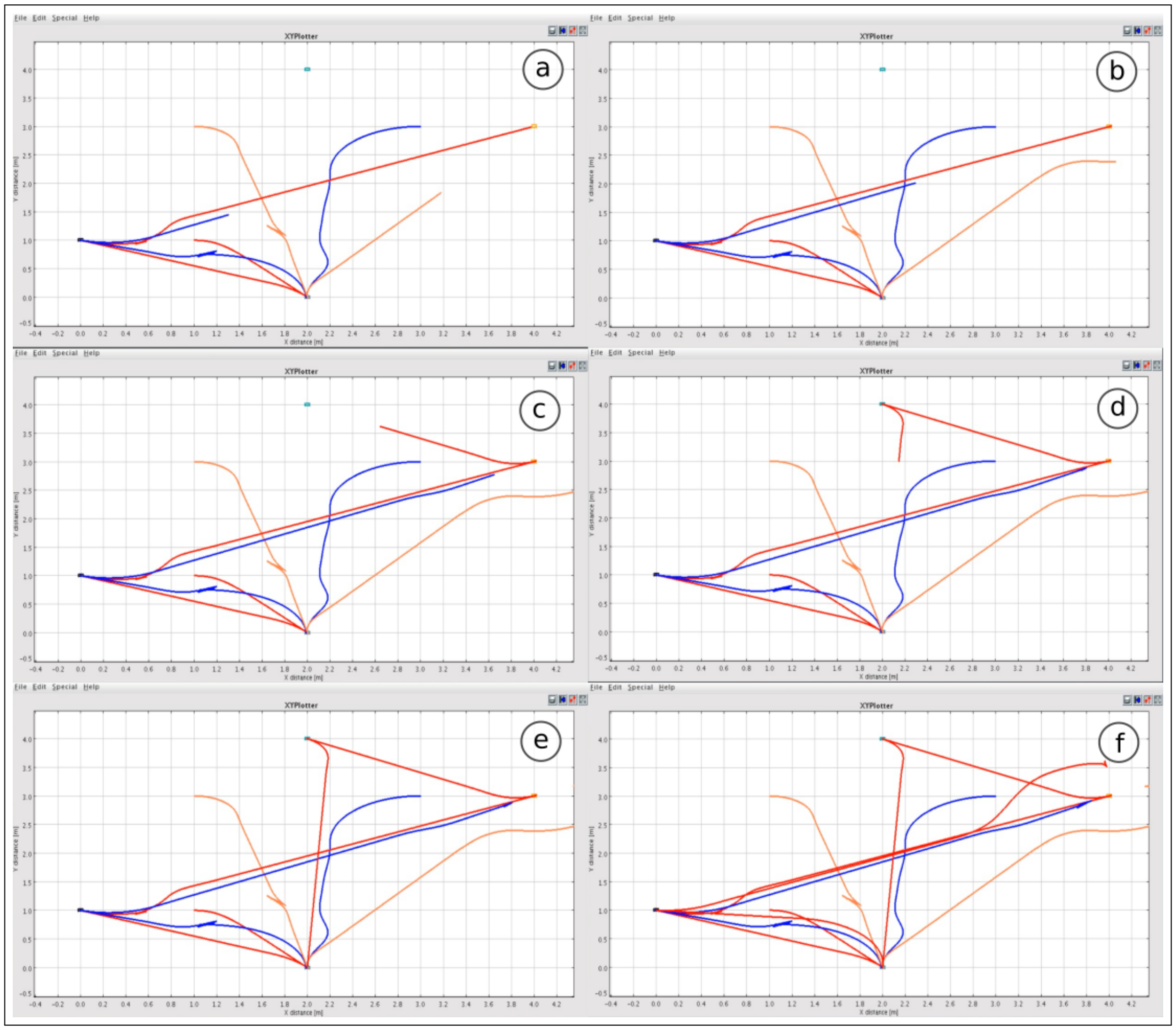


Figure 7.2: Simulation of 3AGV's with each a controller

The simulation configuration containing only two AGVs works according to expectation, as illustrated in figure 7.3. When one leaves the station the other can use it. But when they approach one station at the very same time the same problem may occur.

The last simulation is an implementation of the a camera uncertainty. At the pipeless plant setup at the TU Dortmund a fisheye lens is used to get a wider viewing angle. This causes also detecting problems at the outer limits of the arena. The position error of the AGV's are position dependent. In the middle of the arena the error is small, and the more the position is away from the middle the error increases. This behavior is added in a submodel and implemented in the two AGV setup. In figure 7.4 the position error is shown. The AGV's still can find their way to the stations, but sometimes it takes a few controller iterations more than normal. Resulting in the conclusion that the influences of this camera error are not big enough to mislead the controller.

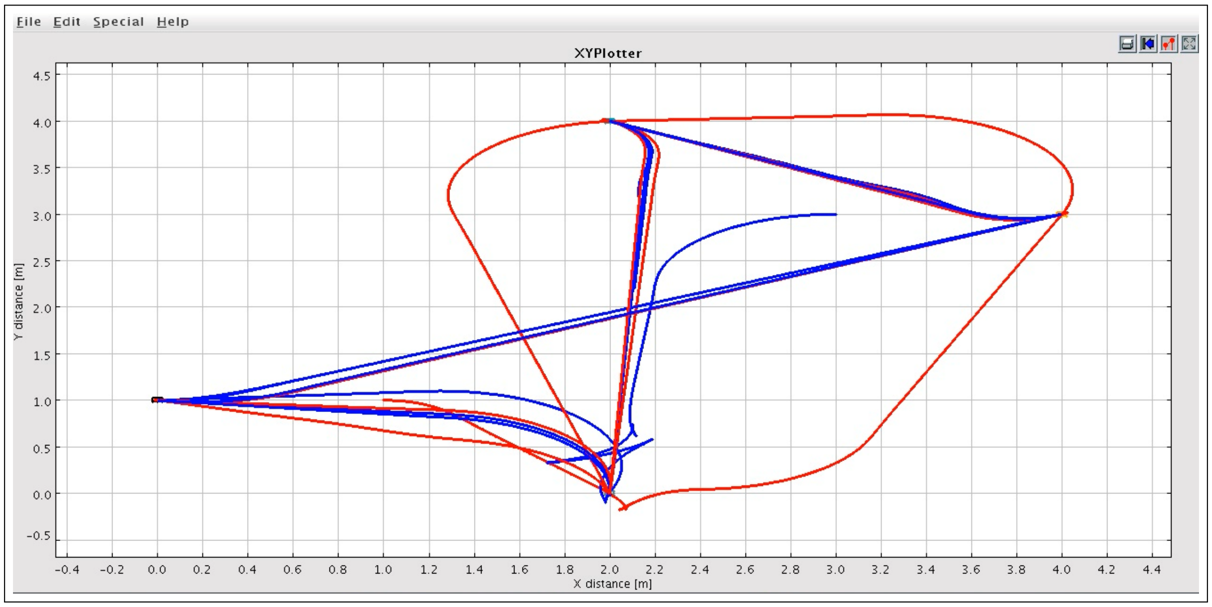


Figure 7.3: Simulation of 2AGV's with each a controller

7.1 Master-slave determination

In this setup there is no priority in AGV's, they have exactly the same set of rules. When they drive at the same time to the same destination it will cause a deadlock as mentioned before. With two AGV's this scenario will be seen less because one AGV will be a fraction earlier most of the time. It looks like the first AGV is the master and the second is the slave. Adding one extra AGV will immediately cause trouble as discussed earlier. On what basis should the model determine who is the master? How will this be handled with four or more AGV's?

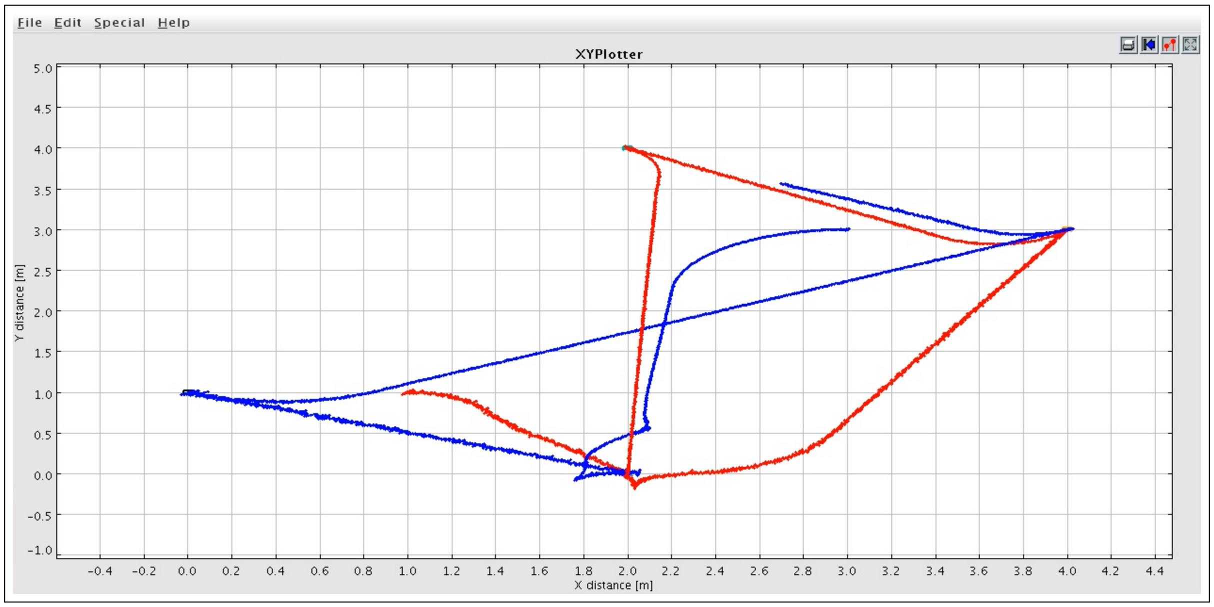


Figure 7.4: Simulation of 2AGV's with position error

8 Conclusion

PTOLEMY II can be labeled as a suitable environment for building a comprehensive model containing heterogeneous submodels. The claim that they are compatible with Python has to be demoted. A scientific modeling language which is unable to use Pythons scientific toolboxes can not state to be compatible with Python. The provided work around in this report using the communication module between PTOLEMY and the Python scripts is needed to solve this problem. But it is possible to say that this implementation is a good representation of reality. If each AGV has its own on board computer this will be represented well. The destination can be sent to that on-board computer and each AGV has its own controller calculation.

The resulting simulations show that the scheduler and controller are not working together properly causing a deadlock. When the scheduler has knowledge of the occupation of the stations this problem could be solved. The simulation also showes that the two AGV setup will work and the fisheye lens will not causing problems for the controller.

This might be a step forward in model based engineering for multi domain models. Detecting problems in the design phase caused by interaction between submodels.

9 Future work

Recommendations:

- Improve the scheduler with knowledge of station occupation, maybe a future planning with time estimation.
- Improve the modularity of the simulation so that the scheduler can handle multiple mix stations for example.
- For the implementation on the real plant an on board computer for each AGV which runs the separate controllers instead of one computer doing all the computation work.

References

[Alvi, 2015] Alvi, M. A. S. (2015). *Robust Control of mobile robots for pipeless plant using NMPC*. PhD thesis, Technische Universität Dortmund.

[Lee, 2008a] Lee, E. A. (2008a). Cyber Physical Systems: Design Challenges. *International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC)*, pages 363–369.

[Lee, 2008b] Lee, E. A. (2008b). ThreadedComposite : A Mechanism for Building Concurrent and Parallel Ptolemy II Models. (April).

[Ptolemaeus, 2014] Ptolemaeus, C. (2014). *System Design, Modeling, and Simulation. Using Ptolemy II.*

[Schoppmeyer et al., 2012] Schoppmeyer, C., Hufner, M., Subbiah, S., and Engell, S. (2012). Timed automata based scheduling for a miniature pipeless plant with mobile robots. *Proceedings of the IEEE International Conference on Control Applications*, (May 2017):240–245.

A Configuration file

```
 {
        "fillstations": [
                         {
                         "name": "fillstation1",
                         "location": [0.0, 1.0],
                         "products": [0, 1],
                         "rate": [0.2, 0.2]},

                         {
                         "name": "fillstation2",
                         "location": [4.0, 3.0],
                         "products": [2, 3],
                         "rate": [0.2, 0.2]}
        ],

        "mixstations": [
                         {
                         "name": "mixstation1",
                         "location": [2.0, 4.0],
                         "delay": 2.0}
        ],

        "storestations": [
                         {
                         "name": "storestation1",
                         "location": [2.0, 0.0],
                         "capacity": 6,
                         "delay": 2.0}
        ],

        "productstostation":[0, 0, 1, 1]
}
```

B Overview PTOLEMY model

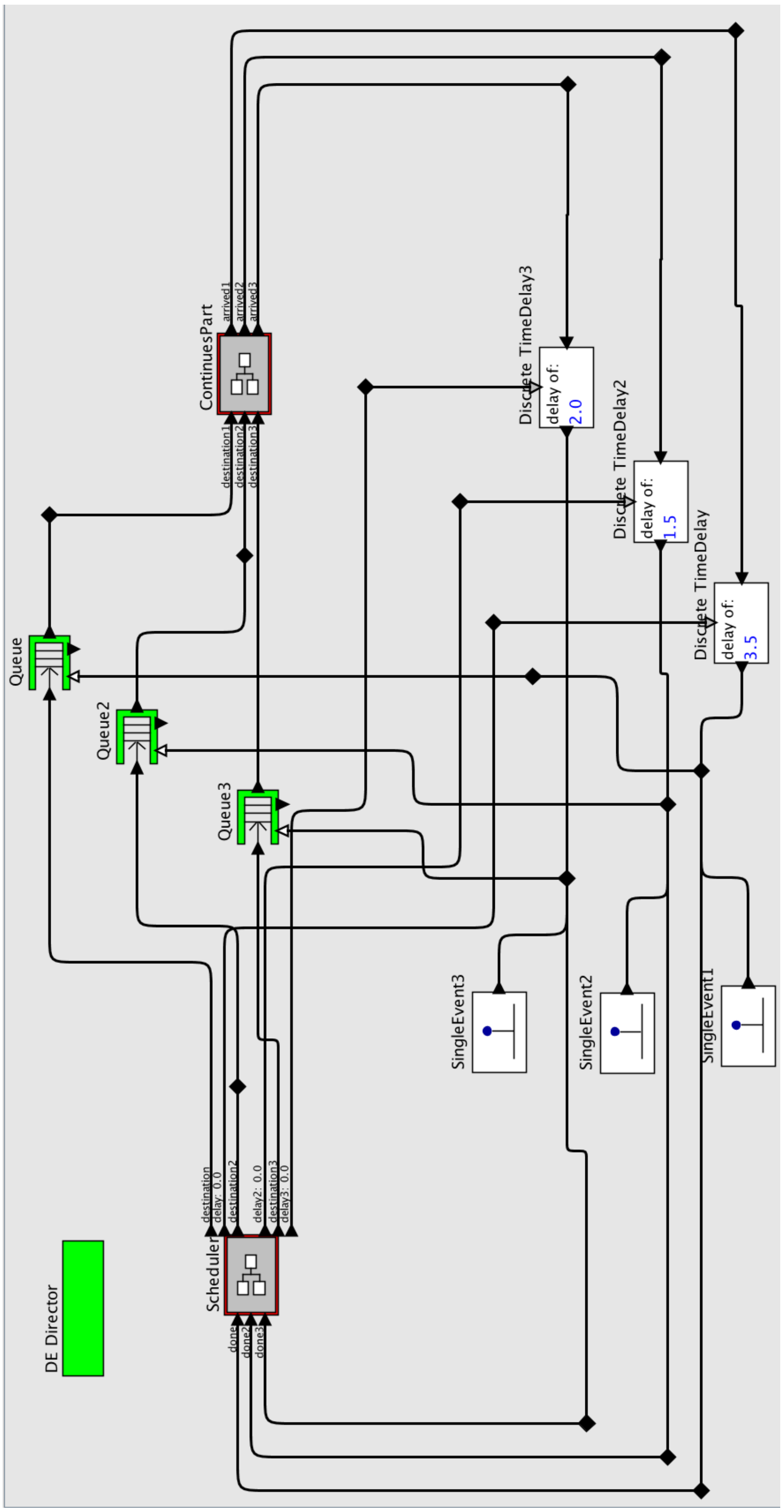


Figure B.1: Model overview

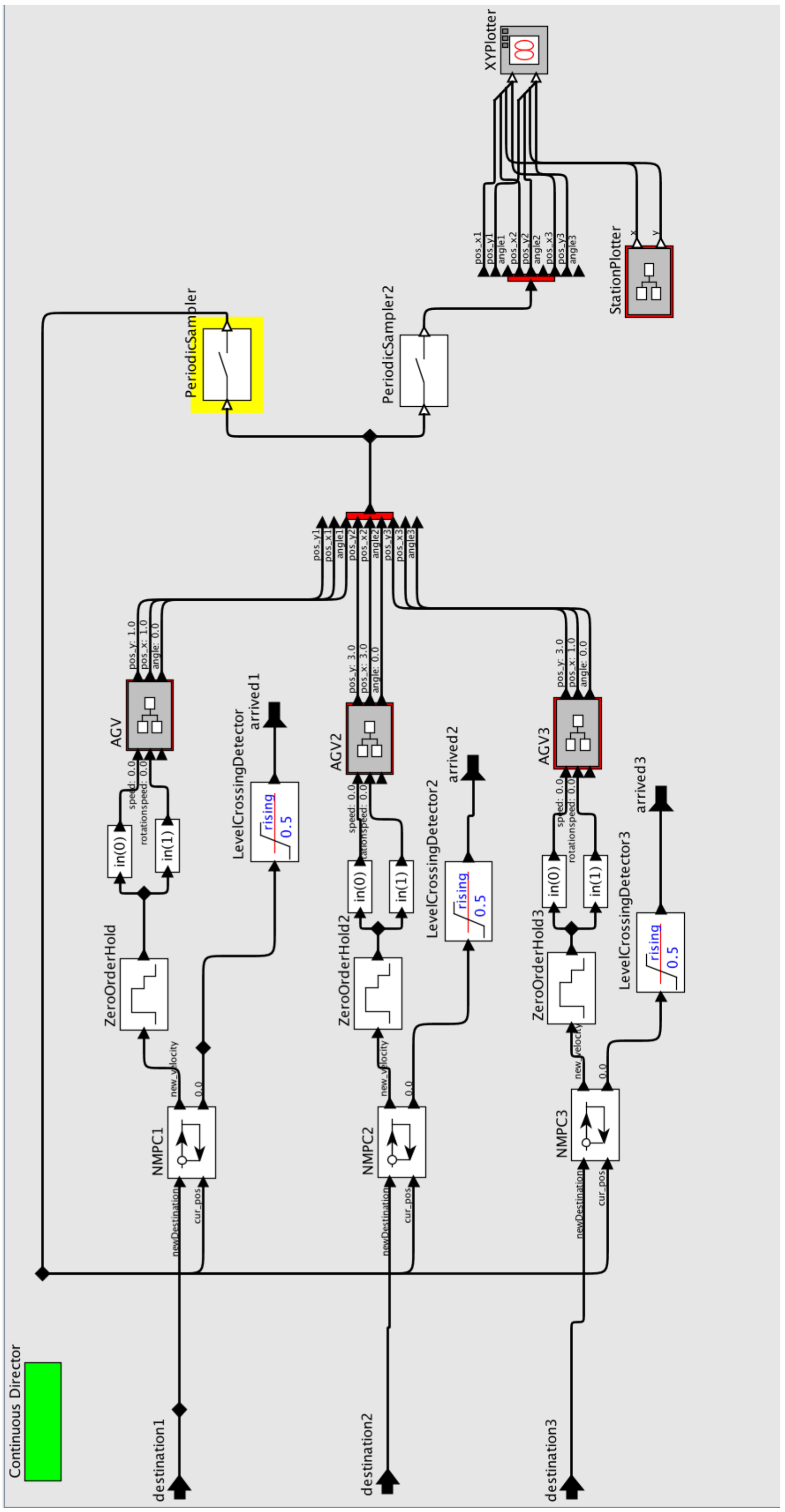


Figure B.2: Continues part overview

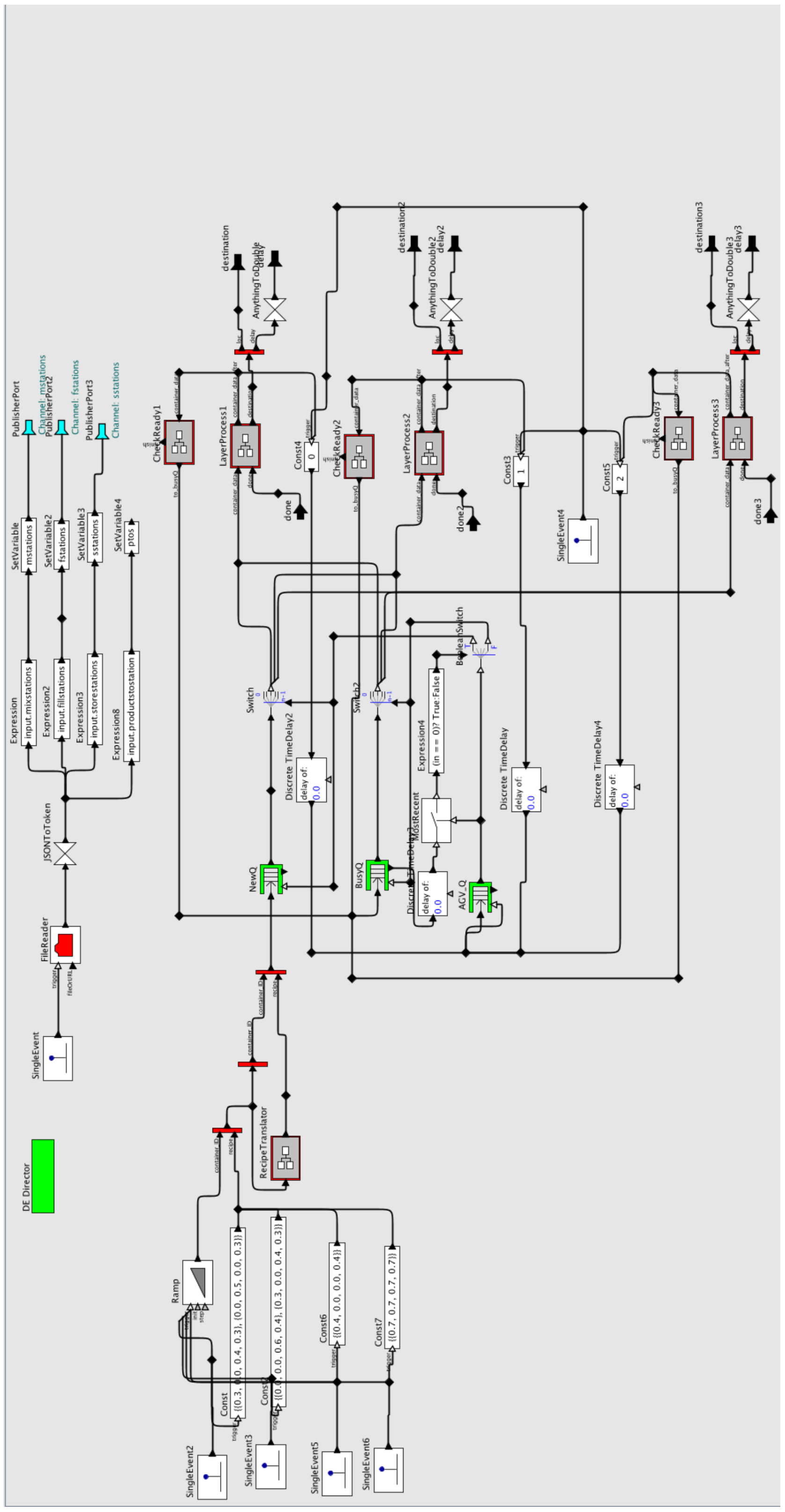


Figure B.3: Scheduler overview