Demonstrator combining ROS/TERRA-LUNA

P.D. (Peter-Jan) Vos

MSc Report

November 2015

032RAM2015
Robotics and Mechatronics
EE-Math-CS
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

Summary

The Production Cell is an existing model of a plastic moulding plant and is used as a platform to test multiple hard real-time embedded control systems. The usage of hard real-time systems ensures a reliable control system, but limits the possibilities for additional tasks that can be performed since they can exceed the time window of the hard real-time system.

At the University of Twente a 'bridge node' is made which links a hard real-time system and a soft real-time system. The LUNA bridge connects the in-house developed hard real-time execution framework LUNA to ROS. ROS is an increasingly popular soft real-time software package to control robots.

With the bridge node, called luna-bridge, it is possible to control actuators in hard real-time and update their setpoints based on complex calculations done on another computational platform. To demonstrate the luna-bridge, a new module is added to the Production Cell which identifies and extracts certain blocks of the plant. This sorting module is equipped with a webcam, which images are analysed in a ROS-node. The blocks that move around the Production Cell are marked with certain shapes, which are identified in the ROS-node. If a positive match is found with the hard-coded shapes, a new message is sent over the luna-bridge to the hard real-time system.

The hard real-time software runs on a QNX-based PC104. An AnyIO-card is used to communicate between the motor/encoder and the LUNA application. The LUNA application receives messages from the ROS node, and converts these to a setpoint. The sorting rod is then controlled to this position and the block is extracted from the belt.

The image processing node takes 18 ms to analyse the shapes of one frame. On average, a total of 9.2 frames are captured by the webcam when a block passes. Out off these 9.2 frames the image processing algorithm is able to extract the three shapes in 8.5 frames. Identification of the shapes on these frames is done with 100 % accuracy based on a test with 36 blocks, so that the sorting module is always able to correctly identify the block marked for removal.

The sorting module is able to move to its extraction position within 0.4 seconds. The module is therefore always in time to extract the block marked for removal, which will arrive 0.4 seconds later and is successfully removed after another second. However, when the Production Cell is operating at maximum speed, the sorting module can unintentionally push off the block in front of the desired block.

Additions to TERRA, or the 20-sim to TERRA code generation, are required to add a timing mechanism to the platform so the sorting module can operate safely while the Production Cell is operating at full speed. The mechanical design of the sorting module can be improved by making it smaller, decouple the rod from the shaft and adding end-switches. The software uses hard-coded shapes which are tagged for removal, a modification so that the removed shapes can be set at run-time would be nicer. In future projects, the luna-bridge can also be integrated in more complex set-ups which require bidirectional communications.

The new sorting module is a stand-alone system which is added to the Production Cell. It works without interfering with the normal workings of the Production Cell, and can be removed if necessary. The new module uses a hard real-time LUNA application to control the motion, while the image processing is offloaded to a soft real-time system. The platforms communicate through the luna-bridge, which sends commands from the soft real-time ROS application to LUNA. By using the luna-bridge, it is possible to combine the benefits of a hard real-time system with a soft real-time system.

Samenvatting

De Productie Cell is een bestaande model-opstelling van plastic-giet machine. De opstelling wordt voornamelijk gebruikt om verschillende implementaties van hard real-time control systemen te testen. Hard real-time systemen zorgen voor betrouwbare regelsystemen, maar beperken de taken die het systeem kan uitvoeren doordat ze gebonden zijn aan een bepaalde tijdsperiode waarin alle berekeningen moeten worden uitgevoerd.

Een 'bridge-node', die een hard real-time system en een zacht real-time systeem kan koppelen, is daarom ontwikkeld aan de Universiteit Twente. Deze 'luna-bridge' koppelt het hard real-time LUNA framework aan ROS. ROS is een populair zacht real-time softwarepakket om robots aan te sturen.

Met deze node is het mogelijk om de aansturing van actuatoren in LUNA te houden, terwijl de complexe berekeningen worden gedaan in ROS. Om te bewijzen dat de luna-bridge voor dit soort taken kan worden ingezet wordt een nieuwe module aan de Productie Cell toegevoegd. Deze nieuwe sorteermodule kan blokken van de Productie Cell verwijderen als deze gekenmerkt zijn door bepaalde figuren. Een webcam verzorgt de beelden die in ROS worden geanalyseerd, waarop een nieuw setpoint naar LUNA kan worden verstuurd.

Het hard real-time platform wordt verzorgd door een op QNX-gebaseerde PC104. Een AnyIO-kaart koppelt de hardware op de sorteermodule met deze PC. Gebasseerd op de signalen van de luna-bridge wordt de sorteerstaaf van de module naar de juiste positie gestuurd via een LUNA applicatie.

De beelden van de webcam worden in gemiddeld 18 ms geanalyseerd. Als een blok onder de webcam doorgaat levert dat gemiddeld 9.2 frames op waarop de figuren op het blokje te zien zijn. Van deze frames kan de software in 91% van de gevallen de figuren succesvol isoleren van de rest van de afbeelding. Van de geïsoleerde figuren wordt 100% correct geïdentificeerd, gebaseerd op een test met 36 blokjes.

Zodra de sorteermodule een signaal krijgt om van positie te veranderen kost het de staaf 0.4 seconden om naar die positie te gaan. Hierdoor is de module, tijdens normaal gebruik van de Productie Cell, altijd op tijd om een blokje te verwijderen of door te laten gaan. Als de Productie Cell volledig is geladen is het mogelijk dat de staaf het blokje voor het te verwijderen blokje eraf gooit, omdat de staaf 0.4 seconde te vroeg op zijn positie is.

Modificaties aan TERRA, of de 20-sim-naar-TERRA code generatie, zijn nodig om dit soort timing problemen op te lossen. Mechanisch kan de sorteermodule worden verbeterd door deze kleiner te maken, de staaf te ontkoppelen van de motor en door eind-schakelingen toe te voegen. Software-technisch zou het beter zijn als de figuren die worden verwijderd niet hard in de code zitten. Tijdens het schrijven van dit verslag zijn de mogelijkheden van de luna-bridge uitgebreid, zodat complexere opstellingen mogelijk zijn die bidirectionele communicatie gebruiken. Een dergelijke opstelling zou een interessante toepassing zijn voor vervolgonderzoek.

De sorteermodule in zijn huidige staat is een volledig onafhankelijke toevoeging aan de Productie Cell die eenvoudig verwijderd kan worden. De module wordt aangestuurd door het hard real-time LUNA. Complexe taken worden verschoven naar het zacht real-time ROS, die op basis van deze algorithmes de aansturing in LUNA kan aanpassen via de luna-bridge. Door deze luna-bridge is het mogelijk de voordelen van een hard real-time te combineren met die van een zacht real-time systeem.

Contents

1 Introduction

An increasing number of machines is controlled by embedded systems (Ebert and Jones, 2009). An embedded system in the context of this thesis is a computer with the single purpose of controlling a certain device: whereas a general computer has a broad scope of tasks, the embedded system performs a pre-defined set of tasks and has a restricted user-interface.

Embedded systems are commonly used for controlling plants where reliable and safe behaviour is essential. Examples of embedded systems can be found everywhere, such as the air-bag control system in a car, the fly-by-wire systems in a plane or the radio-chip in a phone (Gupta and Chow, 2010). In certain applications, such as the air-bag, the embedded system must be able to react fast on changing conditions. In such an applications, hard real-time control systems are used to ensure responsiveness. A hard real-time system has a control loop running at a certain frequency which has to compute and execute its control commands within that time window (Broenink, 2015). By setting this frequency sufficiently high, one can ensure the control signals are always up-to-date and within the pre-defined margins. The frequency is limited by the computational power or architectural limitation of the platform.

At the University of Twente, a model of the plastic-moulding plant at Stork was made (van den Berg, 2006). This Production Cell has been used as a platform to test multiple hard real-time embedded control system implementations (Groothuis and Broenink, 2009). The Production Cell (PC) consists of six units, each with their own specific task. The units each have their own individual controller node and must be synchronised with each other using a supervisor node (Maljaars, 2006).

The usage of real-time control systems ensures a reliable control system, but limits the possibilities for additional tasks that can be performed. Extensive additional computations, such as image processing, exceed the time window of the hard real-time system. This is especially true when the hard real-time system is also space or energy constrained, such as in a drone (Kempenaar, 2014). A hard real-time control system in combination with a computational platform can extend the possibilities of the embedded systems while keeping the control of the actuators in hard-real time (Groothuis et al., 2008). The embedded computer handles the simple, high-frequency, hard real-time loop control algorithms, while more advanced computers are used for the execution of the more complex, low-frequency, supervisory control algorithms. An increasingly popular choice for complex robotic systems is ROS (Robotic Operating System) (ROS, 2015), since a lot of existing solutions are available for the platform. However, ROS is non real-time software running on Linux.

At the University of Twente a ‘bridge node’ is created which allows communication between ROS and LUNA. LUNA is an in-house developed hard real-time execution framework, which is used for the loop-controllers. With the ‘bridge node’, called luna-bridge (Bezemer and Broenink, 2015), it is possible to control actuators in hard real-time and update their control loop based on complex calculations done in ROS.

1.1 Project goals

The goal of this project is to demonstrate that it is possible to control an actuator in hard real-time while offloading complex algorithms to a soft real-time application using the luna-bridge. The Production Cell is used as a base for the demonstrator. The project consist of two parts: the creation of a complex algorithm in ROS and the design and hard real-time control of the actuator in LUNA.

The complex algorithm of choice is an image processing algorithm which processes the figures on the blocks of the Production Cell. The image processing is done in a ROS-node which uses the luna_bridge node to send commands to the embedded system.

The embedded system controls a sorting module in hard real-time, which has to be added to the Production Cell. The module is a new stand-alone unit for the Production Cell that can extract blocks from the belt. This new module has to be designed and build, for which the design trajectory is followed (Bezemer et al., 2012). The design trajectory uses a model-driven design philosophy, which focusses on the re-usability of modules and a certain toolchain (Broenink et al., 2010).

1.2 Report overview

A summary of the previous work related to this project is given in Chapter 2. The Production Cell in its previous state, as well as software related to this project, is introduced in this chapter. The design decisions are shown in next chapter, Chapter 3. First, the requirements are discussed, the mechanical design follows from these requirements. The software for both platforms is discussed in this chapter. The results of the new module are discussed in Chapter 4. Lastly, the conclusions and recommendations are presented in Chapter 5.

2 Background

The hardware and software that were available at the start of the project are presented in the following chapter. The design methodology as used in the RaM group, is introduced first. Using such a structured way of working has certain advantages, as explained in Section 2.1. The toolchain required to support the design methodology is shown in the next section. Finally, the Production Cell will be introduced. The new set-up is an add-on to the Production Cells base.

2.1 Design methodology

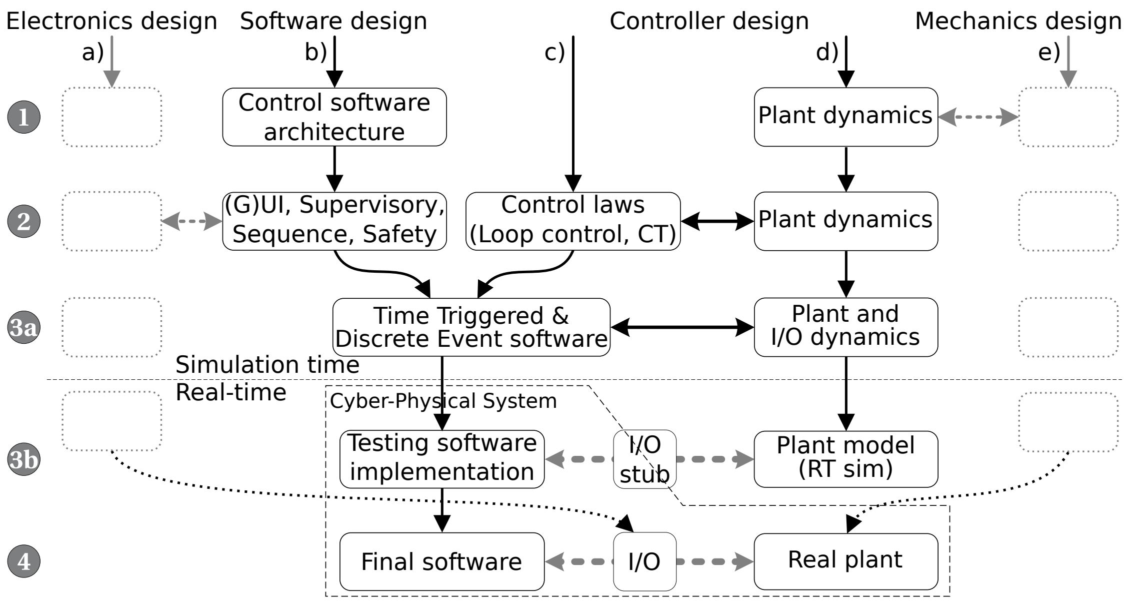


Figure 2.1: Overview of the design flow (Bezemer, 2013).

A design approach is used to manage the complexity of designing the software and the hardware. A model-driven design, as detailed in (Bezemer, 2013), is used to follow the best-practice way of working. The design flow is presented in Figure 2.1. In the case of this project, knowledge of all the displayed engineering fields is used to form an end-product. The thesis focuses on track (b) of the design approach.

In Figure 2.1, four steps are shown to achieve a working cyber-physical system. These steps are:

1. Software groundwork: Define the global overview of the loop-control software, see Section 2.1.1.

2. Algorithm design: Additional algorithms are designed in this part to handle high-level tasks. A division is made between tasks that require hard real-time algorithms and more demanding resource-intensive algorithms.

3. Verification and implementation: The control software is now complete. The plant model is enhanced with I/O dynamics such as in the real plant.

   (a) First the control software is tested in simulation time on the plant model. The control software can be improved in this step by further tuning.

   (b) Then the software is tested on the target system on a simulated plant in real-time. Simulation and test results are used to further improve the behaviour of the software.

4. Realisation: When all designs (Electronic, Software, Controller and Mechanical) function properly, the software can be deployed on the target system and tested.

The way of working utilises model-driven development (MDD) to reduce the complexity and allows the developer to cover multiple fields of engineering. Model-driven development requires the use of models to describe systems, so that these models can be used to perform tedious tasks that would otherwise require manual work by the developer. The MDD technique must be supported by the tools, so that a certain set of tools is used (the toolchain). Examples of tools supporting the MDD technique are 20-sim and TERRA, both of which are used in this project. Model transformation is needed to transform the models between tools, and model-to-code transformation is the final step to obtain control software which can be deployed on the control system. Components that are made using the MDD-approach should be reusable, in order to reduce design time in further research. Re-usability is increased if hardware-specific implementations are not used in the models. The ultimate goal of model-driven development is to create a first-time right application.

2.1.1 Software structure for embedded control

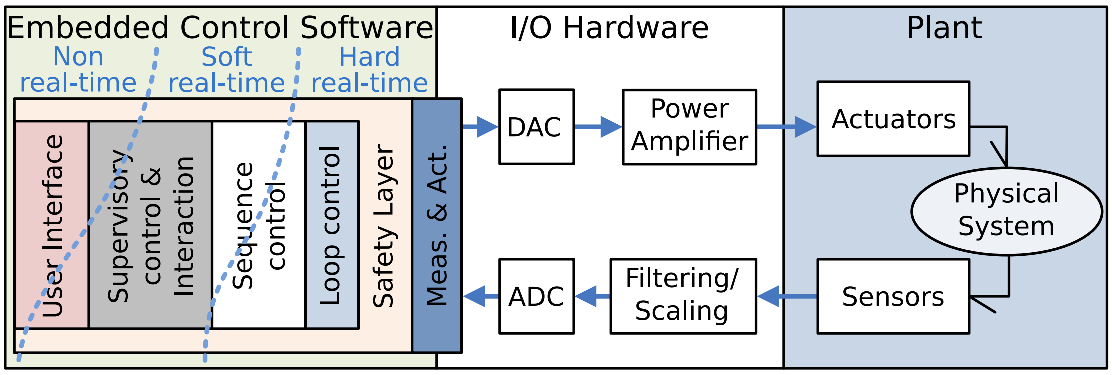


Figure 2.2: Layered embedded control software architecture (Broenink and Ni, 2012).

The architectural overview of the control software is shown in Figure 2.2. It is a layered pattern, dividing the plant, hardware and software. The software is also divided in a hard real-time, soft real-time and non real-time part. The pattern is inspired by (Bennett and Linkens, 1984).

The interaction of the hardware and software is handled in the Measurement Actuation layer. The layer consists of the necessary drivers and algorithms to drive the hardware, read sensor values and filter signals.

The Safety Layer is a surrounding layer of all embedded control software. The Safety Layer is the guardsman for the underlying layers and verifies if the values of the incoming or outgoing signals are within the set operating range.

The hard real-time software consists of the essential control software. Missing a deadline in this layer of the software can cause harm to either the hardware or its surroundings. Therefore, the Loop Controller is in this layer. The Loop Controller in turn, is controlled by the Sequence Controller. Whereas the Loop Controller has no knowledge of the system outside of its designated part, the Sequence Controller has an overview of the system and provides setpoints for the Loop Controllers. Whether or not the Sequence controller is hard real-time, depends on the application.

The Supervisory Control & Interaction layer consists of complex algorithms with long(er) calculation times. Examples of such algorithms are path-planning- and vision-software. The result of these calculations is executed by the Sequence Controller. Calculations of the Supervisory Control & Interaction layer are not essential for safe and stable behaviour, and this layer is therefore soft or non real-time. The same logic applies to the User Interface, and therefore falls in the same category.

2.2 Toolchain

The model-driven development approach requires a certain set of software packages (a toolchain), which support the MDD way of working. The toolchain is divided in two sections: the first part of the tools is required to get the real-time software running on the QNX platform. The second part is the software running in non real-time on a Linux platform. The two platforms are connected using the luna_bridge.

2.2.1 Non real-time platform

The non real-time or soft real-time system requires an installation of ROS (Robotic Operating System) to run the luna_bridge. ROS is a set of software libraries and tools to build robotic applications (ROS (2015)). It is open-source and includes a collection of tools and libraries. Central in the ROS ecosystem is the roscore, the central server application. The server is used to run 'nodes', ROS-specific applications. Nodes can publish their information to the roscore, or subscribe to a certain topic. Using these nodes, a network of complex algorithms can be build, with each node providing a part of the overall computations. ROS is inherently not real-time due to this publish-subscriber nature of the application, in which a scheduled timing of a node can not be guaranteed.

A node written in C++ subscribes to a fixed set of topics. The data structures of these topics are defined in the topic and checked by the compiler, so that the compiler can match the used fields with the defined field of a data structure. Users are alerted if an expected and defined data structures do not match, resulting in a robust design.

Despite the name, ROS is not an actual operating system (OS). ROS' operating system of choice is Ubuntu, which is used to operate the soft real-time platform. Ubuntu is a widely adopted Linux distribution, built on Debian's architecture. The Long Term Support (LTS) release is used, which improves the stability and durability of the system.

The system has to be able to perform image processing. ROS has no build-in library for computer vision, so that an external library has to be used. The OpenCV library is the most commonly used computer vision library and there are several ROS packages available that allow the use of ROS in combination with OpenCV (Developers Team, 2015). OpenCV's primary interface is C++, although interfaces in other programming languages are available. OpenCV is a free, open-source library.

2.2.2 Real-time platform

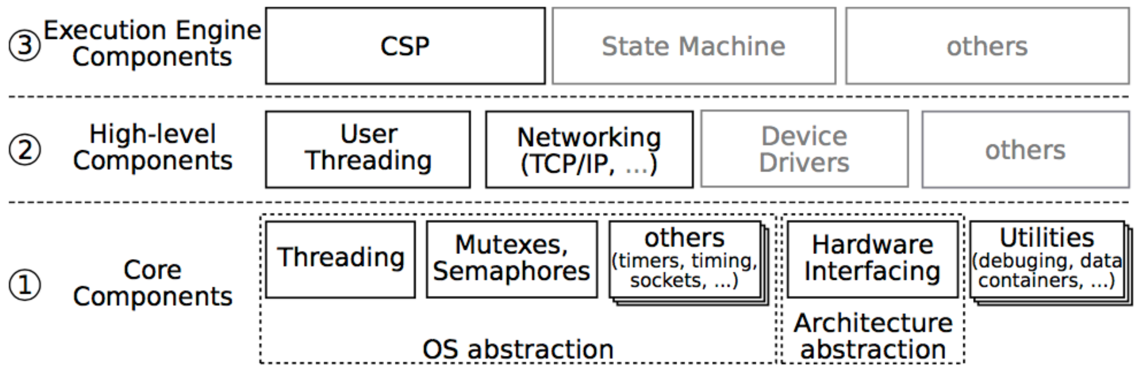


Figure 2.3: LUNA framework architecture overview (Wilterdink, 2011).

A hard real-time system requires a real-time operating system. At the RaM group, the QNX OS is typically chosen for real-time applications and is thus well integrated in the other tools. The luna_bridge can communicate with LUNA applications on the QNX platform.

LUNA Universal Networking Architecture (LUNA) is a hard real-time framework with multi-core support (Bezemer et al., 2011). LUNA is component based, Figure 2.3, level 2. Depending on the application certain components can be enabled or disabled. Due to the incorporation of an abstraction layer, LUNA is able to run on different platforms.

Furthermore, LUNA provides a Communicating Sequential Process (CSP)-execution engine. Using CSP allows the user to check for correctness, such as dead-locks, by using applications like FDR (Hoogendijk, 2013). Preventing deadlocks is essential in guaranteeing that processes are executed before their deadlines.

LUNA applications are built using a model driven design tool, called TERRA (Twente Embedded Real-time Robotic Application) (Bezemer et al., 2012). TERRA is based on the Eclipse framework and provides a model editor, model validation, code generation and support for external tools. The models are CSP-models from which C++ code is generated. The external tool support allows 20-sim models to be imported in TERRA. Both TERRA and LUNA are free, open-source tools, developed at the RaM group at the University of Twente.

20-sim is a commercial modelling and simulation program. Models in 20-sim can be entered in a number of fashions, such as bond graphs, block diagrams and equations. Since the underlying structure of 20-sim is based on bond graphs, it has great support for multi-domain dynamic systems. A typical plant can thus be simulated in 20-sim, so that a controller can be made. An add-on for 20-sim is available to transform models from 20-sim to TERRA such that models made in 20-sim can be used in TERRA.

2.2.3 Luna-bridge

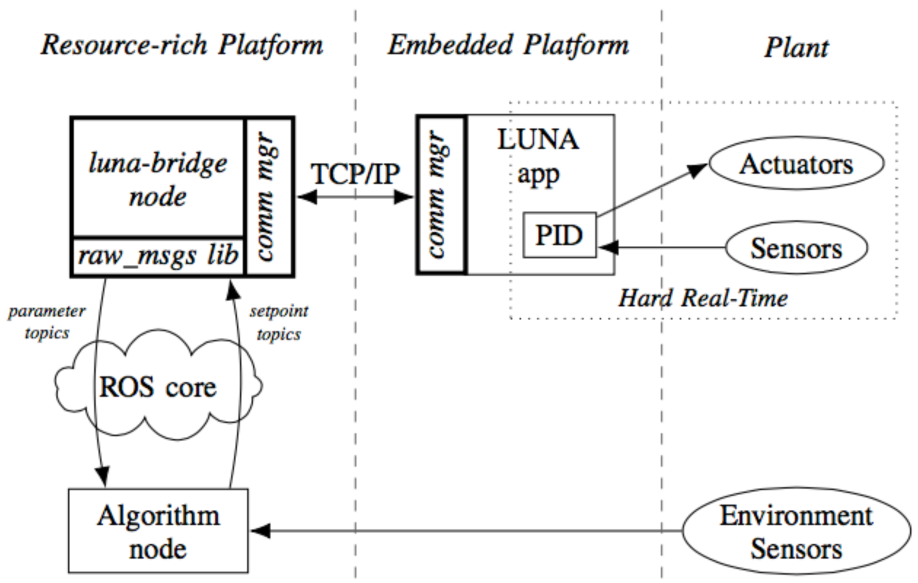


Figure 2.4: Architectural overview of the interconnected software platforms (Bezemer and Broenink, 2015).

The luna-bridge integrates the ROS and LUNA frameworks, so that a set-up as in Figure 2.4 is possible. In such an application, the luna-bridge runs as a node on the roscore. The LUNA application on the embedded system (Section 2.2.1) connects to this ROS node and can communicate with the ROS network explained in Section 2.2.1.

The default ROS subscription system, Section 2.2.1, prevents the luna-bridge from subscribing to topics with data sets that are not predefined in the code. This would require the user to re-compile the luna-bridge every time another topic or data type needs to be accessed. Therefore, a way has been found to bind to an available topic in run time. The details of this solution can be found in Bezemer and Broenink (2015).

The luna-bridge is under active development at the RaM group. The version used in this report is the first version, made by Bezemer (Bezemer and Broenink, 2015). This version currently supports boolean data transfer from ROS to LUNA. Progress is being made so that different data types can be send between LUNA and ROS.

2.3 Experimental setup

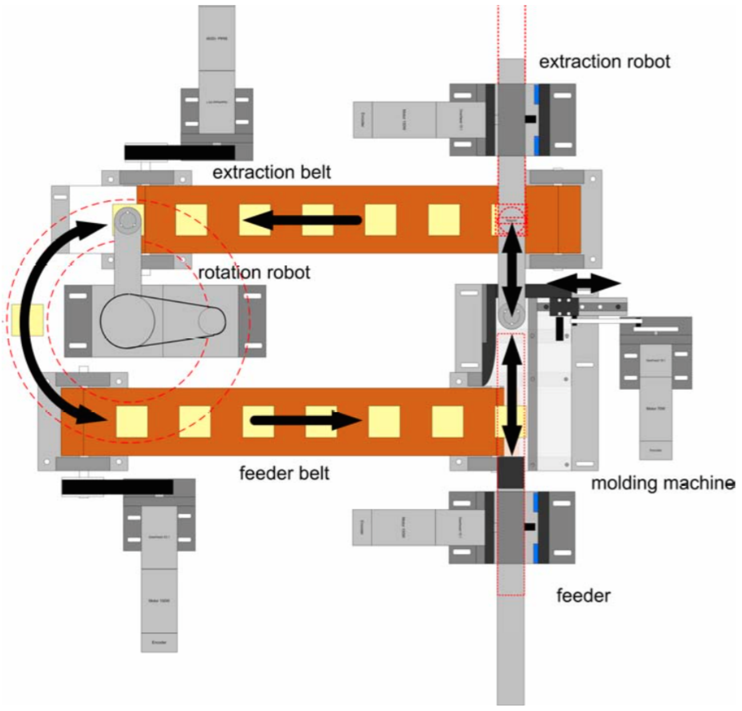


Figure 2.5: Overview of the Production Cell. The Production Cell mimics a plastic moulding plant, where the blocks represent the raw material. The blocks are circulated to pass all actuators (van den Berg, 2006).

The Production Cell is designed by Van den Berg (van den Berg, 2006) in 2005. The Production Cell is build to mimic a realistic plastic moulding plant by having multiple actuators that need to synchronize their activities.

The Production Cell consists of six motors. Five of these motors are 150W motors, operating at 24V. The moulding machine motor is a 70W, 24V motor. All motors are connected using custom-made H-bridges to the FPGA. The setup is build in a circular fashion, Figure 2.5. The blocks are recycled and pass all actuators. The raw material is represented by aluminium blocks on the set-up. The top plane of the blocks is replaced by a steel slice, which allows the blocks to be moved using a magnet.

Firstly, the blocks are fed to the feeder via the feeder belt. The feeder will push the block to the moulding machine door, which opens once the extraction arm is in position. The extraction arm will pick up the blocks using a magnet, and drop the blocks on the extraction belt. The block is then transported by the extraction belt to the rotation arm, which will pick them up to return them to the feeder belt. LEDs are used on the Production Cell to visualize the sensor data for demonstration purposes.

Different control algorithms have been tested on the Production Cell in combination with several Operating Systems, formalisms and tools. The Production Cell requires the actuators to operate in parallel, to cooperate and synchronize their activities. In this project, the setup is controlled by an embedded control system (PC 104) with a FPGA. A gCSP model with controllers and motion profilers imported from 20-sim is used to control the Production Cell (Groothuis et al., 2008).

3 Design

To verify that the luna-bridge works, a hard real-time control system must be integrated with a complex algorithm in ROS. Two systems are required for this demonstration: An embedded, hard real-time system and a resource-rich system. The following list of requirements is set up according to the MoSCow principle:

- The hard real-time embedded system must use LUNA
- The resource-rich platform must use ROS
- Existing demo's must still be usable after the new platform is installed
- The new platform must maintain the style of the current set-up
- The solution must be usable as a demo
- The new platform should be stand-alone
- The style could be visually elegant

The design of the mechanics, software, controller and electronics is based on this list. To satisfy the requirement of both a resource-rich platform and a demonstration set-up, the decision was made to add a camera to the set-up. Using the camera, the resource-rich platform can perform image processing on the passing blocks. Both the audience and the image processing software can identify the blocks by marking them with a certain pattern. Once identified, the block can be either removed, or remain untouched based on the pattern on the block.

3.1 Hard- and software overview

The hard- and software overview of the project is presented in Figure 3.1. The existing setup is shown in the top part, consisting of the original Production Cell as discussed in Section 2.3. The Production Cells sensors and actuators are connected to a FPGA card. The FPGA is installed in an Atom PC, running a Linux distribution (Debian 5). In the current implementation of the Production Cell, the Atom PC is used to start the Floating Point demo on the FPGA.

An overview of the new sorter module is at the lower, striped, part of Figure 3.1. The motor and encoder are connected to an AnyIO card installed on a PC104 running QNX 6.5. The QNX

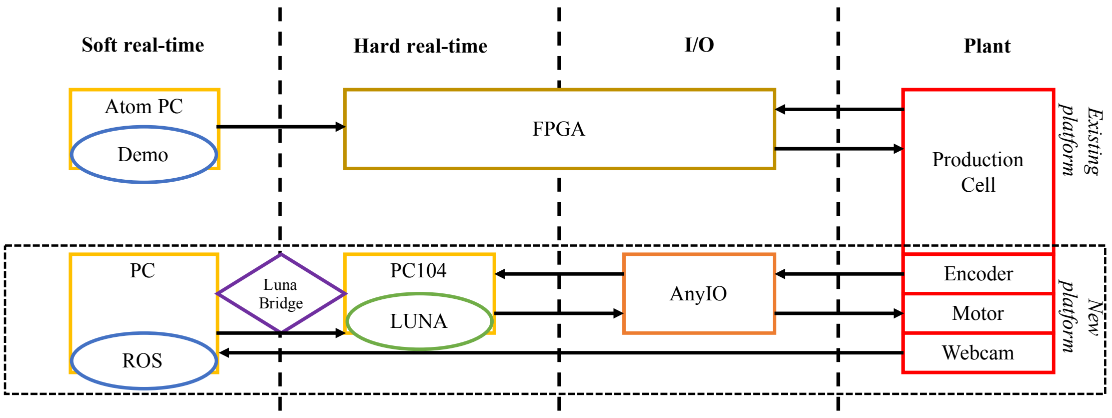


Figure 3.1: Overview of the hard- and software structure of the Production Cell.

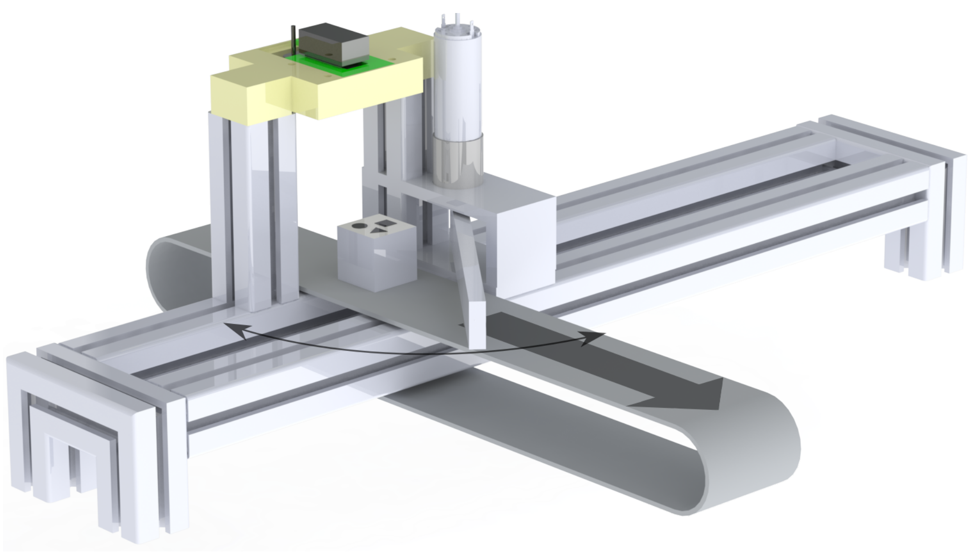


Figure 3.2: The new sorting module and the belt. The arrows indicate the direction of movement of the belt and the actuator.

control application is further elaborated on in Section 3.4. The webcam is connected to a development PC running Ubuntu 14 with the ROS Server application. The PC is able to communicate with the QNX application using the luna-bridge. The image processing software of the resource-rich platform is discussed Section 3.5.

Note that the structure of the new module matches that of a resource-rich platform which performs a computationally expensive task (image processing) and a small, power-constrained platform to control the motor as was introduced in Chapter 1. In the existing platform, the soft real-time platform is only used to activate the FPGA and display an interface to the FPGA. There is no offloading of tasks from the hard real-time platform to the soft real-time platform in the old situation.

3.2 Plant

The Production Cell needs to be modified to house the new hardware, consisting of a camera holder and the sorting module. The hardware additions and modifications to the current setup are presented in the following section. The sorting module will be build on a system profile which can easily be removed from the Production Cell to minimize the impact on the current setup.

3.2.1 Camera module

A webcam has to be placed on the Production Cell to provide a video-stream for image processing. A top-mounted position is chosen to ensure a constant background and minimal noise. Other positions would require advanced image processing algorithms to account for movement in the background. The positions also ensures the visibility of the shapes, since they are attached to the magnetized top-plane which is required to be facing upwards. The webcam mount can be fitted with LEDs to ensure constant lighting conditions. The frame must be sturdy enough not to move during normal working procedure of the Production Cell, since the image processing algorithm cannot separate camera movement from object movement.

The webcam is aligned with the belt in such a matter that it captures as much of the belt as possible. The minimal distance between the lens and the setup should be 100 mm in order for the webcam to be able to capture the whole contents of the belt. The belt itself is 85 mm and the blocks are 35 mm, so the webcam is placed at a height of 220 mm as seen from the Production

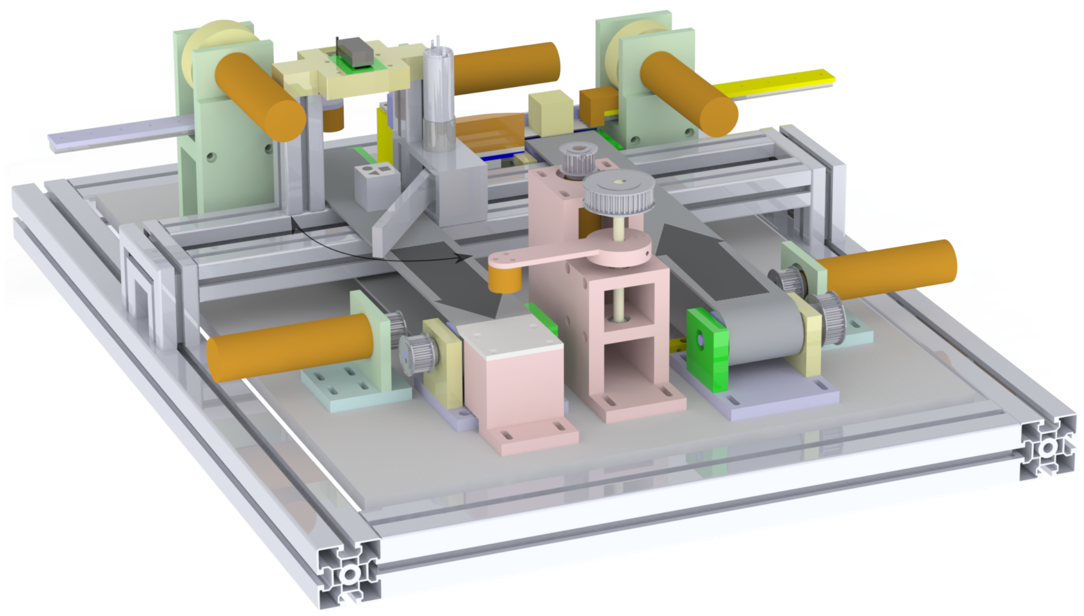


Figure 3.3: Impression of the sorting module and webcam holder installed on the Production Cell. The new hardware is installed on Boikon Aluminium profiles so no destructive adjustments have to be made to the existing hardware. Modified SolidWorks model of van den Berg (2006)

Cell. The webcam selection criteria are found in Appendix B. A 3D SolidWorks model of the camera module is presented in Figure 3.2.

3.2.2 Sorting module

Space constraints limit the placement possibilities of the sorting module, as can be seen in Figure 2.5. The module can either be placed on the extraction belt, after the camera module, or on the feeder belt. Placement on the feeder belt would require either a precise timing and constant behaviour of the Production Cell, or an additional sensor to find the block. Placement behind the camera module is more robust, since the distance between the identification and extraction is limited. However, placement close to the camera module requires a fast response of the controller and the image processing algorithm. Preliminary testing showed that the image processing was quick enough, so the motor is placed behind the camera module.

An aluminium profile from Boikon is used to create a plateau for the modules to be build on. The sorting module and camera module are placed on the same frame. An aluminium plate is used as a base for the sorting module and doubles as a raiser to level the rod to the right height so that it floats above the extraction belt. A SolidWorks drawing of both modules can be found in Figure 3.2. The Production Cell with this new add-on is depicted in Figure 3.3.

For the sorting module, a top-mounted motor with a rotating rod is chosen to guide the blocks from the belt. The top-mounted construction ensures sufficient space on the Production Cell and brings the shaft of the motor in the right position for the rod. The rod needs to move between a rest position and a 45° angle as seen from the blocks. In the rest position the blocks must go through unobstructed, while in the extraction position blocks are guided off the belt.

Alternative designs

One of the implementations that is not selected, is to use a long bar which pushes the blocks off the belt as sketched in Figure 3.4a. Advantages of the push-implementation are the ease with which end-stops can be applied. The controller can be a simple position controller, or even a pre-determined signal with an end-stop. Drawbacks of this implementation are the precise timing requirements and the speed of the pusher. The pusher needs to move at the

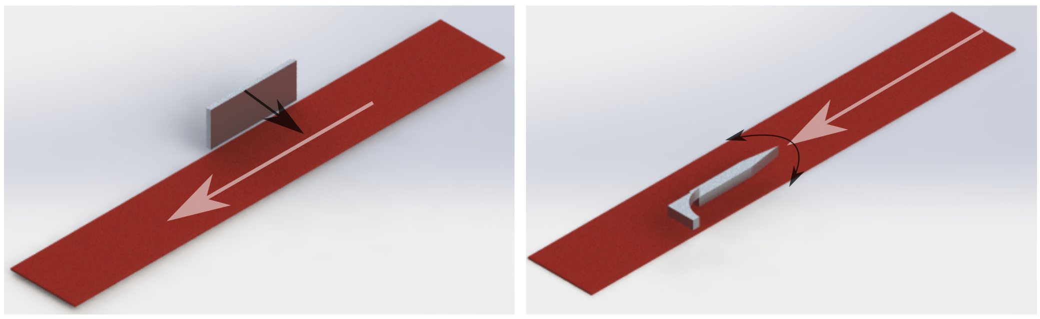

(a) Push blocks from the belt. **(b)** Sort blocks.

Figure 3.4: Possible hardware additions to the right extraction belt.

Table 3.1: Selection criteria for the sorting module implementation. More pluses is better.

Implementation	***Speed***	***Safety***	***Implementation***	***Control***	***Costs***	***View obstruction***
Push (Figure 3.4a)	++	- - -	+	+	+	++
Slide (Figure 3.2)	- -	++	+	+	-	+
Sort (Figure 3.4b)	++	++	- -	-	++	- - -

exact moment a block comes by. Since the Production Cell's main goal is to serve as a demo, safety is also important and this implementation would allow blocks to move off the Production Cell because of the limited control over the way the block leaves the belt.

Another implementation is shown in Figure 3.4b. A top mounted sorting mechanism is placed on the belt, which requires little movement to sort the blocks of the belt, or keep them on the belt. Another guiding mechanism might be required to rectify for misaligned blocks before the extraction. However, since this system is top-mounted, it will obstruct the blocks from view, possibly defeating the purpose as a demo system. The mechanical complexity of the implementation may also be a problem. An overview of the merits and drawbacks of each implementation is given in Table 3.1.

3.2.3 Mechanical design

The resulting system can be seen in Figure 3.5 and is build by hand using the mechanical design discussed in Section 3.2. The sorting module can be removed and installed on the Production Cell without permanent modification and without interfering with its usual behaviour due to the use of the Boikon frame (7).

3.3 I/O

The motor amplifier and encoder are connected to an AnyIO card, as presented in Figure E.2. The AnyIO card is a FPGA based PCI extension card which allows for analogue and digital signal processing on the PC104. The computation of the control algorithms is done on the PC104, after which the AnyIO card is again used to convert the signal to the PWM signal required by the H-bridge.

The motor amplifier used is an H-bridge designed by Bert van den Berg and integrally used in the Production Cell. It is capable of delivering up to 150W of power and requires a PWM input

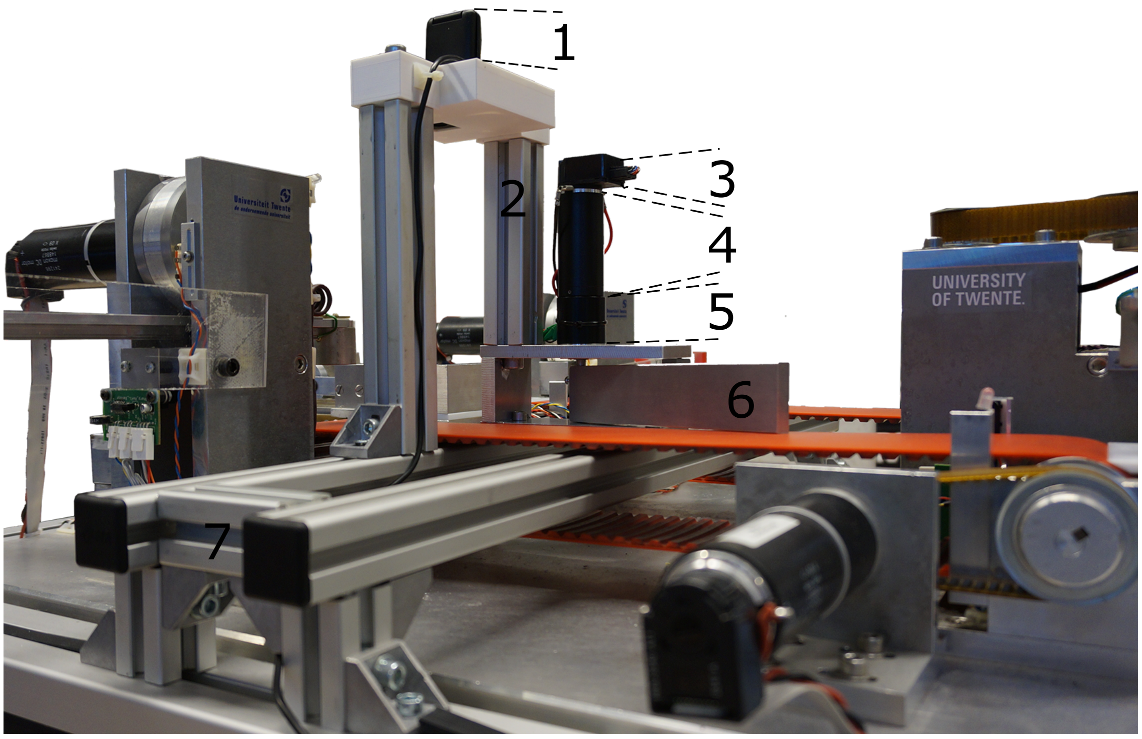


Figure 3.5: The new add-on module installed on the Production Cell: 1) Webcam; 2) Webcam holder; 3) Encoder; 4) Motor; 5) Gearbox; 6) Extraction rod; 7) Boikon frame.

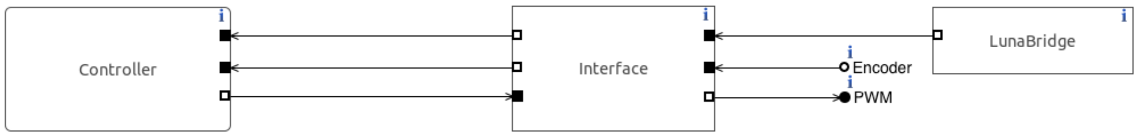


Figure 3.6: Architectural model of the real-time system. The LunaBridge can be seen as a port.

signal. By using the same specifications for the motor, encoder and motor amplifier, all parts are interchangeable within the Production Cell platform. The motor selection criteria can be found in Section B.1.

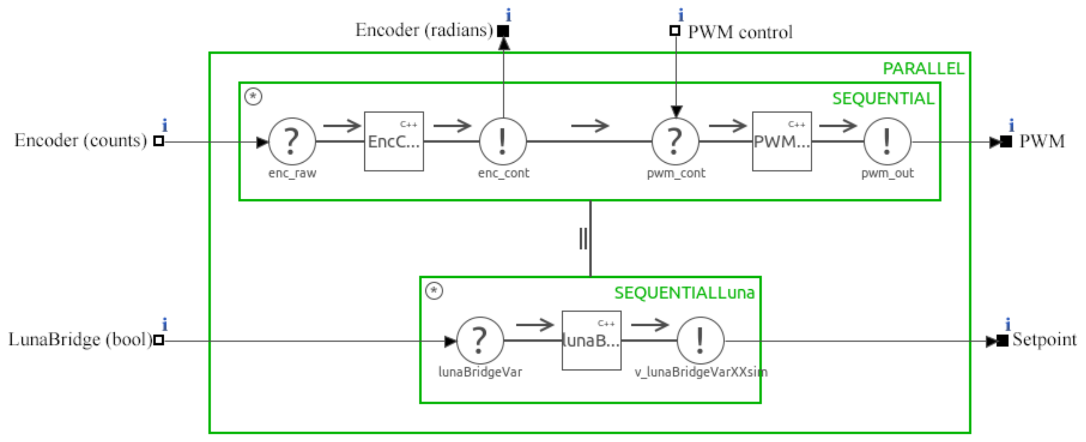
The AnyIO card requires a 16-bit unsigned integer for the H-bridge. A value ranging from 0 to $(2^{15}-1)$ results in an increasing PWM signal to go right. Increasing $(2^{15}-1)$ to (2^{16}) decreases the width of the PWM signal to go left, the motor does not move if the PWM value is either 0 or (2^{16}).

The encoder uses a quadrature signal: one turn of the motor results in 2000 counts. The encoder is directly connected to the AnyIO card. The encoder's initial position is set to 0 radians when the PC104 boots. Specifications of the motor, encoder and gearbox are presented in Section B.1.

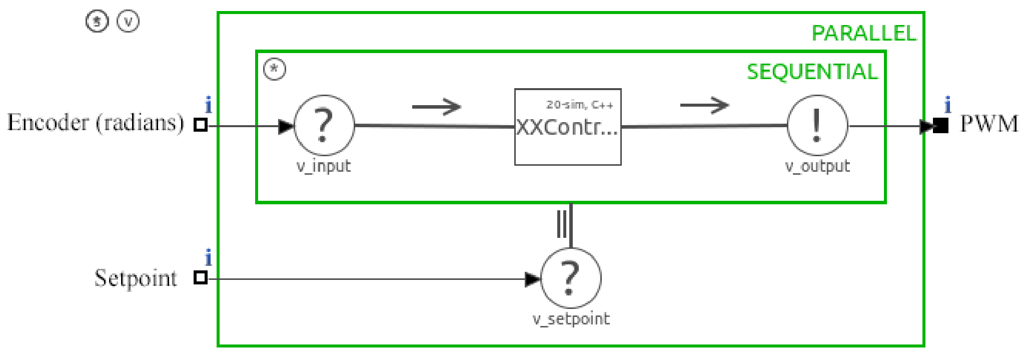
3.4 Hard real-time

The software needed for the sorting module consists of a hard real-time part and a soft real-time part, as introduced in Section 2.2. The hard real-time application receives setpoints from the soft real-time system using the luna-bridge.

The hard real-time platform runs a LUNA application which controls the position of the sorting rod based on the information it receives from the soft real-time platform. The architectural TERRA model of this application is shown in Figure 3.6. The application is divided into three



(a) Submodel of the interface.



(b) Submodel of the implementation of the 20-sim controller in TERRA.

Figure 3.7: Overview of the TERRA submodels. Black ports (sqare) are outgoing, white ports are incoming.

submodels: the controller, the interface and the luna-bridge. Three signals are required for the application to work: the setpoint, the encoder signal and the PWM signal.

Since the luna-bridge is a new addition to the TERRA-LUNA toolchain, it is not yet fully supported in TERRA. To use it, a LunaBridge submodel is made with modified drop-in code to connect to the luna-bridge. More information about the required modifications to the code and usage of the luna-bridge in TERRA is can be found in Section A.4. In practical use, this submodel can be seen as a port since no modification of the variable is taking place inside the model. As mentioned in Section 2.2.3, only one-way boolean communication is possible from ROS to LUNA in the luna-bridge used in this project.

The interface, Figure 3.7a, converts the raw number of counts to radians, and the output of the controller to a value for the H-bridge. A safety feature in the PWM conversion prevents the output from writing values if the encoder values are not within bounds.

Parallel to the conversion of the AnyIO signals is the conversion of the luna-bridge variable. All processes connected to the luna-bridge have to be run in a separate, parallel-recursive process to prevent deadlocks. The deadlocks are introduced due to the irregulars nature of the luna-bridge variables from the ROS-side. By running all connected blocks in a separate process, the controller can continue working without interruptions. A more elaborate explanation can be found in Section A.4. The booleans that the luna-bridge reads on the ports are converted to setpoints in the interface layer, so that a 1 results in an extraction setpoint for the controller and a 0 returns the sorting rod to its initial position.

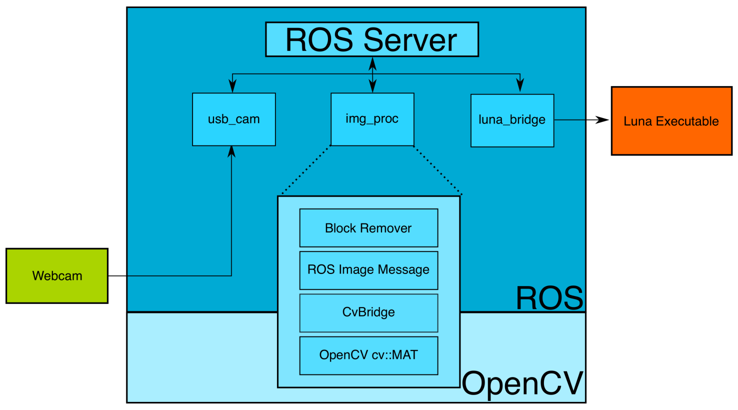


Figure 3.8: Software structure of the ROS-side of the project. The image processing will be done in the OpenCV block, which uses the information from the usb_cam node. Based on the information from the img_proc node, the block can be extracted. The command is send over the luna_node to a LUNA executable.

The controller is designed in 20-sim, see Appendix C. The controller submodel contains a C++-code block with imported 20-sim code and calculates the control signals.

Initially the controller was made to wait 0.3 seconds after receiving the remove signal, go the setpoint and return after 1 second. However, such timing capabilities are not supported in the conversion from 20-sim to TERRA. The capabilities can be added by modifying the C++-code generated from 20-sim, but would violate the MDD way of working. A workaround using the timer channel of TERRA was unsuccessful.

A new implementation for the removal is chosen which responds to the block that is currently visible by the webcam. The drawback of this implementation is that it is possible that the sorting rod is reacting to blocks that have yet to come. The result is that the block in front of the tagged block can be accidentally removed, or that the rod returns to its initial position before the tagged block is removed. The limitation is further discussed in Section 4.2.

3.5 Soft real-time

The resource-rich platform is connected to the webcam and performs the image processing in a ROS-node. The ROS software structure is centred around the ROS Server, or roscore, as explained in Section 2.2.1. In Figure 3.8 the ROS structure is shown. The figure shows that all communication between the nodes is done via the ROS Server and not directly between nodes.

The usb_cam (Pitzer, 2015) is an external node which allows any camera with Linux drivers to publish images from the webcam via the ROS server on the usb_cam topic. The node publishes the videostream of the blocks, which are processed by the image-processing-node. An overview of the ROS nodes is given in Figure 3.8.

The blocks are tagged with three shapes to distinguish them from each other. Each of the twelve block receives an combination of a square, triangle and circle. A total of nine unique blocks can thus be made, which are presented in Figure F.1. The shapes are chosen because they are easily recognized by humans, but not necessarily for the image processing software. QR-code recognition software, for example, is readily available and is very optimized and fast. QR-images also include a lot of redundancies, ensuring the right outcome. Going with more human-friendly shapes means giving up these kind of possibilities.

The shapes are attached to the top of the blocks, ensuring the camera is always able to see them since the steel top is the only magnetic part. The image-processing algorithm determines the shapes on the blocks to check if they meet the predefined conditions for removal. For example, all blocks with two triangles and one circle are to be removed.

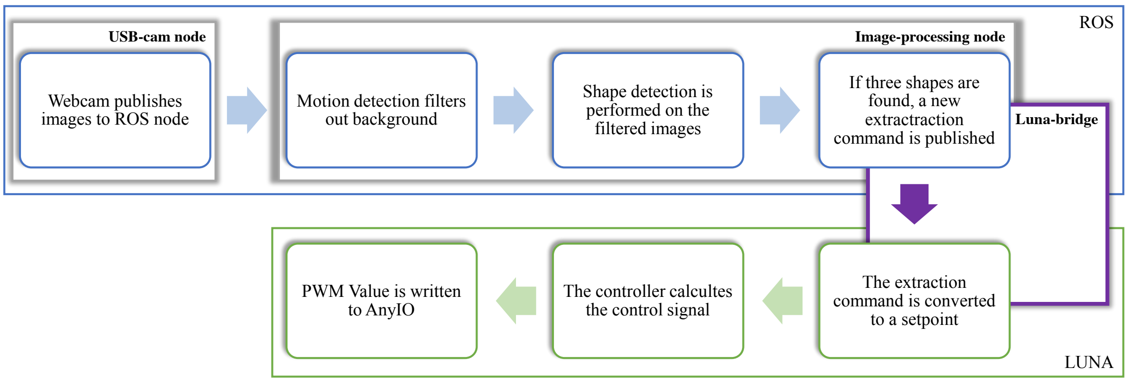

ROS
USB-cam node
Webcam publishes images to ROS node
Image-processing node
Motion detection filters out background
Shape detection is performed on the filtered images
If three shapes are found, a new extractraction command is published
Luna-bridge
The extraction command is converted to a setpoint
The controller calcultes the control signal
PWM Value is written to AnyIO
LUNA

Figure 3.9: The steps that are taken when a new frame arrives.

The basic workings of the image processing node are displayed in Figure 3.10. The img_proc node subscribes to the webcam image stream. The images that the node receives are in the ROS Image Message format, the CvBridge converts these ROS images to OpenCV images (Antahuja, 2011). The OpenCV images are processed, and based on the found shapes, the Block Remover decides if the block should be removed or not. The decision is send over the luna-bridge to the hard real-time platform.

The image processing node uses different classes, each with their specific task as can be seen in Figure 3.10. The process is first initialized, after which the image buffer class will be updated each time a new frame is published. The image buffer class stores the current frame in colour and greyscale, as well as the two previous frames in greyscale so that motion can be detected.

Once the image buffer receives a new image, the RoiSelection class will try to detect motion in the video. The class looks for differences in pixel values between the current frame and the previous frame, and the current frame and the second to last frame. Based on these frames the moving object is found and a Region of Interest (ROI) is drawn on the frame. The ROI image is then extracted and send to the pattern recognition class.

The pattern recognition class uses contour finding to find how many corners are on the figures on the blocks. The image is first further processed and eventually turned into a binary, black and white, image. Then the number of corners made are counted, and it is determined if the objects are squares, circles or triangles. The result is sent back to the main program and published on a ROS node. A detailed description of the image processing node can be found in the documentation of the node.

The block remover class is initialised if three shapes are found, and checks if the found shapes match these of the shapes set in the class. If the shapes match those of the shapes marked for removal, a TRUE boolean is published to the ROS server. The luna-bridge is subscribed to the topic and starts the block removal procedure. If the shapes do not match the predefined set of shapes, a FALSE boolean is sent.

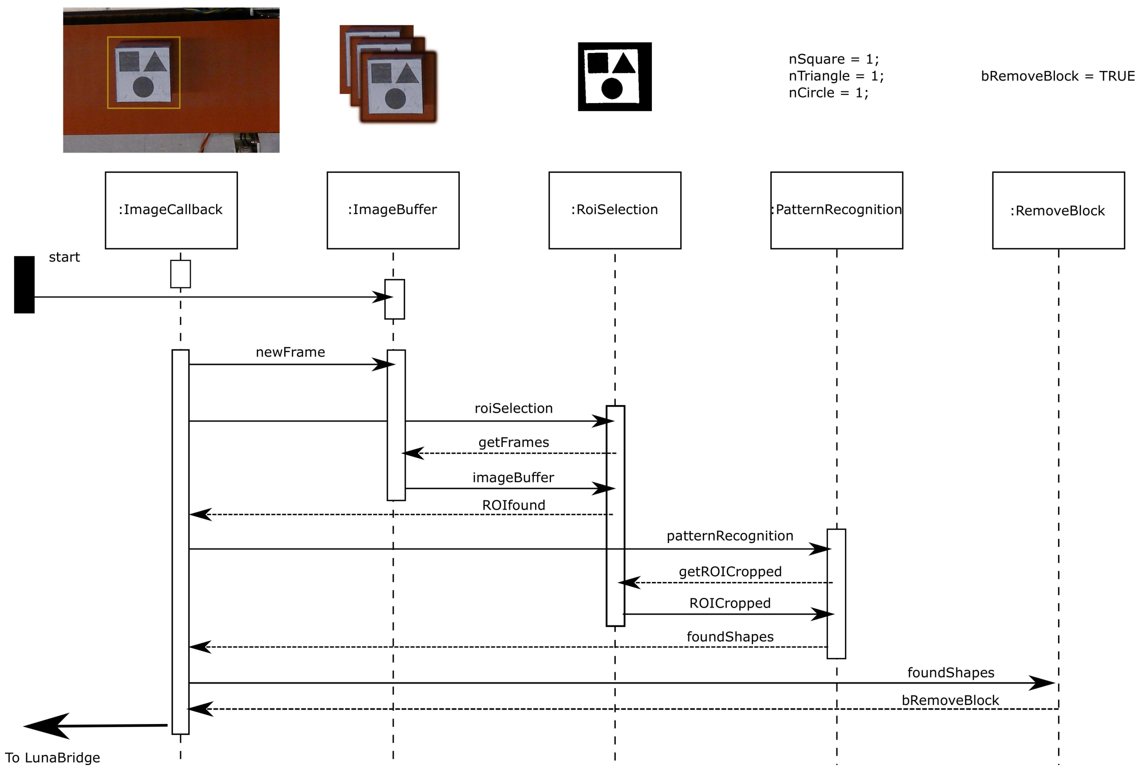


Figure 3.10: Sequence diagram of the node. The program initializes a ImageCallback class which get called each time a new frame is published. The class then updates the buffer so that the last three frames are stored in the ImageBuffer class. These frames are used in the RoiSelection class to find movement and determine the region of interested (ROI). The ROI is extracted from the image and is used in the PatternRecognition class to find the shapes of the figures of the block, which is returned to the ImageCallback class. If three shapes are found, the shapes are sent to the RemoveBlock class which decides if the block is to be removed. If no activity is found, the PatternRecognition and RemoveBlock class are not invoked.

4 Experiments and measurements

The performance of the sorting module depends on two factors: the image processing speed and the responsiveness of the sorting module. They will be discussed in the following chapter.

4.1 Image processing

Reason Ideally, the speed of the image processing module is limited by the refresh rate of the webcam. The used webcam has a refresh rate of 30 frames per second: one frame must be processed within 33 ms.

In this time window, the figures of the shapes must be correctly identified. A wrong identification of the shapes can result in the unjust removal of a block or unmoved blocks which are set to be removed.

Set-up To test the speed of the image processing algorithm, the Production Cell is loaded with six blocks. One of these blocks is set to be removed, the others are left to be untouched. The blocks are evenly distributed along the belts, so that there are no limitations due to the speed of the sorting rod. The experiment is finished when all blocks have passed the webcam once, and is repeated five times. During the experiments, the webcam outputs 640*480px images to the image processing algorithm. The specifications of the PC which runs the image processing algorithm, is presented in Section D.3.

To analyse the speed of the image processing algorithm, the software is modified to write the computation time to an output file. The computation time is calculated by subtracting the system time at the end of the frame analysis with the system time at frame arrival. The found shapes are also written to this output file, so that image processing time can be split apart in the time it takes to process an empty frame and a frame with contents. By combining the found shapes output with a video that is recorded during the experiment, it is possible to check if the shapes are correctly identified.

The output file is analysed with Matlab. The data is split into two sets: One set which contains all frames where three shapes are found on a block, and a set which contains the rest of the data. The mean and standard deviation of the computation time is calculated for each set. Combining the found shapes with the video recordings allows for a manual comparison with the actual shapes, so that an estimate of the accuracy can be made.

Results The average computation time over a whole run is 18 ms per frame. In this time, the whole image processing algorithm as shown in Figure 3.10 is executed. When three shapes are found on the image the computation time is 18.4 ms, well within the time window of 33 ms. When no shapes are found, or when there is no block on the belt, the computation time is 17.9 ms. The results are summarized in Table 4.1.

Per block passing under the webcam, an average of 9.25 ± 1 frames are captured that contain the shapes. The shapes of the blocks are isolated in an average of 8.47 ± 0.95 frames. This means that 91% of the frames that can be identified, are used. The resulting 9% of the frames is wrongly identified as having either more or less shapes than three. Of the frames where the shapes were identified, the shape recognition worked with 100% accuracy.

Since the results are only processed if three shapes are found, the sorting module was able to extract the marked block in all runs with 100% accuracy based on three runs, having identified 36 blocks.

Discussion The success of the image processing algorithm depends largely on the contrast of the shapes with the surroundings. Therefore, the stickers on the blocks should remain clean

Table 4.1: Speed and accuracy of the image processing algorithm.

Frame contents	Computation time (s) ($\pm\sigma$)	Accuracy
Less than three	18.0 ± 2.5	N.A.
Three shapes	18.4 ± 3.3	91.5 %
Total data set	17.9 ± 2.6	N.A.

and white, while the shapes should be as black as possible. The threshold algorithm which converts the grayscale image to a black and white images is sensitive for fading shapes, which can result in 'bites' missing in the shape.

The parameters for the shape recognition are strict, so that the a vague shape is usually rejected before being wrongly identified. Shapes missing certain sections are therefore not recognized, so that the number of found shapes does not equal three, and the block is not further processed. In Figure 3.10, this corresponds to the datastream of foundShapes to RemoveBlock not taking place.

The additional shapes are due to the square shape of the blocks, which is sometimes wrongly interpreted if the region of interest detections fails to extract only the block.

4.2 Sorting module

4.2.1 Extraction speed

Reason In Section B.1 it is discussed that the motor must be able to move from one position to the other within 0.4 seconds. The controller in Appendix C is able to do so in 0.3 seconds in the simulation. Using this controller on the new sorting module, the execution time of the sorting action can be calculated.

Set-up To test the extraction time of the system, the Production Cell is loaded with a marked block and another block, spaced in such a way that the speed of the extraction module is not a limitation. The software on the QNX system is modified to write the encoder and PWM values to an output file.

Results The output file of the QNX system is imported in Matlab. Since each datapoint in the output file corresponds to 1 ms, the data can be converted to a time in seconds. The results of the controller on the plant are given in Figure 4.1.

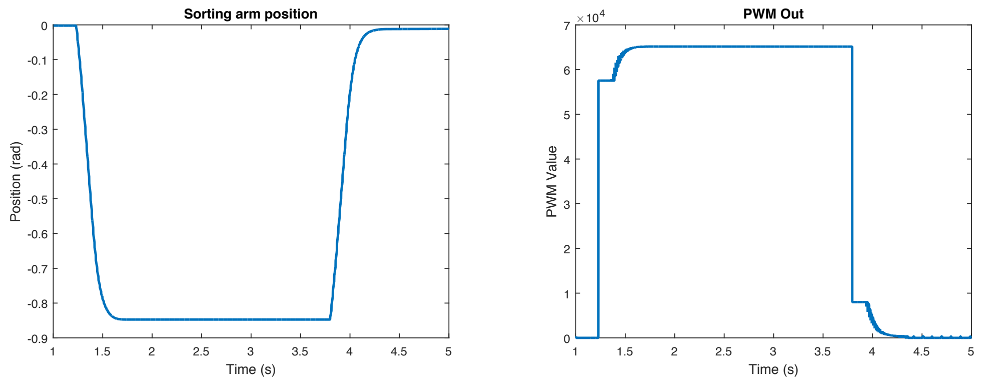
Figure 4.1a shows that the implementation of the controller on the plant is able to move the rod in 0.4 seconds, without overshoot. The discrepancy between the model and the plant is due to a stricter limitation on the PWM output, as can be seen in Figure 4.1b. The PWM output is limited so the QNX is able to print status updates without missing deadlines. Both the PWM and encoder output show a slow movement in the last 0.1 second, so that no overshoot occurs, but the rod is in place.

Discussion Although reaching the full setpoint takes 0.4 seconds, a suitable extraction position is achieved after 0.2 seconds. At this time, the controller is already at 85% of the desired position and can safely extract the blocks, since it will move slowly from that time on.

Due to the limitations of the timing in the controller, explained in Section 3.4, the sorting rod is always in time for extraction. The rod arrives 0.3 seconds before the block is at the rod.

4.2.2 Limitations

Reason Besides the execution time after the extraction command is given, the practical maximum load of the system has to be found. The blocks move over the belt with a speed of 70 mm/s. Combined with the total time of an extraction, this puts a limit on the distance be-



(a) Postion of the sorting arm in time. **(b)** PWM Signal from the controller.

Figure 4.1: In- and ouput of the controller after moving the sorting module to the extraction position and back.

tween the blocks. If two blocks are too close to each other, the sorting rod will either push off the block in front of the desired extraction block or return to the initial position before the block is removed

Set-up To test the minimal time between the blocks to ensure a safe removal, the Production Cell is loaded with three blocks. The first and last block are blocks which must go on unobstructed, while the middle block must be removed.

In the experiments the distance, and thereby the time, between the blocks is increased until a distance is found where the extraction is successful. A successful extraction is an extraction where only the middle block is removed. The blocks start 1 cm apart (minimal distance for the optical sensor of the Production Cell to tell the blocks apart) on the feeder belt (Figure 2.5). The rotation robot then feeds the blocks in a constant manner to the sorting module. The rotation robot is the limiting factor of the speed of the Production Cell, so the sorting module should be able to extract blocks at that pace.

The experiment is recorded using an external webcam. The distance between the blocks is increased by 5 mm after each failed extraction, until a successful extraction is accomplished. Thereafter, the distance is decreased again in small steps until the limit is found. Once the limit is found, the experiment is repeated five times in order to validate the found limit. The external recording is used to analyse the results.

Results The minimal distance between the marked block and the block in front on the feeder belt should be 4.5 cm. Anything below this value will cause the sorting module to push off the block in front of the marked block.

For the block behind the marked block, there is no minimal distance on the feeder belt. The marked block is removed before the feeder can produce the block behind the marked one. The theoretical minimum distance between blocks on the extraction belt should be 5.5 cm, but during normal operation this distance is greater due to the extraction robot's speed.

Discussion In order to remove a block safely, the distance between the blocks should be 4.5 cm on the feeder belt. In practice, this distance is reached during normal operations of the Production Cell.

However, if the Production Cell is fully loaded, a queue is formed at the feeder. When the moulding machine operating at full speed, the distance between the blocks on the extraction belt can sometimes get too small for safe removal.

If a queue is formed at the feeder belt which consists of the marked block in it, it is possible for the sorting module to move too fast to its extraction position, thereby knocking the unmarked block off the belt. If the sorting module is to be used safely, the minimal distance between the blocks should be kept at 4.5 cm.

4.3 Total response time

By combining the time from Section 4.1 and Section 4.2 the total response time can be calculated. The total response time is the time it takes the whole set-up to identify and extract the block, starting from the moment a block is in sight of the webcam.

It takes up to five frames for the webcam to get a full shot of the block. Once the whole block is in sight, a total of seven to ten frames can be obtained which display all the shapes. The processing of these frames takes 20 milliseconds. After the identification, an extraction signal is sent over the luna-bridge, which can be considered instantaneous.

Once the extraction signal is received, it takes the controller 0.4 seconds to reach the setpoint. Combining these numbers, the total response time from detecting movement to having the sorting module to the setpoint is 0.6 seconds. Response time up to the point of being able to safely remove the block is 0.4 seconds. Once the sorting rod is at its setpoint, it takes another 0.4 seconds for the block to arrive at the rod. To have to block removed from the belt takes another second, so the total time of an extraction is 2 seconds.

5 Conclusion and recommendations

5.1 Conclusion

A new demonstration module for the luna-bridge is designed and realised. The sorting module is controlled in hard real-time using LUNA, while offloading the complex algorithms to ROS using the luna-bridge. The image processing node of ROS is able to determine the shapes of the figures on the blocks. Based on these shapes, a command is send over the luna-bridge to the hard real-time QNX system.

The hard real-time system is able to convert the signal from the luna-bridge to a setpoint and use the new sorting module to remove the marked block from the belt. The sorting module is quick enough to return to its initial position to let unmarked blocks pass through unobstructed.

The new sorting module extends the functionality of the plant while maintaining the style, electronics and mechanics of the Production Cell. The new system as a whole can be operated fully independently from the Production Cell, and vice versa.

The capabilities of the luna-bridge can also enhance other platforms besides the Production Cell. Many computationally-limited applications can benefit from offloading complex algorithms while maintaining the essential control algorithms on the application itself. Systems such as drones and mobile robots can maintain a low energy profile and safe control while extending the capabilities of the system.

Concluding, a new stand-alone module has been added to the Production Cell. The new system uses a hard real-time LUNA application to control the sorting module, while offloading the image processing tasks to the soft real-time ROS. ROS is able to write setpoints to LUNA over the luna-bridge, combining the benefits of a hard real-time and a soft real-time system. This demonstrates that the luna-bridge functions as it was intended.

5.2 Recommendations

TERRA-LUNA The TERRA-LUNA software combination is a very powerful tool, but there are some caveats before one can start using it. The installation documentation is minimal and requires substantial knowledge of Linux. This is especially the case for TERRA, which depends on many plugins and extensions with a specific version from Eclipse in order to run properly. Documentation is also a problem of the luna-bridge, as it is absent. To speed up further research, a Virtual Machine image is made with all the necessary tools pre-installed and configured correctly. Also, documentation on the usage of these tools is given in Appendix A.

At the start of the project, the luna-bridge was not yet ready for deployment. It was unstable, and there were many bugs in the code which needed to be found and fixed. The luna-bridge itself is still very limited in its functionality. At this moment a student is working on implementing bidirectional communication, better implementation in TERRA, as well as support for more data types. These improvements are necessary to make the luna-bridge a useful contribution to the research chair.

The code conversion from 20-sim to TERRA is also limited in its usage, preventing one from fully utilizing the Model Driven Design philosophy. For example, it is not possible to convert the PIDFF controller or a motion profile; which was used in the simulations, to a TERRA model due to limitations of the 20-sim export tool: not all 20-sim expressions can be code-generated. The timing can also be moved to another part of the process, for example to TERRA itself, or maybe even in ROS. There is no strict border between what needs to be implemented in 20-sim and in TERRA.

If a way can be found the add a timer channel or some way to delay the process, the number of blocks that can be processed can be greatly improved. In its current state, the blocks need to be 4.5 cm apart, but by adding a delay in the controller and immediately returning to its initial position after extraction, the time can be reduced.

Mechanical design of the sorting module The set-up itself needs to be made safer. The current implementation is risky, since a wrong initialization of the encoder position can permanently damage the set-up. One way to solve this is by adding end-switches to the setup, as is the case in the rest of the Production Cell. The benefit of adding end-switches is twofold: the position of the arm can be initialized properly and the power to the motor can be shut off if the arm hits the end-switch during operation.

Furthermore, the design of the set-up is bulky. The sorting module takes up a lot of space, especially the boikon frame is very present. If future work is done on the set-up, one could improve on the current design by minimizing the footprint of the whole system.

Mechanically, the mounting system of the rod to the motor can be improved. The current implementation is quick and easy, but the screw gets looser with every movement. Eventually, this will cause a backlash in the system and a sorting mechanism which will not move to the desired setpoint. A way to decouple the rod and gearbox shaft would greatly improve the mechanical design, but would probably increase the size of the system.

Software of the sorting module In the software, the shapes that need to be removed are hard-coded. An improvement can be made here, by making it possible to modify these shapes at run-time. A possible implementation is to subscribe to a ROS-topic and publish the desired shapes on that topic.

Also, the signal send over to the luna-bridge depends only on the last identified image of the block. Due to strict settings in the image-processing, this goes well in testing, but if the shapes on the blocks start to fade, a more robust identification is desirable. A blockRemoveBuffer class can be introduced which stores the last couple of found shapes, and if the majority of these shapes match, a more substantiated decision can be made to extract the block.

Future work In future projects, it could be possible to integrate the sorting mechanism more into the Production Cell set-up. Since the hardware used for the sorting mechanism and the rest of the Production Cell is similar, the software can be used on both platforms and is thus very modular. The functionality of the current Production Cell can be made up-to-date by using modern-day tools such as the one used in this project. Once the Production Cell is running on ROS and TERRA-LUNA, the sorting module can be tightly integrated in the whole set-up. Also, the functionality of the Production Cell can be modified and made more flexible if ROS is used as a way to offload heavy computations.

Finally, the application of the luna-bridge should be taken beyond the Production Cell. Many applications can benefit from offloading computationally expensive tasks while maintaining hard real-time control. The luna-bridge runs over a network, so that wireless communication is a possibility. After researching the speed and latency of the luna-bridge, it would be interesting to see the luna-bridge used in other than systems as the Production Cell.

A Instructions...

A.1 ...on starting the demo

1. Move the sorting arm to its initial position, in line with the belt. The initial position requires the blocks to move unobstructed. Once the PC104 boots, the encoder will be initialized with the initial position on boot set to 0.

2. Boot the Ubuntu PC and the QNX pc (in no particular order).

3. Connect the Ubuntu PC with the QNX system over LAN. Ensure the LAN cable from the PC 104 is connected to the Ubuntu PC. On the Ubuntu PC, go to network configurations (next to the clock) and select the PC104 connection setting.

4. Open a terminal on the Ubuntu PC and type: `terminator -l ROS`. Terminator is terminal application that allows the user to use multiple terminals in one window. If that does not work, open terminator and split the screen in 4 terminal windows, then type the following commands in one of the windows:

   - `roscore`
   - `roslaunch img_proc CameraLaunch.launch`
   - `rosrun luna_bridge luna_bridge`
   - `rosrun img_proc img_proc`

   These bottom two commands launch the luna_bridge and img_proc from their respective folder. The roslaunch command launches the webcam from the img_folder while the roscore starts the ROS server application.

5. Start QNX Momentics and launch the TopLevel program on the PC104.

A.2 ...on using the VirtualBox image

This VM contains a preinstalled version of ROS, QNX, Eclipse, OpenCV, 20-sim and some other tools to get started.

The VM is suitable for:

- Development of ROS nodes
- Development of TERRA applications
- Development of 20-sim models
- Conversion of 20-sim models to CSP-models
- Deployment of QNX applications
- Using the Luna-bridge

The host PC must install VirtualBox as well as the extension pack to enable all functionality (e.g. usb webcam support). The image is completely pre-configured, so that the steps in Section A.1 can be followed to start the demo. However, depending on your OS it may be necessary to change the location of the webcam in the CameraLaunch.launch file for which the necessary instructions are in Section A.3.

LUNA is installed in `/opt/LUNA`, pre-compiled LUNA installations for QNX and Linux can be found in `/opt/LUNA_INSTALL`. The installation in these folders contains a slightly modified

LUNA installation which must not be updated. If LUNA needs to be recompiled, one should use the following settings in the makemenu configuration:

- Target system: QNX
- Target Image> Install location: Change this location to prevent overwriting the working LUNA installation.
- Advanced Configuration > Use external toolchain > Use hosts toolchain > Compiler version: 6.5
- External libraries: > xxsim.
- High-Level > Link Drivers: Select all
- High-Level: device-manager and uthreading

The configuration for ROS can be found in Section A.3.

Eclipse and TERRA are set up to use these locations for compilation of QNX applications. QNX applications that are build are made for QNX 6.5. The output models of 20-sim can be found in `/home/rtsd/.wine/drive_c/temp`. A python script is supplied there to rename the contents of the Controller folder to the right format for TERRA conversion. The QNX project used in this project is located in `~/VosPJ/AllOver`. To run the executable on the PC104, the executable in `~/VosPJ/AllOver/TopLevel` should be dragged to QNX Momentics.

The ROS source files are located in `catkin_ws/src`. CodeBlocks is installed to modify the source code of the cpp files. `catkin_make` is linked against the LUNA-installation in the opt folder. The ROS bash file is sourced at boot.

Further documentation on the image, its usage and tools can be found on the desktop of the image and on the RTSD blackboard page.

A.3 ...on installing the essential tools for ROS on your own computer

It is advisable to use the VirtualBox image that is available at the RAM chair for development. If however, one wishes to manually install the required tools to develop the ROS node the following steps should be taken. The tutorial is written for Ubuntu.

1. Download and install ROS using the instructions found on the ROS site. Follow all install instruction so that the environment is set up.
2. Download and install OpenCV using the Ubuntu Software Center.
3. Download and install the usb_cam node, in a terminal type:
   `sudo apt-get install ros-ROSVERSION-usb-camera`.
   Replace ROSVERSION with the current ROS version, e.g. indigo.
4. Download and install LUNA using the instructions on LUNA website. Configure the LUNA compiler as follows, and make:
   - Target system: Linux-x64
   - Target Image> Install location: Choose a convenient location, e.g. `~/LUNA_INSTALL`
   - High-Level > Link Drivers > ros-channels
5. Softlink the luna_bridge to ROS by executing:
   `ln -s ~/LUNA/ROS/luna_bridge ~/catkin_ws/src`

6. Download the ros-raw_message node from `https://github.com/veger/ros-raw_message` and move the contents to `catkin_ws/src`.

7. Copy the contents of the `catkin_ws/src/img_proc` folder to the `catkin_ws/src` folder on your own installation.

8. Connect the webcam to your pc and edit the cameralaunch parameters:

   - Find the video source of your webcam by executing: `ls /dev/video*`
   - Go to `~/catkin_ws/src/img_proc/launch/CameraLaunch.launch` and update the `video_device` line with the right address.

9. Check if all tools are installed correctly by building the contents of the folder. In a terminal window, type:

   - `cd catkin_ws`
   - `source devel/setup.bash`
   - `catkin_make -DLUNA_INSTALLATION_DIR=~/LUNA_INSTALL/luna-linux-x64`

10. Follow the steps in Section A.1 to start the image processing.

A.4 ...on usage of the luna-bridge

The luna-bridge comes with LUNA. For installation of LUNA and the luna-bridge, see Section A.3 steps 4-6. This section will focus on implementing the luna-bridge in ROS and TERRA.

To implement the luna-bridge in TERRA, the following steps should be taken:

1. Make a model in the archmodel.

2. In the model, make a process ROSLinkBlock.

3. Add outgoing port to process block, name ROSvariable.

4. Add unit int(int), add variable ROSin of this type inside ROSLinkBlock.

5. Place a new writer 'ROSLinkWriter' inside process, connect to variable ROSin(type: int)

6. Connect ROSLinkWriter to port ROSvariable.

7. Generate code in Eclipse.

8. Overwrite with the drop-in code (found in `~/VosPJ/DropIn Code/`)

9. Edit the RosLinkBlock.cpp code to subscribe to the right luna-bridge channel.

10. Generate.

The dropin code can be found in `~/VosPJ/DropIn Code/`. Alternatively, the LUNAbridge model found in `~/VosPJ/LUNAbridge` can be included as an external model. However, if this implementation is chosen, one cannot edit anything in this model so that the model can be seen as a port. If the steps are chosen the model can be changed. All processes that are connected to the luna-bridge need to run in a separate process, parallel and recursive to the other processes as in Section 3.4.

To implement the luna-bridge in ROS, one should use ROS's publisher capabilities. Tutorials for this are readily available, but a short summary is given here.

1. To publish on the chatter_bool class, first create a node handler with that channel: `ros::Publisher removeblock_pub = nh.advertise<std_msgs::Bool>("chatter_bool", 2);`.
2. Create a message object: `std_msgs::Bool msg;`.
3. Add data: `msg.data = 1;`.
4. Publish to ROS: `removeblock_pub.publish(msg);`.

Once all programs are compiled, the luna-bridge node must run to enable to luna-bridge. To run the bridge, run the roscore and run: `roslaunch luna_bridge luna_bridge`. To check if it works correctly, run the luna_bridgeTest that comes with LUNA.

B Component selection

B.1 Motor selection

The sorting rod should make a rotation of 45°, or 0.85 rad, to switch between positions. Blocks are extracted from the moulding area each second. The image algorithm and sorting module combination must be able to identify, tag and extract a block within that time frame. A quick motor is required to move the rod from begin to end position within 0.4 seconds. The action must be executed within 0.4 seconds, to ensure the rod is in time for the block to move out of the way. The limitations of the timestep result in the accelerations and velocities given in Table B.1.

A model of the sorting module is made in 20-sim, Figure C.2, to calculate the inertia and power requirements of the motor. A motion-profiled PID-FF controller is used to ensure the combination is within the time limits that are set. The motor toolbox is used to test various Maxon motor-gearbox combinations that are within the calculated boundaries. A Maxon motor is preferred due their precision and well documented nature, as well as to satisfy to requirement to keep the new module in line with the current Production Cell. The combination must be able to deliver at least a torque of 1 Nm at 62 rpm.

The selected motor is a Maxon RE 30, in combination with a 28:1 gearbox and a 500 CPT encoder. The motor requires a 24V voltage supply like the other motors on the Production Cell, so that the same H-bridge can be used. The final combination is able to deliver a torque of 1.91 Nm and rotate at 250 rpm, which is safely within the required specifications.

B.2 Other components

The webcam used is a Logitech C270. It is selected for its driver support in Linux and its price. The camera is able to output 1280 x 720 pixel images at 30 frames per second. The focus distance is 30 cm, so that the belt and block are not in focus. However, since focus is not a requirement (the blocks are blurred in the algorithm) and the complexity and instability decreases with size the height is not further increased. Furthermore, it is possible to open the housing of the webcam and manually adjust the focus if need be.

The frame used to house the add-on is a Boikon aluminium profile. The Boikon profile is chosen because the existing Production Cell already uses a Boikon construction, so that the new hardware can be added without much modification. A complete list of all hardware that was added to the setup can be found in Section D.1.

Table B.1: Accelerations and velocities for the rod, where the displacement is calculated from the motor to the end of a rod with a radius of 200 mm.

Time (s)	Displacement	Acceleration	Max velocity
0.3	260 mm	13 m/s^2	1.3 m/s
0.3	1.3 rad	65 rad/s^2	6.5 rad/s
0.3	0.21 rot	10.3 rot/s^2	1.03 rot/s

C Controller design

The 20-sim controller is a PID controller. The controller is tuned for speed and minimal overshoot. A speedy controller is needed to ensure the sorting rod is in extraction position at the right time while minimal overshoot is required to prevent damage to the motor if the rod overshoots and hits the wall if it returns to its original position.

The sorting module is modelled in 20-sim, resulting in the model of Figure C.2. The Hbridge in the model is a linear system representation of the H-bridge and converts the PWM signal to an input current so that the 20-sim motor tool can be used.

The result of the controller is given in Figure C.1. The output signal is limited, to ensure motor operation is within the limits and the behaviour is correct. The controller is able to move the rod to its desired position within 0.3 seconds, as was required. The sig-saw pattern in the power output is due to simulated non-ideal components such as the transmission and the encoder.

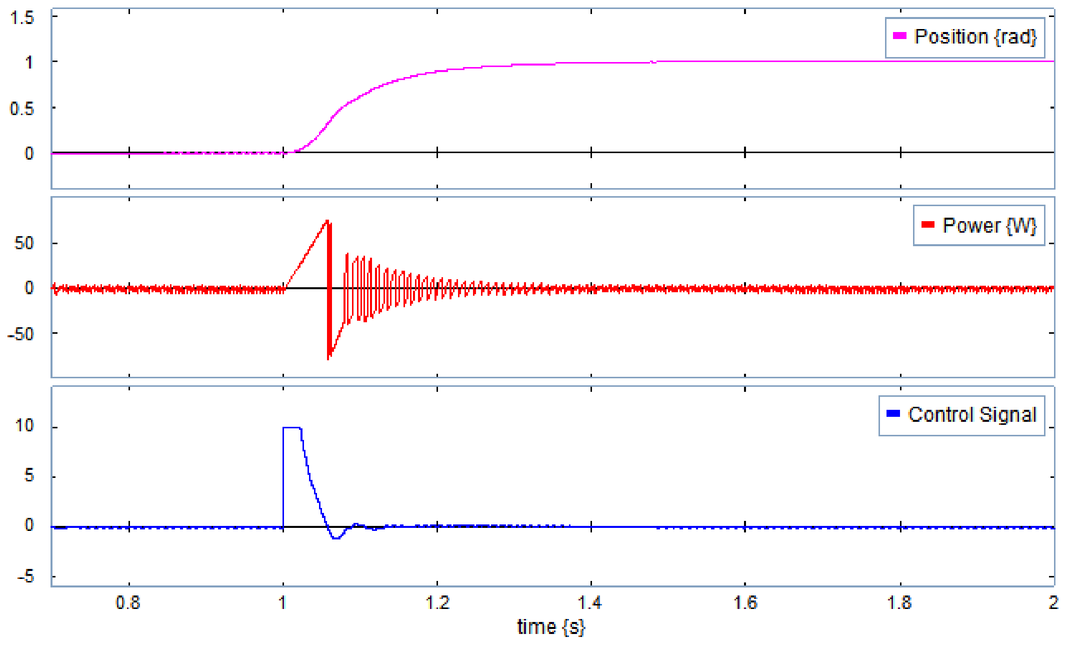


Figure C.1: Output of the 20-sim model.

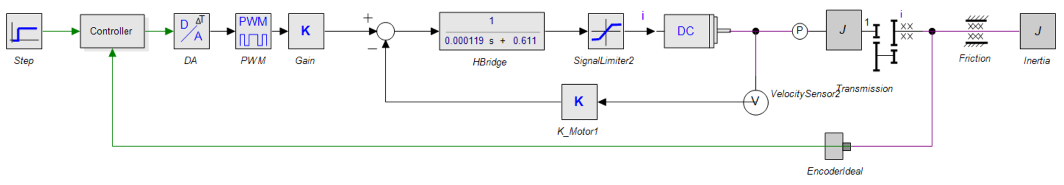


Figure C.2: Model of the sorting module.

D Components used

D.1 Ordered materials

Table D.1: Bill of Materials

Quantity	Article	Cost
1	Logitech C270 Webcam	27
2	Alu profile 30 x 30 (2x 740 mm)	11,66
4	Afdekkap 30 x 30	1,87
12	Hoekstuk 30 x 30	16,63
20	Sleufmoer 8E	11,70
4	Sleufmoer 8E	1,87
24	Cylindrische inbusmout M6x16	11,70
1	Maxon RE30, 24V, Part No. 310007	253,27
1	Maxon GP32, 28:1 Part No. 166162	137,55
1	HEDS 5540, 500 CPT, Part No. 110511	92,91

D.2 Software Packages used

Table D.2: Software used on the Ubuntu PC

Operating System	Ubuntu 14.04.2 LTS
Installed Software	ROS Server (Jade)
	Eclipse 4.4.2 LUNA
	TERRA 1.3.3
	LUNA (modified r1924)
	20-sim 4.5
	CodeBlocks
	usb_cam-node for ROS Jade
	OpenCV 2.4
	QNX Development Suite 6.5
	Wine 1.6.2
	Terminator 0.96

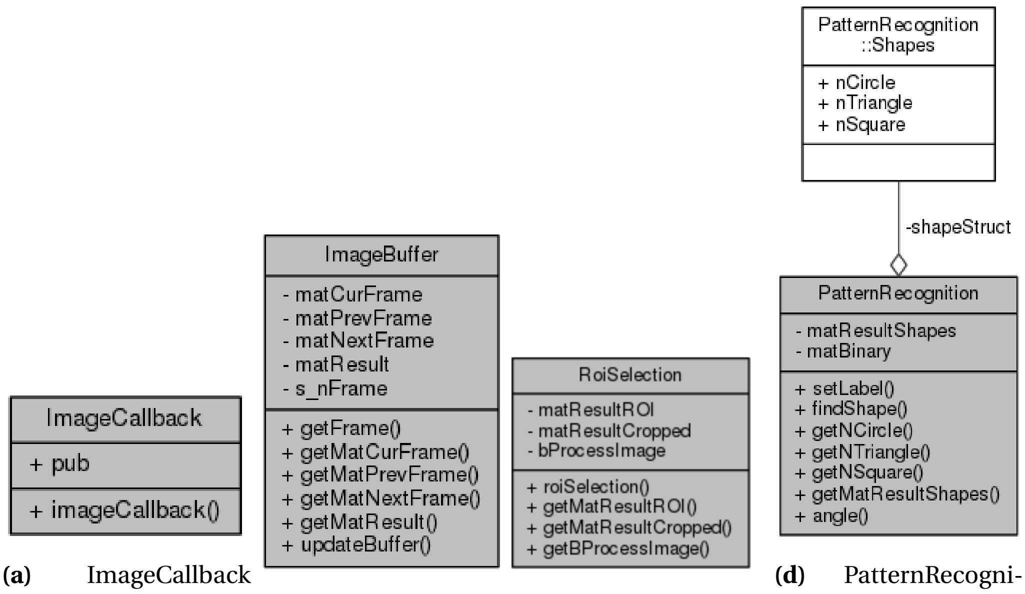
D.3 Specifications development PC

Table D.3: Specifications of the Ubuntu PC

Processor	Intel Core i7-4790
Memory	8 GB
OS type	64-bit

E Additional figures

E.1 Image processing classes



(a) ImageCallback class. **(b)** ImageBuffer class. **(c)** RoiSelection class. **(d)** PatternRecognition class.

Figure E.1: Diagram of the classes used in Figure 3.10.

E.2 Wiring scheme

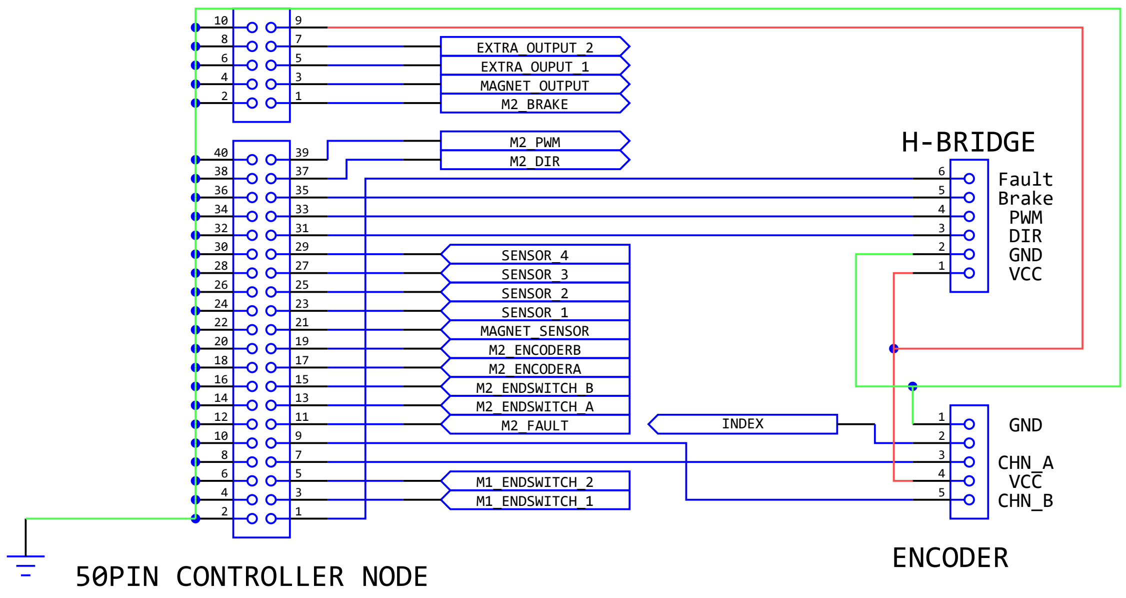


Figure E.2: Wiring scheme of the encoder en motor the the PC104.

F Shapes

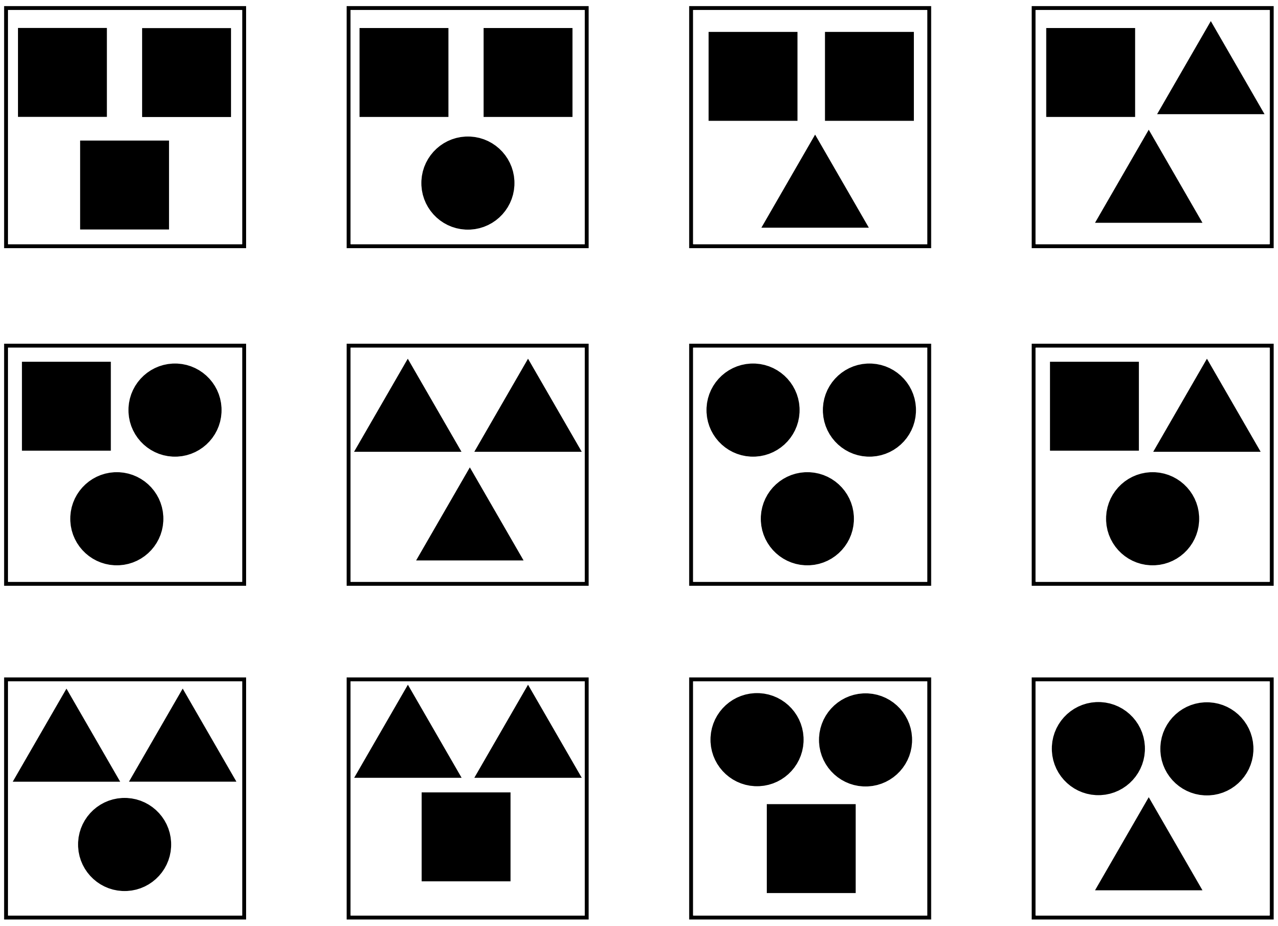


Figure F.1: The figure on the bottom left (two triangles, one circle) is hard-coded for removal.

Bibliography

Antahuja, S. (2011), Working with ROS and OpenCV.
https://siddhantahuja.wordpress.com/2011/07/20/working-with-ros-and-opencv-draft/

Bennett, S. and D. A. Linkens (1984), Real time computer control: an introduction, Institution of Electrical Engineers.

van den Berg, L. S. (2006), *Design of a Production Cell Setup*, Msc report 016ce2006, University of Twente.

Bezemer, M. M. (2013), *Cyber-Physical Systems Software Development - way of working and tool suite*, Ph.D. thesis, University of Twente, doi:10.3990/1.9789036518796.

Bezemer, M. M. and J. F. Broenink (2015), Connecting ROS to a real-time control framework for embedded computing, in *2015 IEEE 20th Conference on Emerging Technologies and Facotry Automation*, IEEE, pp. 1–6, ISBN 978-1-4673-7928-1.

Bezemer, M. M., R. J. W. Wilterdink and J. F. Broenink (2011), LUNA: Hard Real-Time, Multi-Threaded, CSP-Capable Execution Framework, in *Communicating Process Architectures 2011, Limmerick*, volume 68 of *Concurrent System Engineering Series*, Eds. P. Welch, A. T. Sampson, J. B. Pedersen, J. M. Kerridge, J. F. Broenink and F. R. M. Barnes, IOS Press BV, Amsterdam, pp. 157–175, ISBN 978-1-60750-773-4, ISSN 1383-7575, doi:10.3233/978-1-60750-774-1-157.
http://wotug.org/papers/CPA-2011/Bezemer11/Bezemer11.pdf

Bezemer, M. M., R. J. W. Wilterdink and J. F. Broenink (2012), Design and Use of CSP Meta-Model for Embedded Control Software Development, in *Communicating Process Architectures 2012*, volume 69 of *Concurrent System Engineering Series*, Eds. P. Welch, F. R. M. Barnes, K. Chalmers, J. B. Pedersen and A. T. Sampson, Open Channel Publishing, pp. 185–199, ISBN 978-0-9565409-5-9.
http://wotug.org/papers/CPA-2012/Bezemer12a/Bezemer12a.pdf

Broenink, J. (2015), Real-Time Software Development, University Lecture.

Broenink, J. F., M. A. Groothuis, P. M. Visser and M. M. Bezemer (2010), Model-Driven Robot-Software Design Using Template-Based Target Descriptions, in *ICRA 2010 workshop on Innovative Robot Control Architectures for Demanding (Research) Applications*, IEEE, IEEE, pp. 73 – 77.
http://www.rob.cs.tu-bs.de/en/news/icra2010

Broenink, J. F. and Y. Ni (2012), Model-Driven Robot-Software Design using Integrated Models and Co-Simulation, in *Proceedings of SAMOS XII*, Eds. J. McAllister and S. Bhattacharyya, pp. 339 – 344.

Developers Team, O. (2015), OpenCV.
http://opencv.org

Ebert, C. and C. Jones (2009), Embedded software: Facts, figures, and future, , no.4, pp. 42–52.

Groothuis, M. A. and J. F. Broenink (2009), HW/SW Design Space Exploration on the Production Cell Setup, in *Communicating Process Architectures 2009, Eindhoven, The Netherlands*, volume 67 of *Concurrent Systems Engineering Series*, Eds. P. Welch, H. W. Roebbers, J. F. Broenink and F. R. M. Barnes, IOS Press, Amsterdam, pp. 387–402, ISBN 978-1-60750-065-0, ISSN 1383-7575, doi:10.3233/978-1-60750-065-0-387.

Groothuis, M. A., J. J. P. van Zuijlen and J. F. Broenink (2008), FPGA based Control of a Production Cell System, in *Communicating Process Architectures 2008,*, volume 66 of *Concurrent Systems Engineering Series*, IOS Press, Amsterdam, pp. 135–148, ISBN

978-1-58603-907-3, ISSN 1383-7575, doi:10.3233/978-1-58603-907-3-135.

Gupta, R. A. and M.-Y. Chow (2010), Networked control system: overview and research trends, **vol. 57**, no.7, pp. 2527–2535.

Hoogendijk, T. A. (2013), *Design of a generic software component for embedded control software using CSP*, Msc report 014ram2013, University of Twente.

Kempenaar, J. J. (2014), *Communication Component for Multiplatform Distribution of Control Algorithms*, Msc report 001ram2014, University of Twente.

Maljaars, P. (2006), *Controllers for the Production Cell Set Up*, Msc report 039ce2006, University of Twente.

Pitzer, B. (2015), usb cam.
http://wiki.ros.org/usb_cam

ROS (2015), ROS.org | Powering the world's robots.
http://www.ros.org

Wilterdink, R. J. W. (2011), *Design of a hard real-time, multi-threaded and CSP-capable execution framework*, Msc thesis 009ce2011, University of Twente.
http://essay.utwente.nl/61066/