University of Southern Denmark
Faculty of Engineering
MSc in Engineering, Robot Systems - Robot System Design

A Multi-robot Production Facility for Packing LEGO Bricks

FINAL REPORT

Authors
Dominik Weickgenannt
Jorge Rodriguez Marin
Martin Groth Albøge
Steffen Ivan Nielsen
Thor Stærk Stenvang
Timon Tomás
Yonas Alizadeh

December 11, 2015

Abstract

Building reliable robot systems is essential to industrial applications, but can pose significant challenges due to the numerous components involved. The system as a whole and all its components, have to be remotely controllable in a user friendly way, capable of fulfilling the tasks they were meant to do in a fast and efficient manner. Furthermore, incorporate safety mechanisms to reduce hazard of damage both to the system and the operator(s), while being robust enough to recover from non-fatal errors which might occur during operation is needed. In this report the concept and implementation details of such a system is depicted, capable of processing orders issued by a central server unit and acting upon it by retrieving, delivering, sorting and packing LEGO bricks autonomously.

Contents

Project Description

The goal of the project is the development of a robot production facility that packs LEGO orders for a company. The LEGO orders describe a number of particular bricks in different colors and shapes, which are to be selected and packed for delivery.

The system consists of the following elements, performing the tasks listed below:

- Manufacturing Execution System (MES): The central server managing order processing and scheduling.
- Robot cells: Sorting and packing the LEGO bricks.
- Mobile robots: Transporting unsorted as well as sorted and packed LEGO bricks.
- LEGO brick dispenser: Supplying the LEGO bricks.
- Marker locator: Positioning system for tracking the mobile robots.
- Safety system (Pilz Safety Eye): Accident prevention.

The connection of these system elements is shown in figure 1.

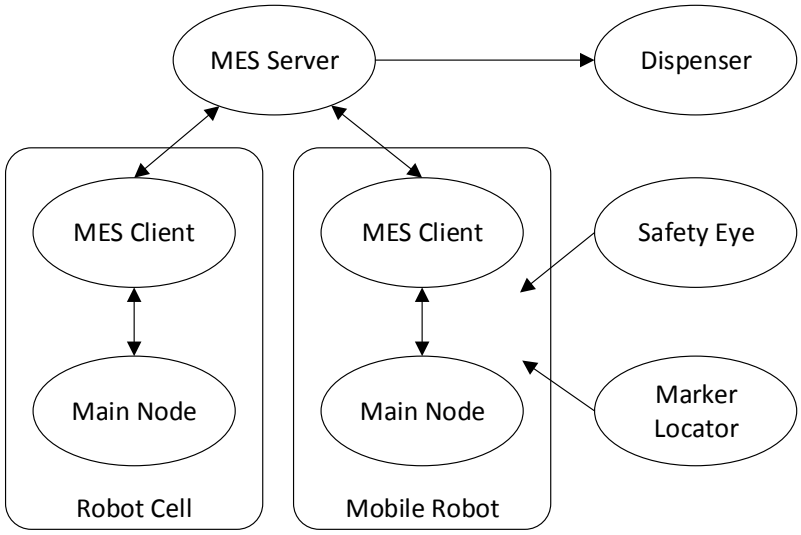


Figure 1: *Overall connection of the system elements*

Chapter 1

Project Management

It was a requirement to use the agile product development methodology called SCRUM for giving shape and bring order to the organization. In the following pages discuss how SCRUM was used through the development process.

1.1 SCRUM

The Agile methodology suited this project since the iterative development has a good synergy when it comes to customer collaboration and changes in requirements. Two aspects in the system development was influencing the defining of tasks. The first aspect is system requirements were not completely finalized from the beginning. The second aspect is a lot of new unfamiliar technologies and tools made it hard to get a breakdown of the processes required to complete the system.

The project started the 7th of September 2015 with a introduction to the system requirements by the product owners and the system development ended the 7th of December with a 24 hour integration test.

The team gathered after the system requirements were presented in order to create an initial prioritized product backlog. The team utilized Trello [1] in order to keep track of the tasks in the backlog. Trello is an online project management application and it was used since its design made it similar to post-it notes for management. It also has the advantage of being online and therefore accessible from both home and workplace through smart phones or computers. Figure 1.1 shows a screen capture of Trello board for a sprint log.

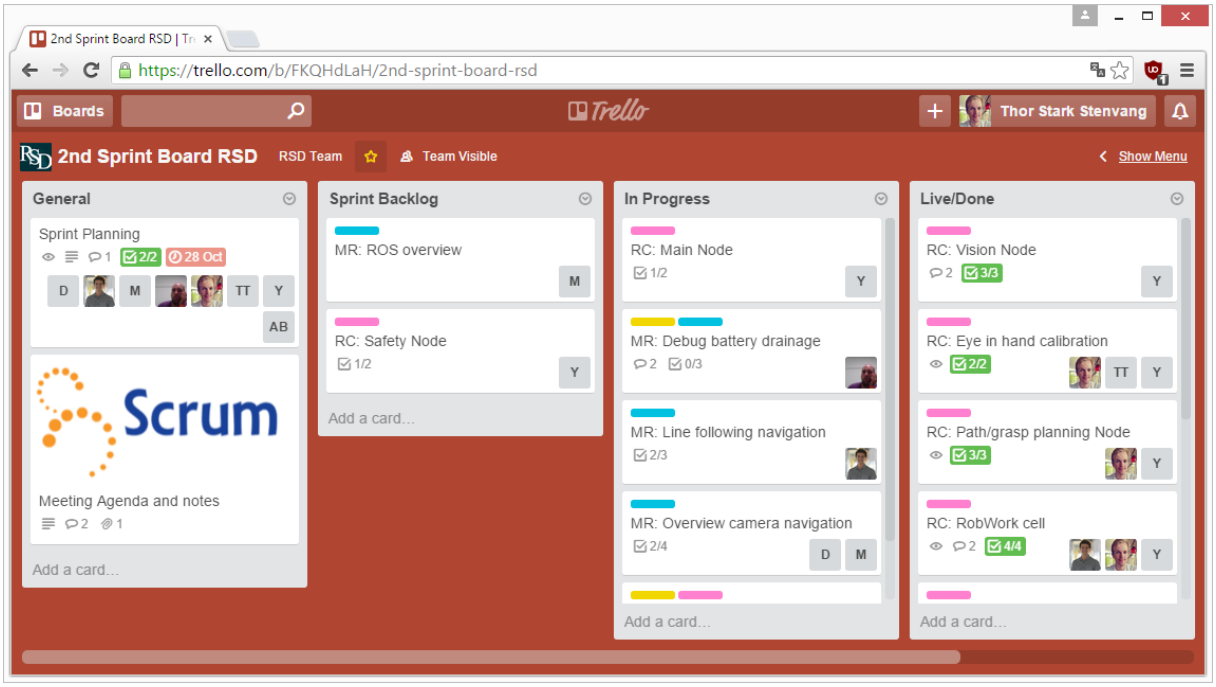


Figure 1.1: *Screen capture of a Trello showing the Trello board for the 2nd Sprint. The three rightmost lists are used to keep track of tasks and their state.*

1.1.1 Sprints

Over the course of the project the work was divided into 4 sprints. Each sprint planning, except the first, was done after the completion of the previous. For each sprint the goal and duration was decided. Prioritized tasks were then added to the *Sprint backlog*. While the *Sprint* was running each member could sign up for tasks and move them from the *"Sprint Backlog"* to *"In Progress"* or *"Live/-Done"*, see Figure 1.1. Listed below are the different Sprints and their description.

- **1st Sprint.**
  - **Goal:** Create a working robot demo for the TEK opening and prepare the simple elements for the robot cell.
  - **Duration:** 14th of Sep - 30th of Sep.
  - **Description:** Additional to the system described in the original system requirements, an additional system had to be developed with a release date of 30th of September. The new requirements were the development of a mobile robot able to perform at the TEK opening at University of Southern Denmark. The Sprint was designed with the new requirements and deadline in mind. There were several hardware issues with the robot cell at the beginning of the sprint, and with these two factors the sprint prioritized the development of the mobile platform.
- **2nd Sprint.**
  - **Goal:** Fulfil the requirements for the halfway demonstration.
  - **Duration:** 5th of Oct - 28th of Oct.

  - **Description:** A deadline for a halfway demonstration of the system was set to the 26th of October. This halfway demonstration required several subtask of the system to be complete and the requirements therefore spread the the workforce onto a broader part of the system compared to previous sprint since both mobile robot and robot cell had to be partly finished before demonstration.

- **3rd Sprint.**
  - **Goal:** Get the system ready so all subsystems and components can be integrated together in the next sprint.
  - **Duration:** 28th of Oct - 18th of Nov.
  - **Description:** With most of the work on the robot work cell done in the previous sprint. The workforce was shifted so more people were working on the mobile robot. This allowed the sub-parts to be ready for implementation and integration in the whole system.
- **4th Sprint.**
  - **Goal:** Integrate all subsystems and components and prepare for the 24 hour integration test.
  - **Duration:** 23rd of Nov - 7th of Dec.
  - **Description:** All sub navigation and skills of the mobile robot had to be implemented together so they in conjugation can perform all the defined task for the mobile robot. Additional the whole system had to be tested and finished including the communication with the MES. Additional bug fixes and recovery behaviour were done in order to complete the 24 hour integration test in an environment where other robots are present.

In order to keep an overview of the project and keeping it on track a weekly SCRUM meeting was held together with the Consultant and a Product owner.

1.2 Task Breakdown Table

Table 1.1 shows a break-down table. The table lists the project tasks broken into relevant subtasks. For each subtask it is listed how many percent of the task that was accomplished by each student.

Group Members: Dominik Weickgenannt, Jorge Rodriguez Marin, Martin Groth Albøge, Steffen Ivan Nielsen, Thor Stærk Stenvang, Timon Tomás, Yonas Alizadeh

Task	**Subtask**	**Done by**
Project Management	Trello	Thor (100%)
Mobile Robot	Hardware	Steffen (100%)
	Main Node	Jorge (100%)
	HMI	Timon (100%)
	Navigation Controller	Jorge (75%), Martin (25%)
	Relative Navigation	Jorge (75%), Dominik (25%)
	Line Navigation	Jorge (40%), Thor (20%), Martin (20%), Yonas (20%)
	Free Navigation	Dominik (50%), Martin (50%)
	Button Control	Yonas (100%)
	Camera Processing	Jorge (60%), Thor (40%)
	Obstacle Detector	Dominik (100%)
	Tip Controller	Steffen (70%), Yonas (30%)
Robot Cell	Hardware	Steffen (25%) , Thor (25%) , Timon (25%) , Yonas (25%)
	Main Node	Yonas (100%)
	Camera	Thor (100%)
	Vision System	Thor (30%), Yonas (70%)
	PLC	Yonas (100%)
	Kuka	Yonas (100%)
	Grasping	Yonas (100%)
	RobWork Scene	Jorge (50%), Thor (25%), Yonas (25%)
	HMI	Yonas (100%)
MES Server and Client	Implementation	Yonas (100%)
Financial Aspect	Calculations	Dominik (100%)

Table 1.1: *Task breakdown*

Chapter 2

Manufacturing Execution System

A Manufacturing Execution System (MES) Server breaks down an order into tasks, such as fetch, sort, pack, deliver LEGO bricks and schedules the tasks among the available resources in a system.

Figure 1 shows the connections of the MES Server. In this project the resources includes three robot cells, three mobile robots and one LEGO brick dispenser.

Due to a lack of a united MES Server for all teams, an individual MES Server was implemented. It acts more like a server than a MES server since it is only the robot cell and mobile robot belonging to the group that is connecting. The server has the role of acting as a communication hub between the two elements. The dispenser is not connected since no hardware is provided for the dispenser to get online.

From the server a user can manually request an order to be processed. The order will be sent to the respective devices and further communication between these is handled. When the server receives 'done' messages from both systems a new order can be made.

Chapter 3

Mobile Robot Design

The mobile platform performed the task of transporting the LEGO bricks between the dispenser unit, and the robot cell. It had to be able to fulfil this task in a completely autonomous manner under normal operation, avoid collision with obstacles, be harmless to its operators and be remotely controllable through a human machine interface. It was also a requirement that its progress should be monitored through the aforementioned user interface.

Due to the high complexity of the task, we decided to pursue a modular approach. In this chapter we will introduce the components comprising the system, the motivation for having them, their implementation and the reasons behind the choices we made during the design process.

3.1 Hardware

3.1.1 Provided hardware

The platform for the mobile robot revolves around the SDU FrobitPro, mounted with a tip loading platform. The whole system is driven by a micro-atx computer system with Ubuntu 14.04 and ROS Indigo. As power source is a $12V$ $7000mAh$ lead battery (same type as used for small mopeds) and a SDU Frobomind controller, that also act as a controller for the driving motors. To enable a continuous operation of the mobile robot, an intelligent CTEK $5.0A$ charger was used. This provided enough charging capability to test the system for an indefinite time while connected.
An overview of the robot can be seen in Figure 3.1. It was arbitrarily chosen that the front of the robot would be in the end with the small wheel.

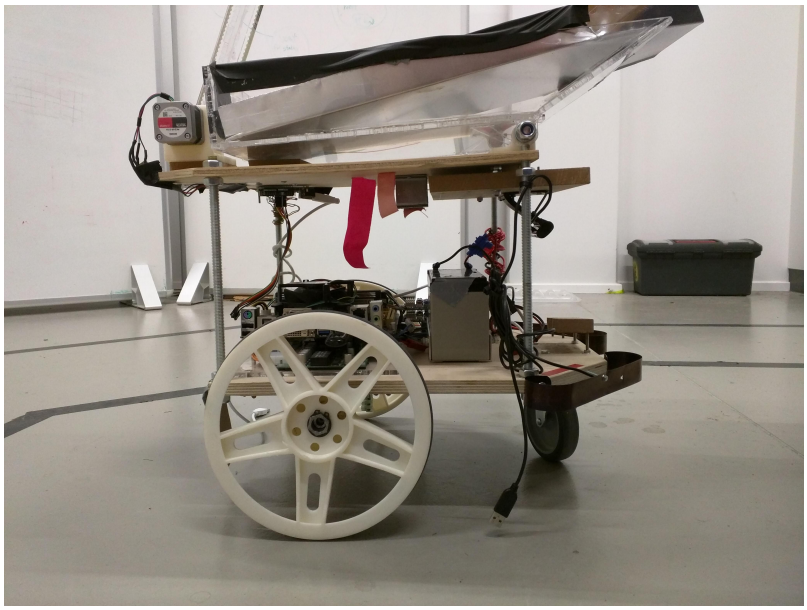


Figure 3.1: *FrobitPro Robot*

3.1.2 The main components for the tipper:

- PK-244-01A Bipolar Stepper motor
- Arduino UNO
- L298N dual H-bridge control board (Self-obtained)
- 2x Microswitch w/arm as end-point sensors for the motion

To obtain valuable information about the surrounding environment, following sensors were used:

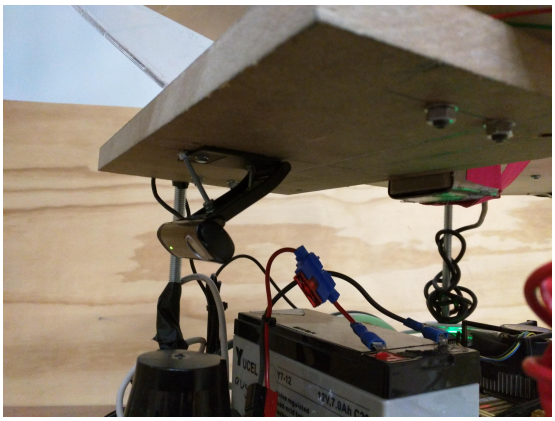
- Sick LIDAR laser scanner
- VectorNav IMU
- Logitech Webcam

The use of these sensors will be described in the appropriate sections.

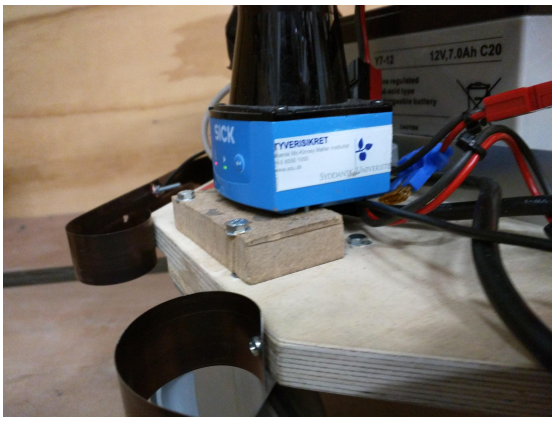
3.1.3 Modifications

Along the development and testing phase of the project it was discovered that custom hardware modifications was necessary for the sensors to be mounted properly to the robot, in regards to protection of the sensors and be able to make proper use of these.

It was decided pre-hand that it was not allowed to make adjustments (e.g. cutting and drilling) in the base plate of the robot, of which it was chosen that the line-following camera would be mounted below the tipper without obstructing the LIDAR's field of view. This can be seen in figure 3.2a.



(a) *Line-following/QR detection camera*



(b) *Lidar laser scanner*

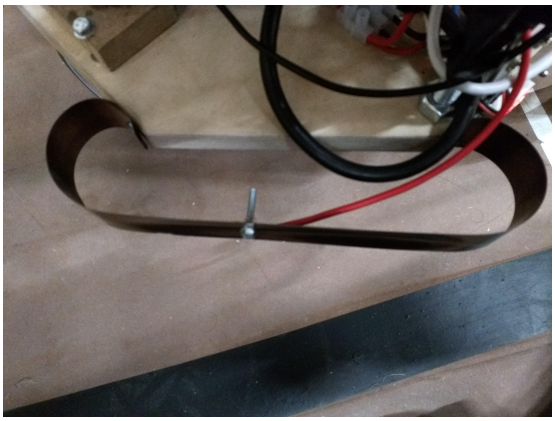
Figure 3.2: *Camera and LIDAR*

The previous use of an IMU on the robot had a remaining mount. With duct tape this was firmly attached. This can also be seen in figure 3.2a in the background.

In the preliminary tests, the LIDAR was put on a wooden board in front of the robot to get as much use of the 270 degrees sensor as possible. Experimentally and in regards to the price of this sensor, it was later chosen to move it within the robot base to avoid any mishaps during an unforeseen collision. See figure 3.2b. This obviously sacrifices some of the sensor angle.

During docking sessions in the dispenser/charging station, it was discovered that attaching a bumper-like system would increase the angle of which the robot could enter the charging station thus reducing the strictness of localization for going to the charger. The bumper idea can be seen in 3.3a.

Finally the Plexiglas tray on top of the robot was found inadequate to deliver all bricks at the conveyor belt, so an aluminium insert was created. To ensure that the bricks were delivered to the conveyor belt and not next to it, this insert was created with a narrow tip, as seen in 3.3b



(a) *Bumper on the Mobile robot*

(b) *Aluminium tray insert on the mobile robot*

Figure 3.3: *Hardware modifications*

3.2 Frobomind

The overarching architecture on the mobile platform is the FroboMind architecture. The architecture was originally designed for field robotics but suits the tasks of the mobile platform well. The resulting

system on the mobile platform partially conforms to the original FroboMind architecture. One of the common links here are the centralized command structure or decision making. This consist of the Main and Navigation Controller nodes. The Main node receives the mission and determines the next command to execute. This can be moving to a position, operating the tipper or communicating with the controlling MES server. If navigation is required the Navigation Controller itself performs the same type of operation, but only with respect to navigation. It has a list of skills available, and combines these to achieve its goal.
Another aspect of the FroboMind architecture used on the mobile platform is the handling of safety. The entire system on the mobile platform will be described in detail throughout the chapter.

The system is based on the Robot Operating System (ROS). This communication middleware handles communication between processes. The benefit of using ROS is the standardisation of communication between components. This standardisation makes it possible to easily replace and reuse components. As ROS has a substantial user community, a large amount of readily available components can be used with minimum modification. Some of the components from the ROS community used in this project are the localization (AMCL) and navigation in the open environment (move_ base). A more detailed description of these components are to be found in 3.5.3. These advanced components also causes some divergence from the original FroboMind architecture as they overreach several different aspects of the architecture.

3.2.1 Frobomind interface

The Frobomind interface is in charge of transforming the hardware sensors values into ROS messages that can be used by the system. This is implemented as a ROS package given inside of the Frobomind framework. A small modification in this package has been made so the status of the battery can be read. This allows the robot to create subroutines of charging when needed and continuing with the previous task when charged.

3.3 Flow control overview

As explained in the chapter 2, the MES server sends an order and the clients have to read it and act accordingly. In the case of the mobile robot, there is a node in charge of doing all the necessary steps in order to complete the order.

The flow control of the mobile robot is depicted in the figure 3.4. This is:

1. Wait for an order from the MES server.
2. Read the order and go the brick dispenser to get some bricks.
3. Go the the conveyor stated in the order.
4. Tell to the MES server to activate the conveyor and unload the bricks previously picked up.
5. Go to the position in which the robot of the workcell leave the order.
6. Wait for the robotic arm to finish the order.

7. Go to charge.

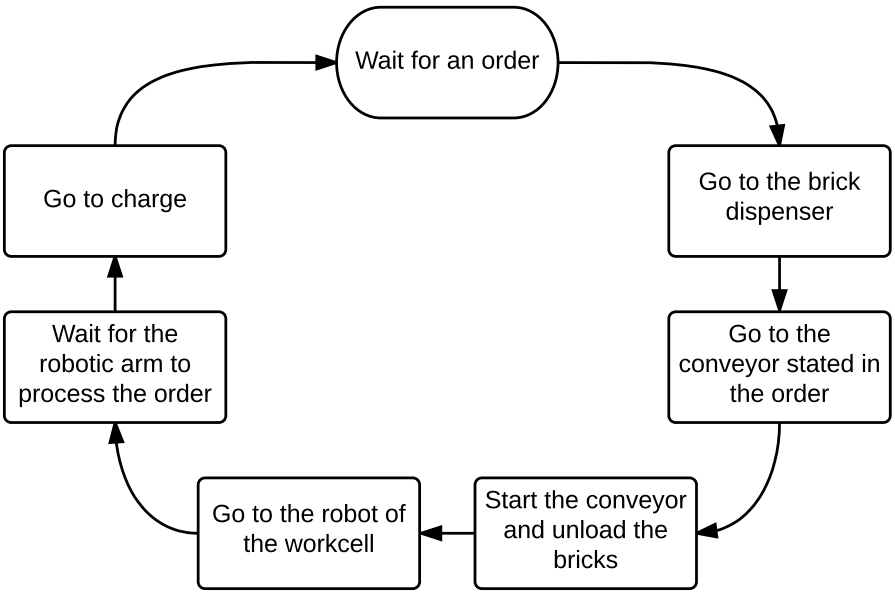


Figure 3.4: *Flow control of the main node in the mobile robot*

This process is handled by the *main node*, and is the highest abstraction level in the control of the robot. It is responsible of coordinating and actuating a second layer of abstraction which, at the same time, will be responsible of directly actuating the hardware. The communications between the *main node* and the second layer of abstraction, is shown in the figure 3.5.

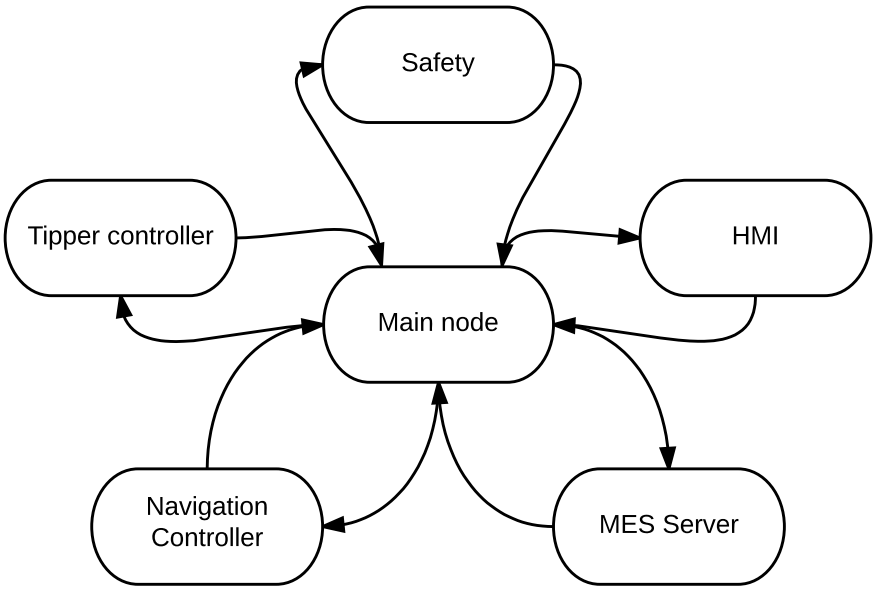


Figure 3.5: *Connections of the main node*

The main node has also an internal state of the robot represented in three states: *idle*, *auto* and *manual*. The mode *idle* means that the robot is not moving or executing any action. The mode *auto* means that is processing an order and, consequently, it can be moving. *Manual* stands for when the robot is being moved manually with the HMI. The states are intrinsically exclusive and the robot can

only be controlled manually if the robot is not processing an order. This avoids the possibility of the user to interfere with the order status, thus increasing the stability and reliability.

The connections are bidirectional and their functions are:

1. **Safety**: Reads the state of the different sensors in the robot that measure the possibility of harm the robot and, if in danger, sends a signal to stops all the actuators.
2. **HMI**: This can change the state of the robot to *auto* to start/continue an order, to *idle* to pause the current order or to *manual* if the user wants to control the robot manually. On the other hand, the main node can tell the HMI (1) its position and (2) the actions being carried out in that moment. This is that in the HMI is shown if the robot is, for example, inside the box and following the line or doing a relative movement.
3. **MES server**: The main node reads the order sent and starts the new hierarchy of actions. Also the main node can send messages to the MES in order to activate other agents controlled by this.
4. **Navigation controller**: As will be explained in the sections 3.6, it is responsible of, knowing the current position of the robot, calculate a path to the desired position and perform it. The main node can receive from it when the robot has reached the desired position.
5. **Tipper controller**: Handles the position of the tipper assembled in the robot. The controller can talk to the main node to tell when has finished the desired movement.

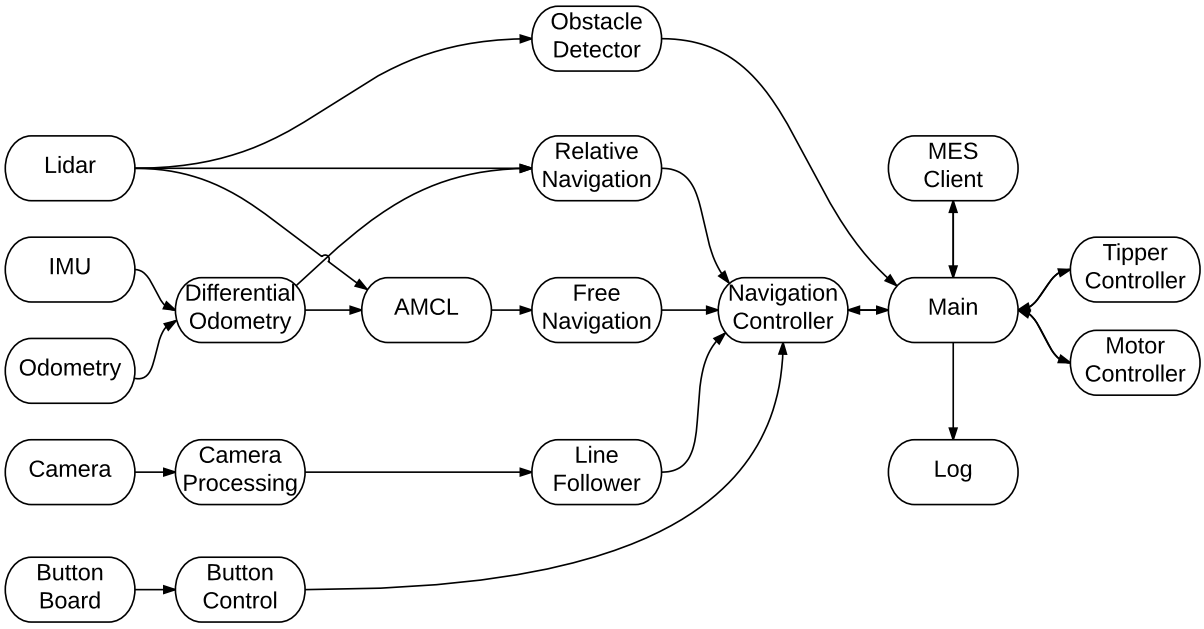


Figure 3.6: *ROS Node Structure*

A abbreviated ROS Node structure can be seen in figure 3.6. This is just a brief global view of the detailed nodes that are described in the following sub chapters to give the reader an overview.

3.4 Sensors and data processing

In the section 3.1 the different sensors of the robot were explained. In this section, the information gathered by these sensors and their processing is explained. Six different kind of sensors have been used in order to fulfil the requirements of the project. These are:

1. The camera, explained in *camera processing* (3.4.1) to follow the line and detect QR codes.
2. LIDAR (3.4.2) used to map the area, to localize the robot when in free navigation (see section 3.5.3), to detect obstacles in front of the robot and as a skill (see section 3.6.2) to allow the robot to approach a static object.
3. Encoders in the motors, used for the combined odometry implemented in frobomind.
4. IMU, used in combination with the wheels odometry to know the position of the robot.
5. Board of buttons, that equipped with two buttons and one RDG led, allows the user the start or stops the robot at the same time that indicates what is is doing with the led.

In the following subsections the data processing made with each of the sensors is explained.

3.4.1 Camera processing

In order to move around the robot workcells, a black line has been installed in the floor so it can be tracked. This line communicates the workcells themselves along with the internal parts of them. These are: the entrance of the workcell, the conveyor and the position where the robot must place the processed order. To distinguish the areas QR codes have been placed in the floor following the convention that all the groups have agreed.

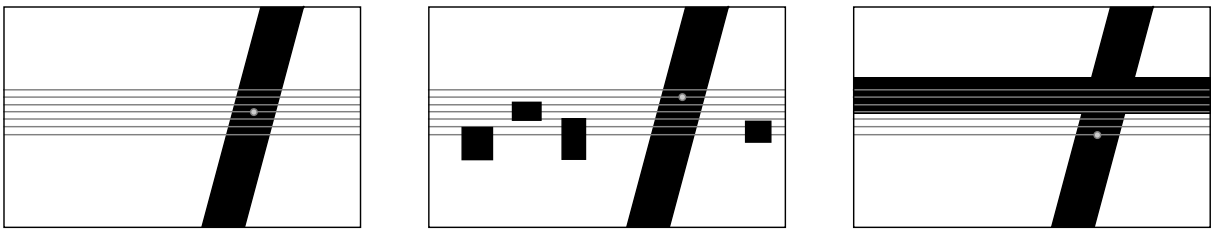
Then, the camera sensor is used with two purposes:

- Detect the line
- Detect and read QR codes

Line detection

Speed and robustness have been the priority when designing the line detection. The filter works at 30 FPS and with a extremely low failure rate. However, an inertia in the filter has also been added, so the last detected point will be returned during a certain time in case the line has been stopped being detected.

Due to the line follower adjusting the PID given a reference point, the full image doesn't need to be interpreted. Because the reference point is going to be the center of the image (read section 3.5.2), the image is cropped with a vertical offset about the reference point. The width of the image is not cropped so the horizontal deviation can be measured in the largest range possible. After this crop, the image is binarized with an RGB threshold which give a robust detection of the black line. The line



(a) *Clear detection of the line* **(b)** *Detection when noise in image* **(c)** *Detection when crossing line*

Figure 3.7: *Representation of possible cases when detecting the line*

is then searched with a virtual horizontal line over the reference point. This line will cross with the black line giving two intersection points that, averaged, give the center of the line.

In order to increase the robustness 10 equally spaced vertical lines are used. The process is the same for them and the final point is the averaged one from the intersection points that are closer to themselves. This is done due to some noise can appear in the image (see figure 3.7b) or a crossing line can be given (see figure 3.7c). The 10 parallel lines reduces the failure rate and makes the filter very robust and fast. A clear representation is shown in the figure 3.7a. The implementation has been carried out using the OpenCV libraries [2].

QR detection and read

Due to the QR detection being done while moving, the image suffers from blur. This could be reduced by a Wiener motion noise filter but this is computationally expensive, so another approach has been taken. Instead, with inspiration from other class mates a fast shape detection is used. The shape detector looks for a white contour in the image which encapsulates a number of black elements. If a QR shape is detected, the speed is reduced until the blur no longer affects the QR reading.

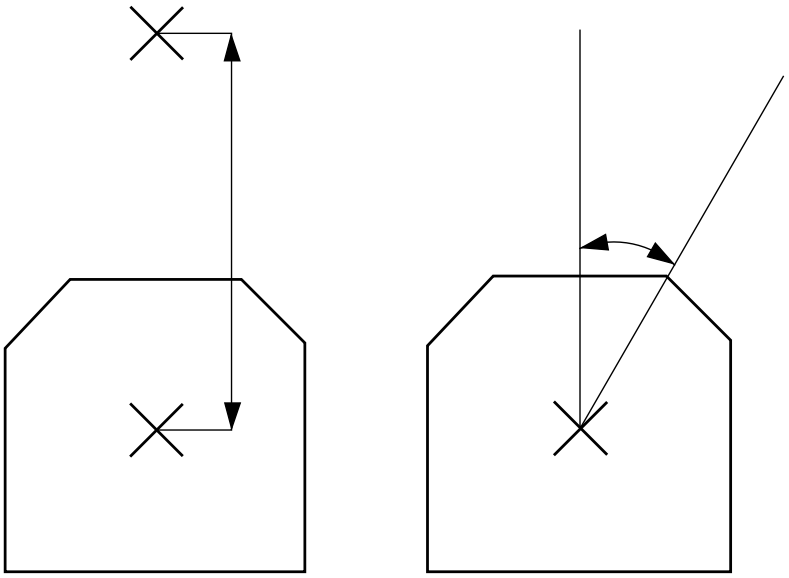
The QR detection is made using the Zlib libraries [3]. This gives the ability of easily reading the information contained in a QR code.

3.4.2 LIDAR processing

The provided LIDAR (SICK TiM310) is capable of measuring the distance in 270 degrees, however due to the setup on the robot, only about 200 degrees are used. The SICK TiM Ros Package [4] allows for direct set-up of minimum and maximum angles and publishes the measured points as a LaserScan message. This message then is directly used by the various skills.

3.4.3 Combined frobit odometry

The Frobit odometry module publishes the odometry status of the robot based on inputs from the wheel encoders and the on-board IMU. Sensor fusion is used to get the orientation of the robot using the IMU and this data is compared to the encoder feedback from the wheels resulting in robust odometry data.



(a) *Linear movement representation*

(b) *Angular movement representation*

Figure 3.8: *Representation of the relative movements*

3.4.4 Button-board

A Tiva TM4C123 Microcontroller board is used for direct hardware control and feedback. Two buttons and a RGB led provides the hardware interface. A serial UART line connects the board to the PC.

3.5 Navigation

In this section the different types of navigation that let the robot move are explained. These are the actions of the robot that let the robot change its position and they are used as means by the navigation controller (3.6) to move around the workplace. As will be explained in the section 3.8, this states are the only ones that activate the deadman signal that let the robot move. These states are intrinsically exclusive, meaning that, for example, line navigation and free navigation can't be used at the same time.

3.5.1 Relative navigation

Making use of the combined odometry from the wheels and the IMU, the relative navigation has been implemented. Two different movements can be carried out: Angular and linear. This two movements are represented in the figures 3.8a and 3.8b respectively.

The relative movement is based on a P-controller that tries to minimize the error between the current position and the desired position, which is calculated from the current position and the required movement.

3.5.2 Line navigation

Taking the information from the camera processing a PID is used to track the line. This PID tries to minimize the distance between the point that represents the center of the line and a desired point. The desired point is the center of the image due to the camera was physically positioned in the perpendicular to the rotation axis of the robot.

3.5.3 Free navigation

When the robot reaches the end of the line it needs to navigate to and in the box containing the charging station and brick dispenser. For localisation in this area the LIDAR, odometry and a camera localization system is available. The camera system is used to initialize the particle filter used for navigation. The system requires a marker placed on the robot visible from the roof mounted camera. This systems is currently not used on the robot. The system was discarded due to the performance of the LIDAR localisation in the area the camera system covers. The removal of the marker from the top of the mobile platform increased the area for catching dispensed bricks thus reducing the required precision for navigating to the brick dispenser. For these reasons the camera system is disabled.

As per project description, the navigation inside the box is to be done based on SLAM. Due to complexity of the entire project and potential diversity of SLAM implementations, an open-source implementation of SLAM was chosen. More specifically the gmapping-package [5] which has a livid community of supporters and builds a ROS wrapper for “OpenSlam’s Gmapping”[6].

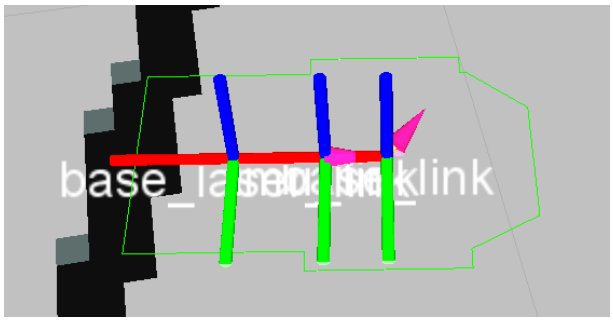


Figure 3.9: *TF Frames*

TF-Frames of LIDAR, IMU and the robot base were set up, see Figure 3.9 , this allows for easy transformation between LIDAR and odometry data, inside the package. Using these, the SLAM-Implementation uses a particle filter to sequentially build up a map of the area. By manually driving through the entire workspace, a map was created.

Figure 3.10: *Slam Map*

Using the map-server package from the ROS 2D-Navigation Stack [7] the generated map was stored, see Figure 3.10 and is then broadcasted as a static map for localization.

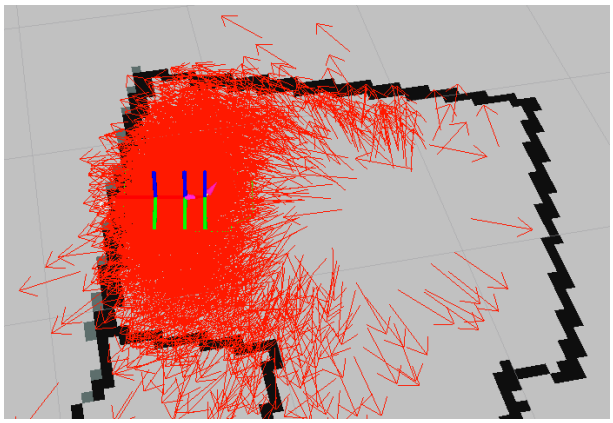


Figure 3.11: *Particle Cloud*

The AMCL (Adaptive Monte-Carlo Localization) package[8] (which is also included in the navigation stack) then uses the static map, the LIDAR as well as odometry data and a particle cloud (as seen in Figure 3.11 to localize the robot while in free navigation. After tuning the parameters, AMCL worked fairly reliably for navigation both inside and outside the box.
The path planning and execution used for navigating in and outside of the box is done with the ROS package move_base[9]. The package uses two path planners, a global and a local, each of these planners maintains a cost-map. The cost-map for the global planner contains the map of the area and in this cost-map a path from the current pose to the goal pose is calculated. When the global planner succeeds in creating a path the local planner takes over. This planner generates multiple plans with the parameters linear and angular speed. These plans are evaluated based on the reduced distance to the goal, the distance to the path and distance to obstacles.
In situations where the local planner is unable to complete the navigation a number of recovery be-

haviours is executed. Initially obstacles further away that 3 meters are cleared. If this does not prove successful the robot performs an in place rotation to clear local obstacles. This also helps the AMCL in adjusting the position of the robot which has proved particularly useful when navigating inside box. Is the navigation still failing, all obstacles outside the area of the rotation are cleared.
Due to the box being fairly monotone in respect to the LIDAR readings (clean, walls, no landmarks), precise navigation inside the box was sometimes not perfect. Also with multiple robots present in the box, issues, especially when trying to charge occurred. To ensure reliability of charging a line was added, using the already implemented Line Following behaviour. The move_base is used to navigate the robot to the start of the charging line. Then the robot follows the line until a pre-specified distance from the wall using LIDAR. If the battery level is increasing, the behaviour is successful, otherwise it will try again. This proved to be reliable in successful charging.

3.5.4 Manual navigation

The ability to control the robot manually from the HMI is also implemented. This controller (see Figure 3.12) shown in the *Advanced mode* of the HMI, lets the user to control the linear and angular motions as well as the speed of these. This behaviour is only allowed when all the skills and routines of the robot have finished so the manual mode cannot overwrite the automatic plans of the robot.

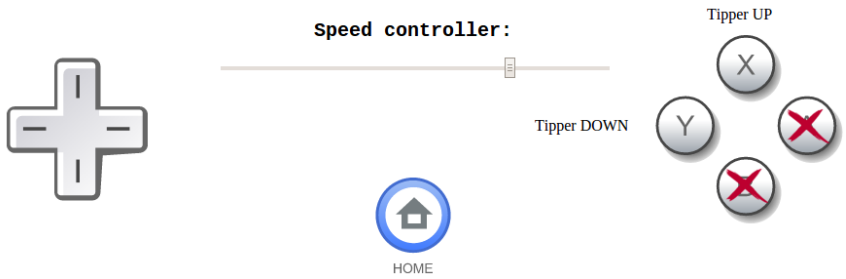


Figure 3.12: *Manual navigation controller in the HMI.*

3.6 Navigation Controller

Using the processed information from the sensors and the different kind of navigations, the robot is able to move and satisfy all the requirements from the project. In this section the *navigation controller* is explained. This is in charge of finding the path, and actions associated to said path, given two nodes.

3.6.1 Navigation state representation

In order to navigate from one position to the desired one, a graph of the possible states in which the robot is stopped or idle has been represented. This graph depicts all the possible states of the mobile robot in which the robot has not movement along with their associated transference functions. These are other states in which the robot is moving or changing its state and that in this project will be referred as *skills* (Section 3.6.2). In the figure 3.13 the graph of the project is shown.

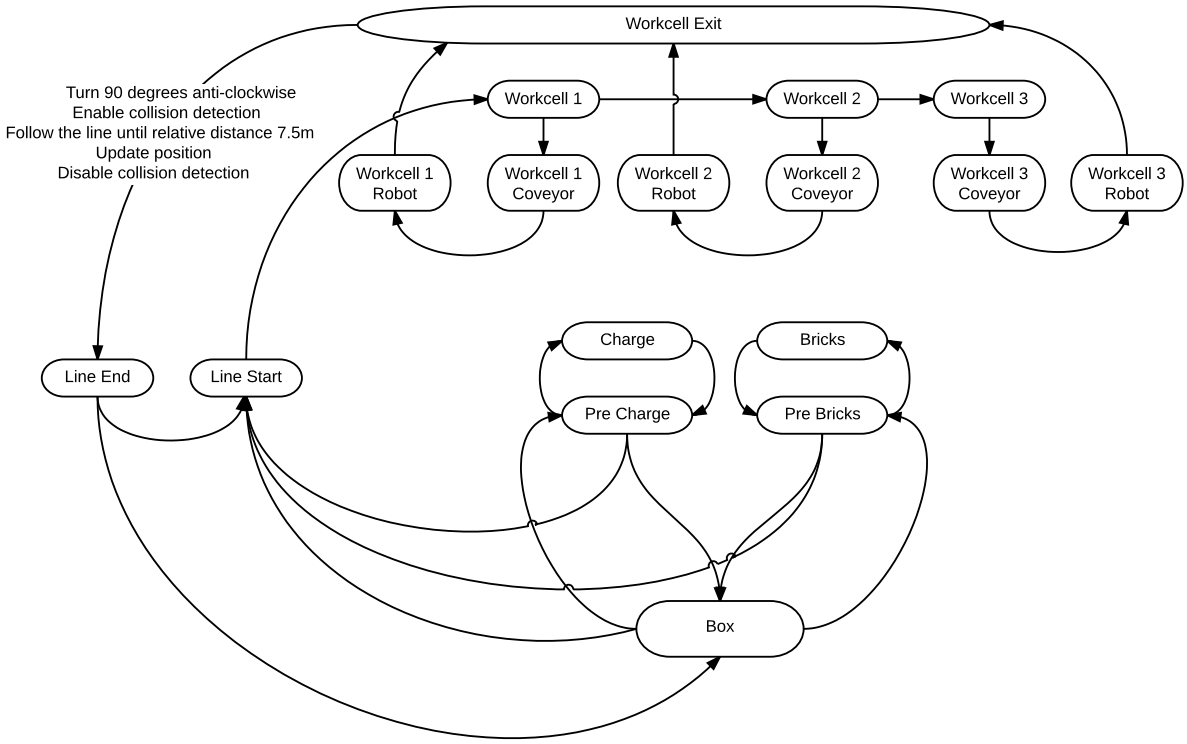


Figure 3.13: *Graph representation adopted in the project*

As an example, in the transference between the node *"Workcell Exit"* and *"Line End"*, the associated conditions are shown. These are: first, needs to *turn 90 degrees clockwise*, then *follow the line until the QR "line_end"* is detected, all this done with the collision detector turned on and changing the internal position of the robot in the end.

3.6.2 Skills

As stated before, the skills are available actions to transfer the robot from one navigation state to another. They let the robot change its state and position. The implementation is based on a couple between one of the navigation types and some information from the sensors used as a condition. The skills designed and implemented are:

- Follow the line until desired QR
- Follow the line until desired LIDAR distance
- Follow the line until desired relative distance
- Linear relative move
- Angular relative move
- Go to free position
- Detect obstacles
- Wait

- Specialized charging skill
- Initialising the AMCL

An example of the skills associated with the transition from one node to another is shown in figure 3.13. The skill list shown only facilitate the transfer when exiting from work-cell 1, due to the use of the follow line until relative distance. This distance will be different for each work-cell.

The navigation skills have previously been described. The specialized charging skill combines multiple of the other skills. Initially the robot moves to a position in free space. Then it follows a line until a certain LIDAR distance. Here it waits 10 seconds to see if the battery level has risen. If it has risen the state is set to charge. Otherwise the robot reverses and restarts the process.
The last skill, initialising the AMCL, is used when the robot transitions from the line navigation to free navigation. This is necessary because the AMCL looses its position when the robot moved into the robot cells.

3.6.3 Graph search

In order to find the most optimal path between two nodes in the graph a graph search algorithm has been implemented. All the transfers between states have an associated cost which is used by a Breadth First Search (BFS).

The BFS is an algorithm of searching in graphs that explores all the neighbours of the root, then the neighbours of this ones etc. BFS is a complete search algorithm and optimal meaning that will always find the optimal solution. In this case as the cost between nodes is a given distance, the BFS will always find the shortest path.

The negative part of the BFS is the space complexity but due to the graph being rather small, the search time is in the order of milliseconds.

3.7 Tip controller

The purpose of tip controller is to create a simple and effective way of delivering a large amount of LEGO bricks to the conveyor belt autonomously. This is done by controlling the mounted belt drive with a stepper motor, an H-bridge and a Arduino micro-controller. To control the limits of tipping mechanism, two micro-switches were added (tip down/tip up).

3.7.1 Stepper controller

The $PK-244-01A$ stepper motor is bipolar. This means that it requires (at least) four wires to control the two coils in contrast to a unipolar motor which has one supply wire and usually only requires two control wires. The four wires are driven by the $L298N$ dual H-bridge, powered by $12V$ from on-board pc power supply. Four control pins are connected to the Arduino and signals on these controls the motor wires. Consulting datasheet for the motor, a minimum of $4V$, $1.2A$ is required to drive the motor. Timing is calculated as a minimum of $3ms$ delay between steps in the timing diagram. To get an initial

high torque from the motor, a half-stepping sequence was chosen and a ramp function was created to slowly start the motor and exponentially increase speed when the tip is in full motion. Due to friction in the system and less torque the full-step sequence was disregarded. The Arduino software has a built-in library for stepper motors, but this was found difficult to manage in regards to delay of the steps.

3.7.2 Arduino and serial communication

Setting a general serial communication to the Arduino is straight-forward. The speed is chosen with regards to the ROS node and a serial connection is initialized with the *Serial.begin(speed)* command. The Arduino is set up to receive a character "u" for tipping (up), and "d" for levelling (down). When an action has successfully executed, a "d" (done) is returned to the ROS node.

3.8 Safety system

The safety in the mobile robot is intrinsic in all the nodes that allow its movement. Additionally the Safety Eyes was supposed to be used to detect humans in the area close to the stairs. However due it not working, it was not being used. There then are two aspects to talk about: (1) the incident handler based on the the given Frobomind implementation. Based on certain inputs, an *activation* signal is created, allowing the robot to move. (2) On the other hand, the obstacle detector is used to feed another skill (see section 3.6.2) that changes the speed, or even stops, the robot depending on the distance of obstacles detected by the LIDAR. This can be activated or disabled depending on the situation.

3.8.1 Incident handler

The incident handler is based in the *simple incident handler* given inside of the Frobomind framework. This system receives two signals: (1) the *deadman* and (2) the *critical fault*. When both signals are received correctly, the incident handler outputs another signal that *enables the actuation*. This signal is directly received from the system that actually performs the actions that move the motors in the robot. Both signals are booleans with a time stamp and need two conditions to be approved:

1. As booleans, need to be true.
2. As stamped, need to be received in an interval smaller than a threshold. In our system the signal need to be received in intervals inferior to $50ms$.

If both, *deadman* and *critical fault*, are received by the incident handler satisfying those conditions, the output signal *actuation enable* will be published and the robot will move.

Deadman

The deadman signal is used by the navigators to actually perform a movement. As stated previously, the signal needs to be sent in intervals lower to 50 milliseconds and be true. The deadman signal

received in the incident handler does not contain information about the sender, but it is not a problem due to the navigators being mutually exclusive. This means that only one deadman signal is received from one navigator.

Critical fault

The critical fault signal is controlled by the main controller of the robot. This node, as explained in the section 3.3, controls among others the mode of the robot. Only in the modes *manual* and *auto* this signal is activated, meaning that if the robot is in *idle*, either because there was a critical fault, or it was manually stopped, the robot cannot move. As stated previously, the signal needs to be sent in intervals lower than $50ms$ and be true.

3.8.2 Obstacle detector

Its function consist of reading all the distances information from the LIDAR and, given a detection angle and two thresholds (*proximity alert* and *colliding*), output a signal which will slow down or stop the robot. The obstacle detector is used inside of an skill (see section 3.6.2) so it can be activated or disabled when necessary.

In practice the obstacle detector is only used in places where (1) there is a risk to crash and (2) its behaviour does not affect to the main one. This means that, for example, inside of the robotic arm work-cell it is disabled due to (1) the robot cannot crash with any moving agent and (2) due to the proximity to other static parts inside the work-cell, it would be in the *proximity alert* mode all the time.

The output of the obstacle detector can be three different exclusive states: (1) *safe*, (2) *proximity alert* and (3) *colliding*. The default mode is *safe*, while the other two modes are enabled based on two thresholds. The thresholds are expressed in the same units as the distance received from the LIDAR being in our project $40cm$ for *proximity alert* and $15cm$ for *colliding*. When in *proximity alert* the robot will slow down its movement (any) in a factor of 0.5 and when in *colliding* it will stop.

The limitations of this system is that with the LIDAR being mounted in the front, only obstacles which are in the area of the LIDAR can be detected. So no obstacles in the sides and behind are detected and these movements are "blind".

3.9 Human Machine Interface

A Human Machine Interface (HMI) is an indispensable part of all complex robotic and automation systems. It not only makes it possible for the operators to gain manual control over specific subsystems or the whole system, but can provide feedback on its state and performance, and serve as a mean of intervention in emergency situations.

To reduce the risk of having communication failures rendering user interaction impossible, we have opted to pursue a redundant HMI solution, consisting of both a virtual and a physical User Interface introduced in the following sections.

3.9.1 Virtual UI

The virtual system is implemented following the Model-View-Controller architecture, where clients are connected to a central server unit.

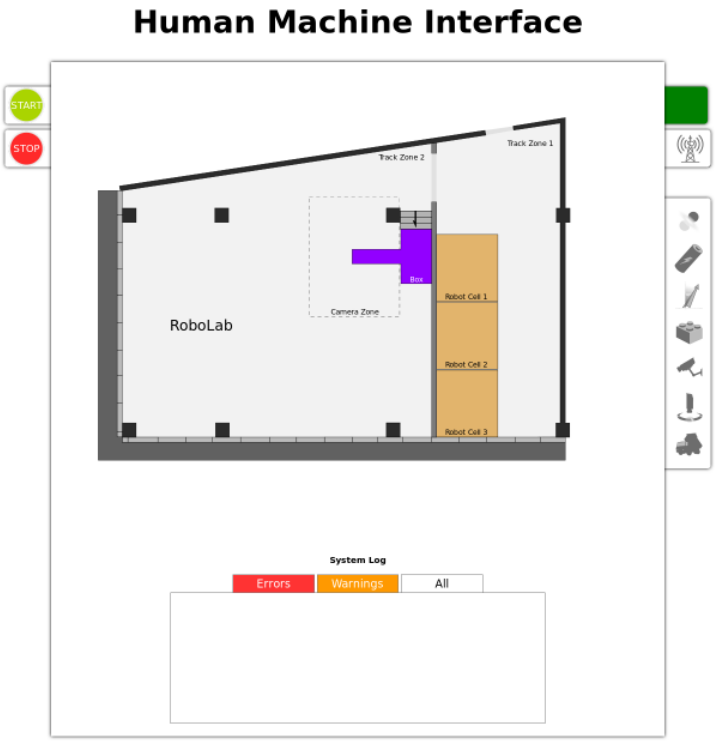


Figure 3.14: *Regular user interface for the mobile platform*

On the client side, a web application - written in HTML[10] and formatted by CSS[11] - is rendered through a web browser serving as the view. JavaScript[12] modules are run in the background handling user interaction and the updating of the interface. The model used for communication is well defined through a protocol. The UI is divided into two parts: a regular - see Figure 3.14 - and an advanced interface - see Figure 3.15. This decision was motivated by the fact, that the information displayed on the UI, as well as the on, off and the emergency stop buttons can be useful even for those users who lack the inner understanding of the system. By separating the other functionality, such as the manual controls and the textual interfaces from the rest, we prevented any unintentional harm caused by the unskilled utilization of the HMI.

On the server side, a proxy - written in Python[13] - conveys the messages published by the ROS[14] components to the client, and vice versa. The client-server communication is based on WebSocket[15] technology - we are using the Autobahn|Python[16] - Twisted[17] implementation -, which enables full duplex communication between the parties.

The system is completely platform independent, requires no installation at the user end, and only a minimal set-up on the server side. This aspect of an industrial application is most desirable, since it leads to faster deployment times and minimized costs arising from dependency issues.

Figure 3.15: *Advanced user interface for the mobile platform*

The virtual system has the following capabilities:

- Displaying of messages broadcast by various subsystems, with different views for the separate message types, namely: Errors, Warnings and all
- Possibility to change the operation mode of the mobile platform through user interaction
- Broadcast of an emergency signal
- Robot location and activity information display
- Provision of manual control of robot movement - with speed adjustment - and tipper states
- Provision of textual command interfaces
- Separation of normal and advanced features
- Automatic error handling in relation to connection malfunctioning

3.9.2 Physical UI

The user can control the mode of the robot with two buttons integrated in a button-board (see section 3.4.4). This board integrates two buttons: One for changing the mode of the robot to *auto* and the other to *idle*. These two modes are explained in section 3.3.

At the same time, the board integrates a RGB led that change its color and the blink frequency to express different modes and actions. The different indications of the LED are shown in the appendix A.1.

Chapter 4

Robot Cell Design

Equipped with a robot manipulator, a gripper and a conveyor belt the task of the robot cell is to sort the LEGO bricks delivered by the mobile robot. These are dropped onto the conveyor belt which moves them closer to the camera and robot. Here the corresponding bricks ordered by the MES system should be picked and returned to the mobile robot. This is the task each robot cell has to fulfil, and just like in the case of the mobile platform, the process can be subdivided into smaller tasks, which we will introduce in the upcoming sections.

4.1 Hardware

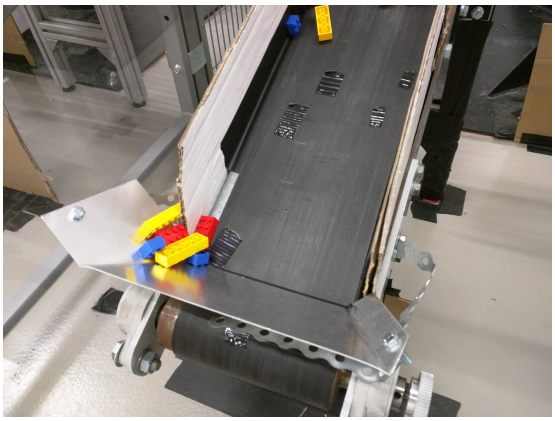
The following hardware elements was provided in each robot cell:

- Kuka KR6 Robot
- PG70 Gripper
- Conveyor Belt with 3 phase 240V Motor
- ABB (ACS150-01E) Frequency Converter
- ABB (PM554) Programmable Logic Controller(PLC) for conveyor control
- SICK (M4000) Laser safety fence
- HD Webcam for LEGO brick detection in 2D
- Kuka touch screen for HMI

The set-up for involved components will be discussed in details where appropriate.

4.1.1 Modifications

The conveyor belt required some changes to be able to handle and control the flow of LEGO bricks. First a funnel, see figure 4.1a, was added as a tip-off location for the mobile robot.



(a) *Funnel at delivery point for the mobile robot*

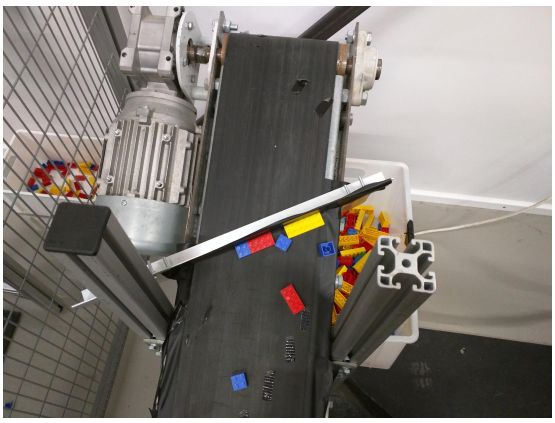
(b) *Brush system to de-cluster the bricks*

Figure 4.1: *Example of conveyor modifications*

Secondly, to avoid clustering and multiple layers of bricks a passive system consisting of a brush and cardboard was invented as shown in figure 4.1b.

To provide the necessary friction to go through the system, duct tape was added at random places to the conveyor belt. See figure 4.2a.

(a) *Duct tape mounted to conveyor belt*



(b) *Slider to ensure spare bricks to fall into the box*

Figure 4.2: *Example of conveyor modifications*

Finally, to avoid unwanted bricks on the floor, a passive slider was mounted near end of the conveyor belt. This allows the bricks to fall into a well placed box and collected at a later time. See figure 4.2b

4.2 Camera mounting and calibration

The provided camera is a *Logitech webcam C930e*. The camera was mounted on the robot end effector. This was done from the hypothesis that a eye-in-hand camera/robot relationship was easier

to calibrate compared to a hand-to-eye system and since there is no concerns with colliding with the camera the view can be closer to the grasping area.

Camera parameters was controlled using the video4Linux ver. 2 API and driver[18]. A resolution of 1280x720 with a 30 FPS was chosen. The resolution was kept high in order to have a rich details in the image and therefore making detecting bricks more stable with a cost of increased computation time due to large image sizes. The auto focus of the camera was disabled and the focus was tuned manually since the camera will be fixed in place relative to the bricks. Brightness, contrast and exposure was also manually tuned in order to capture images with minimum variance. This makes it easier to apply vision algorithms.

Camera calibration was done using the camera_calibration ROS package[19]. Using this tool the intrinsic parameters of the camera was calculated from images of a 9x7 chequerboard held in front of the camera in 73 different positions. Rectification of the image was then performed using the image_proc ROS package[20].

4.3 Work-cell modelling and calibration

Due to new robots and new work-cells a complete new modelling of the work-cell was needed. Since RobWork and RobWorkStudio was used, the format was given by these frameworks.

The modelling of the Kuka Kr6 robot was done by converting an existing 3D model and creating the corresponding XML files. The conveyor belt was measured and designed in a CAD program. The gripper model and the rest of the cell elements was copied from another similar work-cell used in earlier projects.

The calibration of the scene elements was done in two steps. A rough calibration was done based on physical measurements. The more precise calibration consisted of jogging the robot to different points in the scene and measuring the difference in the scene compared to the difference in the real world, minimizing this error during a few iterations.

Figure 4.3 shows a section from the scene. The TCP frame of the gripper is shown alongside the frame in which the collected Lego bricks is delivered at and the defined center frame of the conveyor belt. All Lego bricks are defined in this frame. The frame of the camera mounted the gripper is shown as well.

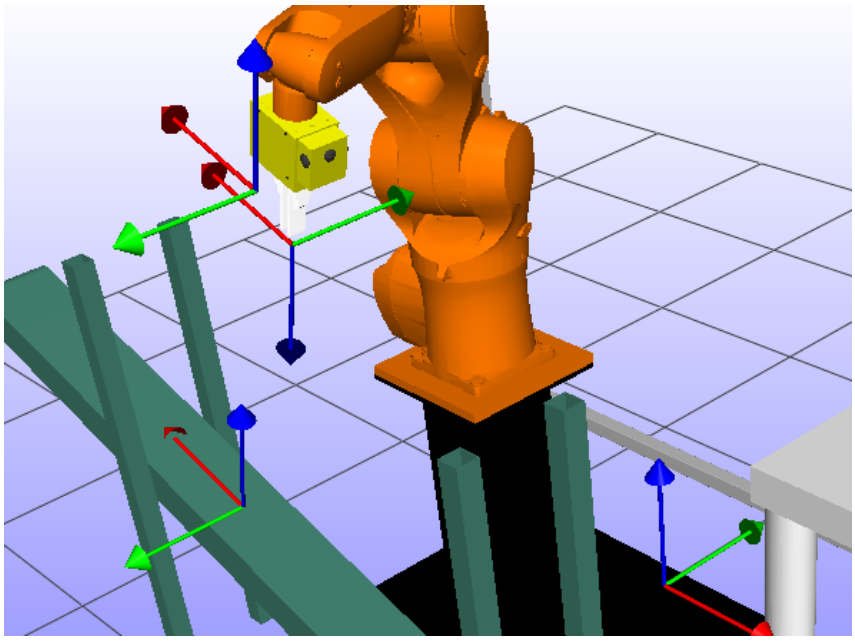


Figure 4.3: *Work-cell: Gripper, camera, conveyor-belt and delivery frame*

Lastly the calibration between the gripper, conveyor and camera was achieved using a simple yet effective method. The center of the conveyor frame was marked on the real conveyor by aligning the z-axis of the gripper and conveyor frame, making the gripper fingers point at this center point. Next the center of the camera and conveyor frame was aligned and the marked point should now be at the center of the image as well. By looking at the offset and correcting the required amount, this method proved well.

4.4 Flow control overview

The overall system flow is explained in figure 1. This section deals with the system encapsulated as the robot cell. The overall states of the robot cell is as shown in figure 4.4. This describes the four waiting states and the actions performed when switching state.

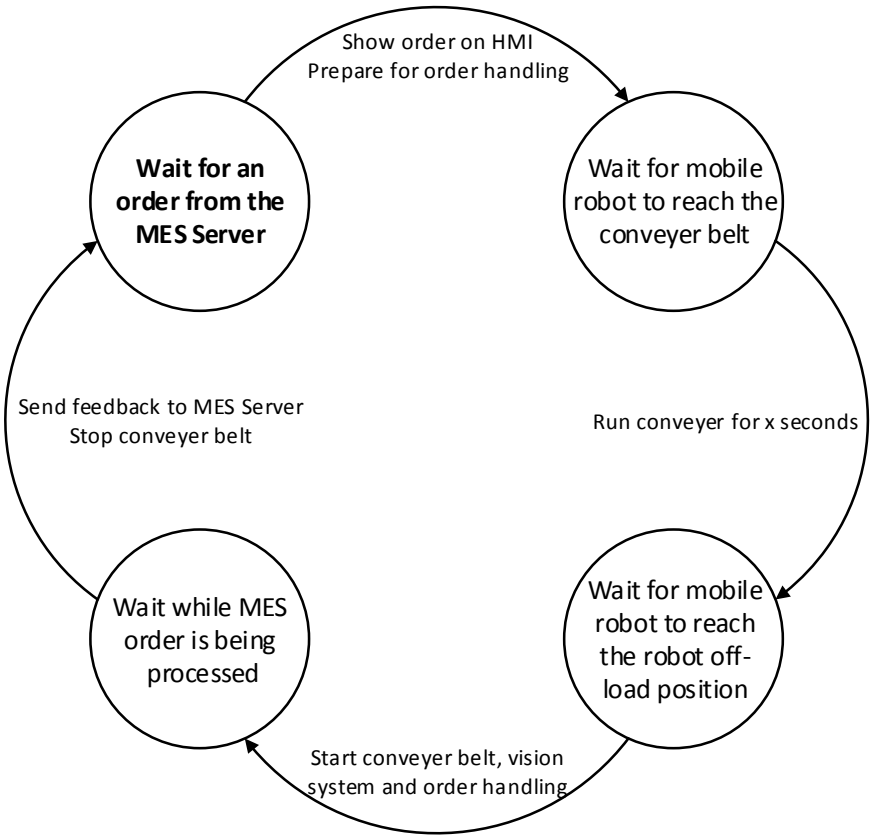


Figure 4.4: *Overall state diagram of the robot cell*

Figure 4.5 shows the elements that the robot cell consists of and the connection between them. The center is the main node which acts as the application and main point of the brick sorting program.

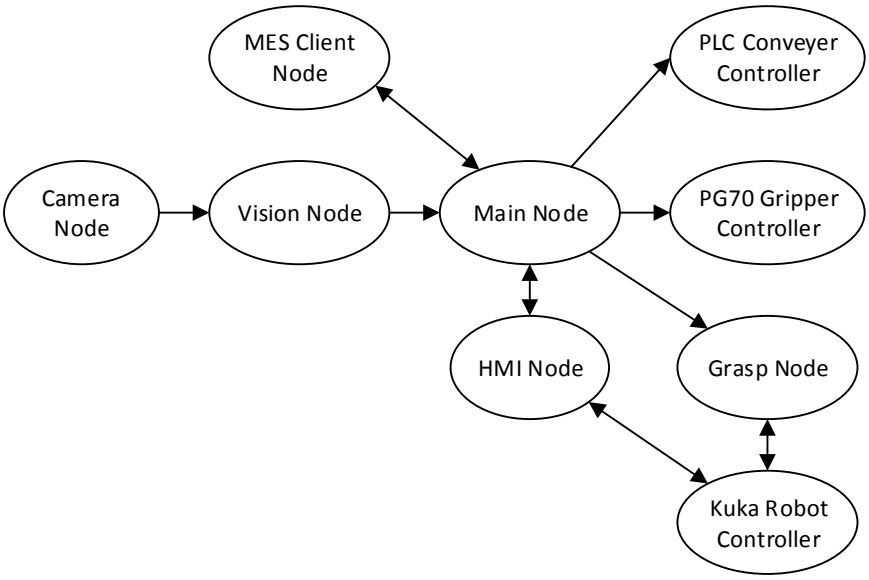


Figure 4.5: *ROS Node structure*

The next sections cover the details of some of these nodes.

4.5 Kuka Robot Controller

A Kuka ROS node is provided for controlling the robot arm in the robot cell. This makes it possible to communicate and exchange information such as if it is currently moving and what the configuration is. The robot can be moved in configuration space with a given speed and acceleration.

ROS services is provided for the above functions and acts as the interface between the Kuka actions and the main application.

Some issues were experienced with the Kuka setup when RSI packages was lost due to excess traffic in the switch used to connect the devices. When this occurred the robot controller stopped and the robot could not be moved before a manual reconnection. This situation could be handled by switching the state of the system to *idle*, reconnecting the robot and switching back to *auto*-mode. The system would then continue from the stopping point.

4.6 Conveyor Belt Controller

The conveyor belt is driven by an AC motor and requires a frequency generator in order to move. The provided frequency generator can either be manually controlled or by a $24V$ digital interface. The interface between the controlling computer and the frequency generator is a PLC. A PLC is an industrial *computer* with $24V$ IO interface and is logic driven. This can be programmed in different

ways such as Ladder Diagrams, Structured Text and Function Block Diagrams.

The PLC in this project is programmed in Structured Text and is simply set up to respond to some serial commands. This makes it possible to start, stop and change the direction of the conveyor belt from an external serial connection.

4.7 Safety

The safety in the robot cell consists of a laser beam from Sick mounted in the entrance of the cell. The safety signal is directly connected the Kuka Controller unit and the robot is thus reacting immediately without requiring user interacting. This leaves the conveyor belt, gripper and the Lego sorting system aside.

The safety signal can be fetched through the Kuka RSI interface which is provided in ROS format. This allows the system to detect when the safety signal is changing and react upon it. This reaction consists of stopping all actions hereby all physical movements and switching state from *running* to *idle*. The order won't be cleared as such, making it possible to continue when the cell is safe again.

4.8 Vision

The vision node has the role of detecting LEGO bricks on the conveyor belt using the camera. The detection of brick includes the color, size, position and orientation. This is information that is needed in order to determine whether the brick is needed or not and how to grasp it.

The vision LEGO detection algorithm is custom made and consists of several elements from the world of vision. Figure 4.6 shows the steps performed on an image in order to search for bricks. These steps are described next.

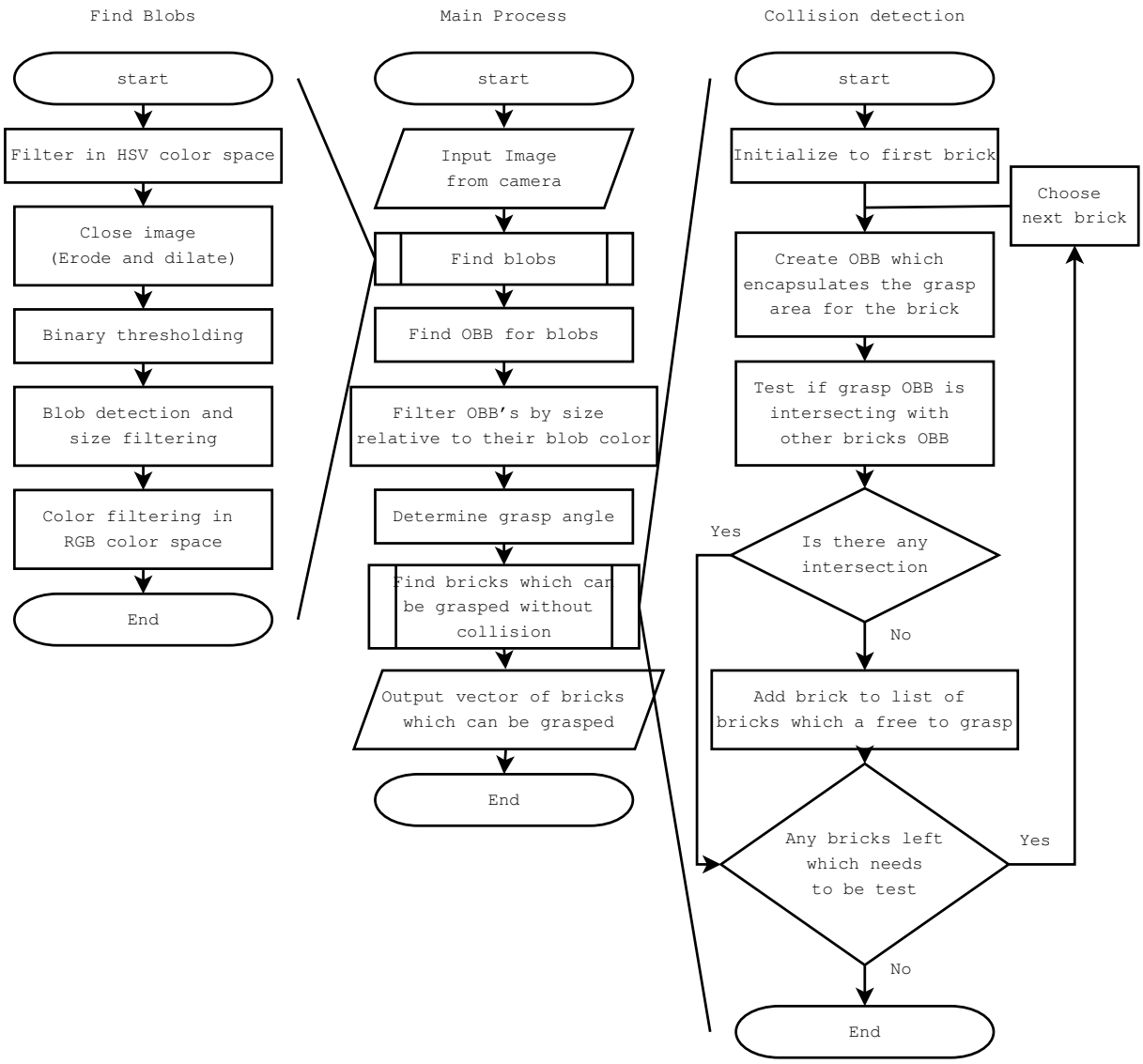


Figure 4.6: *Steps to be performed when an image is searched for LEGO bricks*

The vision node is able to deliver two services. One that uses a quick method to tell whether or not any bricks can be seen in the image and one that describes the image in full detail with all relevant brick information. This is used in order to save calculation time since there is no need to request the full vision algorithm if there is no bricks.

The quick method consists of only the left *find blobs* part in figure 4.6. This involves filtering in HSV space in order to separate the background elements from the LEGO bricks. Then the image is *closed* which consists of the erode and dilate process. A binary threshold is applied in order to get a binary representation of the image which is needed in blob detection. The blob detection algorithm detects the connected pixels in the image and separates them as blobs. The blobs are size filtered in order to get rid of small noisy contours. Lastly the color of the blob is determined by a custom color detection algorithm. The average RGB pixel value of all the pixels belonging the blob is calculated and the amount that the respective colors present in percentage with respect to the sum determine the correct color. Looking at the amount that the colors make up, a conclusion of the color can be made. This is robust with respect to changing light conditions and other changes to the colors.

The next step looks into fitting oriented-bounding-boxes (OBB) to the blobs, using an OpenCV[2]

implementation. This results in the size, orientation and center position of the detected element. The size and color combination is used for further brick filtering.

The orientation of the bricks is always with respect to the y-axis of the frame and smallest side of the bricks. This makes it easy for grasp calculation and the smallest side is chosen since some LEGO bricks can be too large for grasping on the longest side.

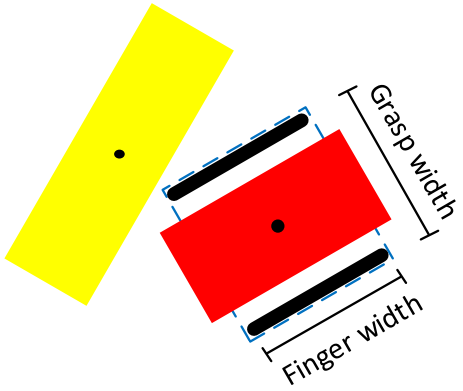
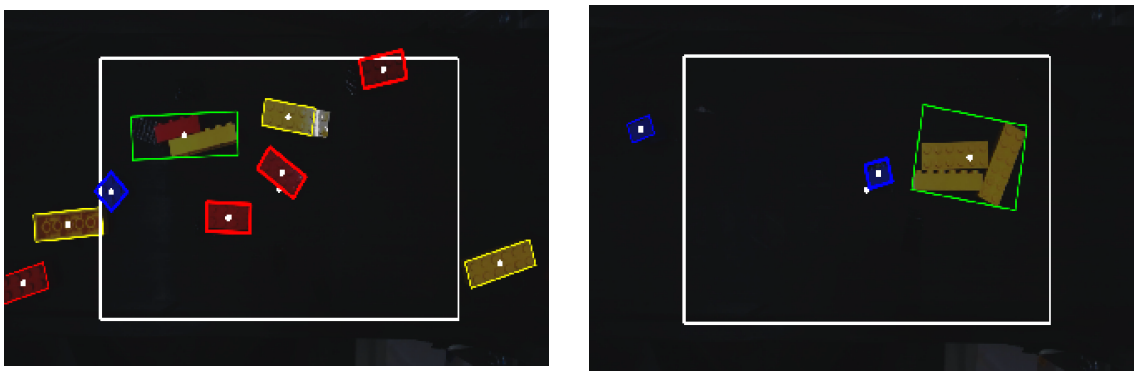


Figure 4.7: *LEGO brick collision detection method*

Finally collision avoidance is done by testing if any obstacles is in the grasp path for a brick. The grasp path for a brick is the area the two fingers of the end effector travels when trying to grasp the brick. Figure 4.7 shows a example where the two bold black lines illustrates the end effector fingers. The area encapsulated by the stripped blue lines is the grasp path area for the grasp when trying to grasp the red brick illustrated by a red rectangle. A grasp path area is then tested against all other bricks except the brick to be grasped. Separating axis theorem[21] is used to test if the rectangular grasp area is colliding with the rectangular brick areas defined by OBB's.

Since the positions of the bricks are found in the image and thus in pixels, it needs to be translated to the distance metric that the robot and gripper are using, namely meters. The pixel to meter conversion was archived using a simple method in which four corner points were marked on the conveyor and the distance between these was measured. The same distance was measured on the image in pixels. The different measurements was summed and the average scale found. It was assumed that if the set-up was kept static the scale would be valid for a longer period.

Figure 4.8a and 4.8b shows two results of the vision algorithm. The white box defines the area that a brick can be picked within. The contour of a detected brick is drawn with the color detected. The width of the contour if either thick or thin. Thick illustrates that the brick is pick-able thus not in collision and within the workspace. The green color marks a *monster*, a brick or collection of bricks that is not detected as a valid brick.



(a)

(b) *Two blue bricks and a monster*

Figure 4.8: *Example of vision output*

4.9 Robot motion and grasping

When a brick is detected and is in agreement with the order it needs to be picked and moved from the conveyor belt to the tipper of the mobile robot. This requires a number of robot and gripper actions.

Figure 4.9 shows the steps taken when a brick is to be picked.

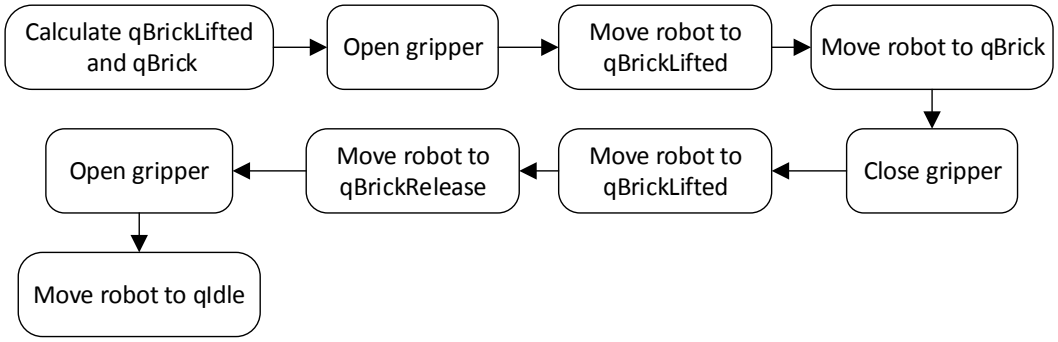


Figure 4.9: *Steps performed when a LEGO brick is to be picked*

The idle position of the robot is with the z-axis of the camera frame aligned with the z-axis of the conveyor frame but with a certain distance displacement, as seen in figure 4.3.

The calculation of *qBrickLifted* and *qBrick* includes inverse kinematics calculations used in order to get the position of the brick in the conveyor frame to a position in the robot frame with respect to the gripper. The configurations respectively represents the one in which the gripper and robot is aligned for grasping, but with a small offset in the z-axis and the configuration without the z-axis offset in which the gripper is in position for closing.

No path planning is used in the process since the work-cell is static. The area in which bricks can be picked from is limited thus eliminating wrongly detected bricks resulting in collision.

4.10 Main Control

The main component in the system is the one responsible for running the brick sorting application. The MES order is delivered to the main and it is responsible of performing the corresponding actions and starting the required services.

The brick sorting loop is only running when an order is present, the auto-mode is set, the security is okay and the robot is at the idle position. If that is the case the conveyor belt is started and the vision rough brick detector activated. When a brick is detected, the full algorithm is performed in order to match a brick with one requested in the order. The conveyor belt is only stopped if this is the case. This makes the brick sorting quite fast since a stop in the flow only occurs when a brick is to be picked.

A security stop will at any time result in a change to idle mode and a stop of all actions.

4.11 Human Machine Interface

A separate Human Machine Interface (HMI) is developed for controlling the robot cell. This allows the user to control relevant parameters from an intuitive GUI.

The HMI is implemented as a plugin to RobWorkStudio. This allows the HMI to act alongside the visual representation of the work-cell allowing a real-time simulation of the actions.

Figure 4.10 shows the HMI plugin. All system elements are using the log for error and information messages. The user is able to manually control the robot, conveyor and gripper. Some camera settings can be directly controlled from the HMI and the video feed shows a direct view of either the raw or vision applied image.

Figure 4.10: *HMI*

Chapter 5

Financial Aspects

The up and coming start-up "Super Smart SDU Robotics Graduates A/S" is planning to expand into a new business area. For this, the idea is to set up a a fully automated LEGO Bricks sorting and packaging line. People will be able to send in their old LEGO bricks and get paid by weight. The income comes from customers ordering very specific combinations of LEGO bricks at a far lower price point than ordering them directly at LEGO or at a toy store.

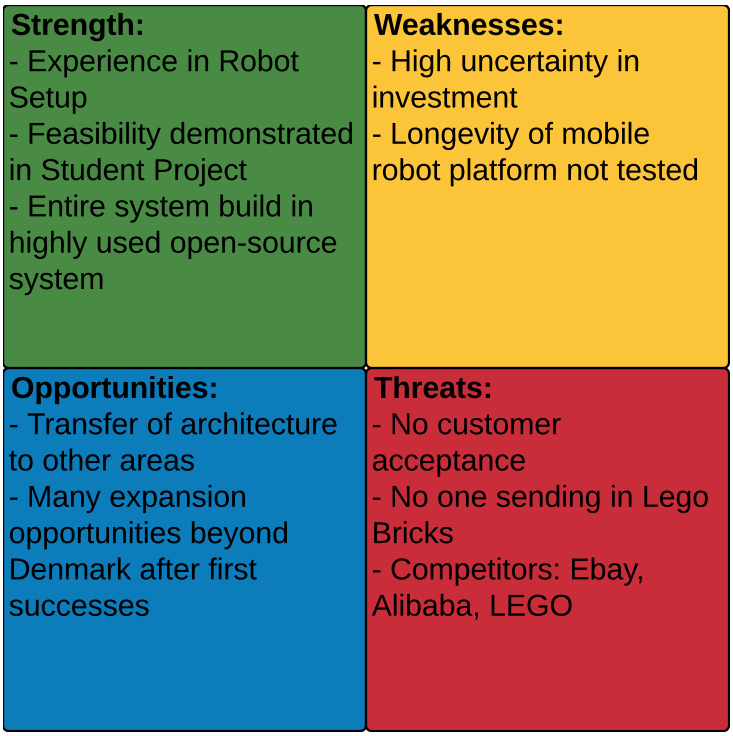


Figure 5.1: *SWOT-Analysis*

The SWOT-Analysis was done within the management and an external consultant as seen in figure 5.1 A careful organized market research has been done to analyse the feasibility of said proposal. These results of the market research, table 5.1, shows both the potential annual income as well as estimated investment cost. The labour cost are based on an engineering salary of DKK 1.500 per hour. For the set-up, 200 engineer-hours are budgeted and for maintenance one hour per week.

Name	Type	Cash Flow	Anual increase
Kuka Robot	one-time	-250.000	
Frobit Robot	one-time	-100.000	
Set-up labor cost	one-time	-300.000	
Maintenance - Hardware	anual	-10.000	4%
Maintenance - Labor	anual	-78.000	5%
Income	anual	250.000	10%

Table 5.1: *Market Research*

Salary is set to increase by 5% and the maintenance annually by 4%. These values are chosen very conservative (above the raw material price index for maintenance material and above the average salary increase over the past five years) to allow for fluctuation and uncertainties.
The income growth over the first five years is set to constant 10% even though a steeper increase is expected after the first market penetration.

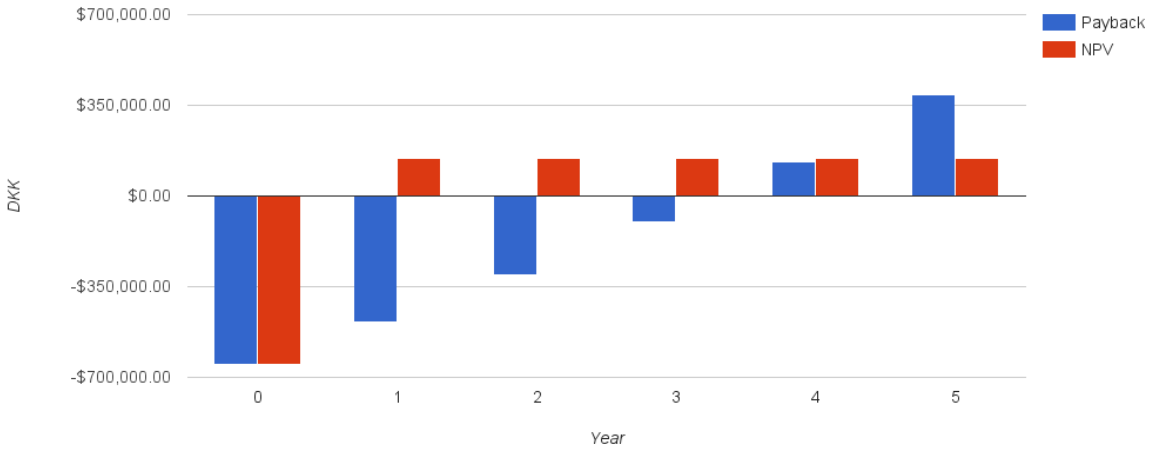


Figure 5.2: *Payback Method and Net Present Value*

Given these assumptions the pay-back method 5.2 (blue), the internal rate of return method (IRR) and the net-present value (NPV) 5.2 (red) are calculated to determine the feasibility of this investment. The pay-back method shows a profit of DKK 390.000 after five years. The investment was judged as having a medium risk, but for calculating the NPV the required rate of return of investment has been set to 12% to be conservative. With this the NPV of this investment after considering five years with the given assumptions would be DKK 80.000 and thus the investment is seen as favourable. The IRR is 16.47% , this far exceeds simply investing in bonds, or paying back current bank loans, of which the highest is DKK $2.000.000$ at 7%. All methods suggest this investment is profitable and the financial department gives the project the go-ahead.

Chapter 6

System Test

In order to prove the capabilities of our design and to bring all the so far undetected bugs and errors to light, a 24 hour integration test was required. During these 24 hours, all components of the system had to be used throughout multiple iterations of the brick selection and delivery procedure.

Since the timespan of the test was such that it encapsulated a whole day, the systems had to cope with changing light conditions, continuous interference by human operators and spectators moving around, degradation of navigation markers and the mobile platforms of the other groups. In the following sections a detailed description of the performed tests are present alongside the results and evaluation.

6.1 Test description

In the beginning of the test, all mobile platforms were situated at the charging stations, fully charged. One iteration consisted of the following milestones:

1. Leaving the charging station in a fully charged state and taking up position under the LEGO dispenser machine
2. Waiting for the bricks to be loaded on the tipper (since the dispenser was not operational, we just simulated the process)
3. Delivering the LEGO to the robot cell
4. Unloading the bricks to the conveyor belt
5. Moving the bricks into position for selection through robotic vision
6. Picking the LEGO bricks specified by the order (this is equivalent to loading them on the mobile platform)
7. Transporting the LEGO back to the box, and docking at the charger

All components of the system were continuously broadcasting location and status information, which was displayed on the HMI, while the map used for SLAM navigation and the console-printed ROS information were also monitored. We compared all these with the actual position and behaviour of the system components, logged their performance, and intervened when errors occurred.

6.2 Log

The date and time was logged alongside the person responsible for creating the entry, the type of the entry and a short note describing the reason for making it.

The following types are distinguished between:

- Order Start
- Order Finish
- Error
- Manual Shutdown
- System Event
- Other

6.3 Test results

Even though in the initial phase camera connection issues caused delays in the testing process, all in all 49 iterations were performed, out of which 36 were finished successfully resulting in a success ratio of 73.47% (details see Figure 6.1). The full log can be seen in Appendix B

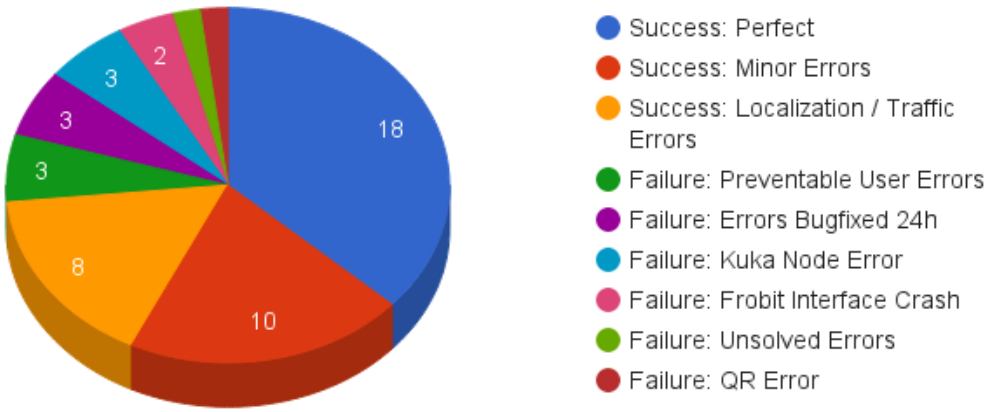


Figure 6.1: *Stress Test Results*

The speed of the system was improved throughout the test and on average an execution time of 9.28 minutes was achieved with a standard deviation of 3.35. The highest factor in deviation in time as well as the highest error rate was due to localization errors and traffic jams inside and around the box. A combined MES-Server, that ensures only one robot moving around in the box would solve this issue completely.

Major Lessons learned during 24-hour stress test:

- Protocol for recovery after breaking safety
- Reboot procedure
- Protocol for hot-swapping Battery
- Ensure QR-tag readability

Bugs fixed or improved during 24-hour stress test:

- Inverse Kinematics error detection and handling
- Increased Kuka Speed
- Adjusted required battery level to make one run
- QR Codes replaced

Open To-Do's before shipping:

- Improve vision for better *monster* recognition
- Fix loose connections
- Solve unloading issue
- Solve Kuka Node crashing
- Fix Frobit Interface Crashes
- MR GO node crashed once, no reason / solution found

Chapter 7

Discussion

In this chapter the discussion about the final status of the project, as well as future work or improvements, is presented. Following the structure of the report this is divided in two parts: (1) the mobile robot and (2) the robot work-cell.

7.1 Mobile robot

From the 24 hour test a number of issues arose. These were lack of runtime due to the battery, collisions with other robots and minor errors in QR detection and unloading of bricks. The battery issue caused a lot of time spend recharging during the test. This could be resolved by adding better batteries or reduce the power consumption. Colliding with other robots often happened when they had past each other and could no longer detect each other with the LIDAR. This could be helped by adding another LIDAR to give the robot 360° vision. This could possibly also increase the the precision of the AMCL as more data points would be available. This could be a particular advantage inside the box. The marker system could also be changed to cover that area, which would give it a useful capability. The problem of reading QR codes while moving could be solved by using a global shutter camera. This would significantly reduce the motion blur. The mobile platform itself is clearly a prototype. If is was to operate it should be protected by an outer shield to protect the wiring on the robot. It could also be a useful development to make the robot modular to allow users or integrators to adapt the system for specific uses.

In the software part of the mobile robot improvements have been considered in free navigation by using multiple local planners. By having multiple planners they could be optimized for each stage of navigation. The HMI can be improved in the UI side as well as the functionality. A web service for controlling not only the mobile robot, but the robot workcell and the MES Server can be done.

7.2 Robot workcell

The role of the robot cell was to simply detect and select the bricks that matched the order sent by the MES Server. This was executed by the vision detection task and the grasping task using the gripper and robot.

The vision algorithm being able of detecting and identifying bricks while the conveyor belt was running, performed well. A precise calibration of the scene, camera and gripper was done resulting in a very small number of erroneous grasps. The vision was independent of daylight fluctuation and no brick was marked collision-free while not being so.

There is however room for improvements. The current implementation of the system allows only grasps on the center of the thin side on the bricks. This results in a large number of bricks being ignored due to a grasp attempt will lead to collision.

In order to increase the speed of the brick sorting the following can be considered. The speed of the gripper closing and opening was quite slow. Increasing this would led to faster brick selection. A suction based gripper would allow bricks to be picked even if no clear area is available around the brick. This requires the vision to be able to distinguish between bricks even if a bunch of same-coloured bricks are presented.

A more low-level solution could consist of a better hardware set-up that results in more separated bricks thus allowing more bricks to be picked.

Finally in order to increase multitasking, an eye-to-hand mounting of the camera could have been chosen. This would allow the robot to grasp and move bricks while the vision could analyse the next move.

Chapter 8

Conclusion

The tasks of the mobile platform was the transportation and loading of the LEGO bricks to be sorted. In transporting it would have to operate on two different kinds of navigation environments. One being guided by black lines and one in an unprepared area. The mobile robot system was able to successfully navigate in both these areas thus completing its transportation role. Though the execution during 24 hour test was not absolutely flawless it did perform reasonably well. A large contributor to navigation errors was traffic jam or collisions with other robot systems.

The loading of bricks onto the conveyor was done by modifying the tipper on the original FrobitPro. After the modifications the transfer was achieved repeatedly during the final test.

All in all the mobile robot was able to go to the brick dispenser, transport bricks in the open area and along the guide lines to the conveyor where they were unloaded. It could then go to the robot to pick up the final order and transport it back to the charging station where it would stay and charge until battery level was sufficient for the next order.

The robot cell developed is able to; receive unsorted bricks from the mobile platform, pick the correct bricks according to order specification from MES server and placing the order back on the mobile robot. All this functionality can operate automatically or supervised by an user through the developed HMI. The 24 hour integration test showed a fairly stable Robot cell system with mostly minor errors which could be resolved and recovered from manually.

Bibliography

[1] "Trello," https://trello.com/.

[2] "OpenCV," http://www.opencv.org/.

[3] "zlib," http://www.zlib.net/.

[4] "SICK TiM Ros Package," http://wiki.ros.org/sick_tim.

[5] "Gmapping ROS-Package," http://wiki.ros.org/gmapping.

[6] "Openslam Gmapping," https://www.openslam.org/gmapping.html.

[7] "Navigation stack," http://wiki.ros.org/navigation.

[8] B. P. Gerkey, "AMCL," http://wiki.ros.org/amcl.

[9] E. Marder-Eppstein, "move_base," http://wiki.ros.org/move_base.

[10] "HTML," http://www.w3.org/html/.

[11] "CSS," http://www.w3.org/Style/CSS/.

[12] "JavaScript," https://www.javascript.com/.

[13] "Python," https://www.python.org/.

[14] "Robot Operating System," http://www.ros.org/.

[15] "WebSocket," https://tools.ietf.org/html/rfc6455.

[16] "Autobahn," http://autobahn.ws/.

[17] "Twisted," https://twistedmatrix.com/trac/.

[18] "Video4Linux," http://linuxtv.org/wiki/index.php/Development:_Video4Linux_APIs.

[19] P. M. James Bowman, "camera_Calibration," http://wiki.ros.org/camera_calibration.

[20] J. L. Patrick Mihelich, Kurt Konolige, "image_Proc," http://wiki.ros.org/image_proc.

[21] S. Gottschalk, M. C. Lin, and D. Manocha, "Obbtree: A hierarchical structure for rapid interference detection," in *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '96. New York, NY, USA: ACM, 1996, pp. 171–180. [Online]. Available: http://doi.acm.org/10.1145/237170.237244

Appendix A

A.1 Buttons board's LED meaning

- **White, not blinking**: The robot is in *idle* mode.
- **Green, blinking**: The robot is in *auto* mode.
- **Purple, blinking**: The robot is in *manual* mode.
- **Yellow**: The robot is in *proximity alert*.
- **Red**: The robot is in *collision*.

Appendix B

B.1 24 Hours test log

Date and Time	Person	Type	Note	Time
2015-12-07 09:06:22	Dominik	Error	Camera Connection Issues	
2015-12-07 09:06:53	Dominik	Order Start	First Order	
2015-12-07 09:08:23	Dominik	Error	Start node incorrectly set up	
2015-12-07 09:10:12	Dominik	Order Start	First Order Attempt #2	
2015-12-07 09:11:31	Dominik	Error	Accidently sent to wrong WC, aborted	
2015-12-07 09:17:49	Dominik	Order Start	First Order Attempt #3	
2015-12-07 09:19:56	Dominik	Error	Manual interrupt of safety fence	
2015-12-07 09:20:18	Steffen	Other	System recovery / turn on safety, recalibrate robot and continue order	
2015-12-07 09:23:05	Steffen	Error	Vision error / trying to grab a monster	
2015-12-07 09:23:24	Steffen	System Event	Emergency Stop / manual mes server order done feedback	
2015-12-07 09:27:34	Steffen	Order Finish	Successful order #1	0:09:45
2015-12-07 09:38:05	Steffen	System Event	Beer opened	
2015-12-07 09:47:04	Dominik	Order Start	Order started	
2015-12-07 09:53:09	Steffen	Order Finish	No mistake ??Celebration	0:06:05
2015-12-07 10:01:22	Dominik	Order Start	Order sent, waiting for battery level	
2015-12-07 10:05:33	Steffen	Other	#99 Beer finished	
2015-12-07 10:08:22	Steffen	Other	Jorge is messing in code	
2015-12-07 10:21:29	Yonas	Error	Bug in battery charge level handler (fixed that)	
2015-12-07 10:23:58	Dominik	Order Start	Order started	
2015-12-07 10:33:24	Dominik	Order Finish	No mistakes	0:09:26
2015-12-07 10:45:53	Dominik	Order Start	Order started	
2015-12-07 10:52:21	Dominik	Error	Inverse Kinematics failed (fix being applied)	
2015-12-07 10:57:00	Dominik	Error	Battery at 10.9	
2015-12-07 10:58:10	Dominik	Error	Change Battery, system still online	
2015-12-07 11:11:59	Dominik	Manual Shutdown	Changes to Kuka Speed	
2015-12-07 11:12:24	Dominik	Order Start	New Order	
2015-12-07 11:15:38	Dominik	Error	Tipper got stuck (Tipper Motor plug disconnected)	
2015-12-07 11:28:12	Dominik	Order Start	New Order	
2015-12-07 11:38:44	Steffen	Order Finish	No mistakes	0:10:32
2015-12-07 11:51:58	Dominik	Order Start	Order started	
2015-12-07 12:00:21	Dominik	Order Finish	Successful order	0:08:23
2015-12-07 12:00:51	Dominik	Order Start	Order started - waiting for Battery to charge	
2015-12-07 12:44:48	Dominik	Error	Order does not get executed after charging is finished. Debugging	
2015-12-07 12:50:03	Dominik	Manual Shutdown	Fix applied, rebooting	
2015-12-07 13:12:06	Dominik	Error	SSH / Boot problems	
2015-12-07 13:22:45	Dominik	Error	IMU was misfunctioning / unplug plug	
2015-12-07 13:23:07	Dominik	Order Start	New Order	
2015-12-07 13:28:01	Dominik	Error	Inverse Kinematics failed (previous fix was not working)	
2015-12-07 13:32:14	Dominik	Error	Localization failed due to people walking inside the box	
2015-12-07 13:32:39	Dominik	Order Finish	Finished with failures	0:09:32
2015-12-07 14:16:52	Dominik	Order Start	New Order	
2015-12-07 14:37:09	Dominik	Error	Unable to recover localization	
2015-12-07 14:52:44	Dominik	Order Start	New Order	

Date and Time	Person	Type	Note	Time
2015-12-07 14:57:48	Dominik	Order Finish	Finished without error	0:05:04
2015-12-07 15:17:00	Dominik	Order Start	New Order	
2015-12-07 15:27:26	Dominik	Error	Charging time Adjustment to speed up process failed	
2015-12-07 15:27:47	Dominik	Order Finish	Finished Order with above mentioned error	0:10:47
2015-12-07 15:40:25	Martin	Order Start		
2015-12-07 15:48:27	Martin	Error	Robot arm dropped a brick. It was on the edge of the conveyer	
2015-12-07 15:52:36	Martin	Order Finish	Success. One brick missing due to the gripper dropped it.	0:12:11
2015-12-07 16:13:33	Martin	Order Start		
2015-12-07 16:20:06	Dominik	Order Finish	Pure Success, No Mishaps, no Comments(except this one)	0:06:33
2015-12-07 16:35:28	Thor	Order Start		
2015-12-07 16:38:31	Thor	Error	1 brick was not unloaded to the convezor	
2015-12-07 16:42:46	Thor	Order Finish	Success. One brick too much due to previous error	0:07:17
2015-12-07 16:55:06	Thor	Order Start		
2015-12-07 17:03:20	Thor	Order Finish	Success.	0:08:14
2015-12-07 17:18:53	Martin	Order Start		
2015-12-07 17:22:04	Martin	Error	Some bricks was not unloaded	
2015-12-07 17:25:25	Martin	Order Finish	No errors apart from unloading	0:06:32
2015-12-07 17:38:36	Martin	Order Start		
2015-12-07 17:40:30	Martin	Error	The robot got stuck inside the box. Needed help with pose estimation	
2015-12-07 17:46:23	Martin	Error	MR did not return from wc. Communication error, solved manually.	
2015-12-07 17:48:42	Martin	Order Finish	See errors above	0:10:06
2015-12-07 17:56:24	Martin	Order Start		
2015-12-07 18:08:57	Martin	Error	Unable to navigate the Frobit in the box. Needed a push to get free	
2015-12-07 18:13:51	Dominik	Order Finish	Traffic jam inside box, otherwise fine	0:17:27
2015-12-07 19:44:11	Dominik	Order Start		
2015-12-07 19:50:30	Dominik	Order Finish	Success no errors	0:06:19
2015-12-07 19:51:51	Dominik	Order Start		
2015-12-07 19:53:43	Dominik	Error	localization failed, needed to reset location	
2015-12-07 20:00:28	Dominik	Order Finish	Success, above error	0:08:37
2015-12-07 21:02:14	Dominik	Order Start		
2015-12-07 21:11:37	Dominik	Error	A robot drove into ours inside the box	
2015-12-07 21:13:04	Dominik	Order Finish	Finished with above mentioned error	0:10:50
2015-12-07 21:13:47	Dominik	Order Start		
2015-12-07 21:20:59	Dominik	Error	Kuka node crashed - Order abborted	
2015-12-07 21:34:11	Dominik	Order Start		
2015-12-07 21:41:34	Dominik	Order Finish	without errors	0:07:23
2015-12-07 22:27:43	Dominik	Order Start		
2015-12-07 22:35:50	Martin	Error	Missed a QR CODE. Was worn down.	
2015-12-07 22:38:17	Dominik	Order Finish	nailed it	0:10:34
2015-12-07 22:38:52	Dominik	Other	QR Code replaced	

Date and Time	Person	Type	Note	Time
2015-12-07 22:39:57	Steffen	Other	#000 Self awareness achieved. Robot cell is running out of control. We have taken the necessary precautions and shut down the system	
2015-12-07 22:40:52	Steffen	Order Start		
2015-12-07 22:41:28	Steffen	Other	#000 Shutdown failed. Robot power supply independant of household electronics.	
2015-12-07 22:42:22	Steffen	Other	#000 All hope is lost	
2015-12-07 22:42:54	Steffen	System Event		
2015-12-07 22:43:38	Steffen	Order Finish	Returning 0 bricks, because. Robot cell still working independantly	0:02:47
2015-12-07 22:44:24	Steffen	System Event	Yonas did Martin.. hard	
2015-12-07 22:44:44	Steffen	Order Start	New order sent	
2015-12-07 22:45:08	Steffen	Other	#000 Robot is rebelling. Attacking robot cell 2. HELP	
2015-12-07 22:45:47	Steffen	Other	#000 Robot in cell 2 seems infected with same AI as ours	
2015-12-07 22:47:01	Steffen	System Event	Yonas cheated	
2015-12-07 22:47:23	Steffen	Other	#000 Robots are multiplying in strength and numbers. Nitrogen freeze considered. GOD HELP US ALL	
2015-12-07 22:54:05	Martin	Error	hit another robot	
2015-12-07 22:54:31	Martin	Order Finish		0:09:47
2015-12-07 22:57:20	Dominik	Order Start		
2015-12-07 23:07:41	Dominik	Other	Other rebootet our Mes server , we sent the MES messages to successfully finish order	
2015-12-07 23:16:13	Dominik	Order Finish	After a bit of a longer mexican stand off, it finished successfully	0:18:53
2015-12-07 23:57:06	Dominik	Order Start		
2015-12-08 00:07:59	Dominik	Error	Kuka node died due to communication timeout	
2015-12-08 00:15:11	Dominik	Order Finish	Traffic jam, but it finished	0:18:04
2015-12-08 00:45:21	Dominik	Order Start		
2015-12-08 01:01:37	Martin	Error	Localisation lost due to other robot and human interference. Manual pose estimation help	
2015-12-08 01:02:55	Martin	Error	Robot arm down. Communication error.	
2015-12-08 01:03:21	Martin	Error	Localization failed due to human traffik (we think)	
2015-12-08 01:28:44	Dominik	Order Start		
2015-12-08 01:36:50	Dominik	Order Finish	awe - wait for it - quite a bit	0:08:06
2015-12-08 01:59:54	Dominik	Order Start		
2015-12-08 02:04:34	Dominik	Error	After illegalliy bugfixing for a while, the PLC power plug was accidently pulled out with the foot (or sabotage, we are still investigating)	
2015-12-08 02:16:03	Dominik	Order Start		
2015-12-08 02:21:51	Dominik	Order Finish	nailed it	0:05:47
2015-12-08 02:26:28	Dominik	Order Start		
2015-12-08 02:34:28	Dominik	Order Finish	recieved it, executed it, delivered it	0:08:01
2015-12-08 02:42:18	Dominik	Order Start		
2015-12-08 02:52:57	Dominik	Error	Traffic Jam in Box	
2015-12-08 02:53:09	Dominik	Order Finish	Worked fine, expect traffic jam	0:10:51
2015-12-08 03:57:56	Dominik	Order Start		
2015-12-08 04:04:57	Dominik	Error	Frobit Interface crashed	
2015-12-08 04:07:02	Martin	Order Start		

Date and Time	Person	Type	Note	Time
2015-12-08 04:11:26	Martin	Error	Robot was stuck. Manually moved.	
2015-12-08 04:18:05	Martin	Order Finish	Some Localization error in the beginning, then fine	0:11:03
2015-12-08 04:22:56	Dominik	Order Start		
2015-12-08 04:32:32	Dominik	Order Finish	Mission Accomplished	0:09:36
2015-12-08 04:42:33	Martin	Order Start		
2015-12-08 04:46:24	Dominik	Error	Gripper failed to grasp 1 brick	
2015-12-08 04:49:39	Martin	Error	Crash into enemy frobit	
2015-12-08 04:50:21	Martin	Order Finish		0:07:48
2015-12-08 05:01:19	Dominik	Order Start		
2015-12-08 05:11:42	Dominik	Error	Frobit Interface crashed	
2015-12-08 05:21:42	Dominik	Manual Shutdown	Reboot because of Frobit Interface Error, causes some strange connection issues (cant connect vnc,	
2015-12-08 05:36:13	Dominik	Order Start		
2015-12-08 05:41:35	Martin	Error	Dropped all bricks to soon. Mr go or nav controller failed	
2015-12-08 06:17:28	Dominik	Order Start		
2015-12-08 06:25:38	Dominik	Order Finish		0:08:10
2015-12-08 06:33:40	Dominik	Order Start		
2015-12-08 06:36:40	Dominik	Error	Robot Cell Unresponsive	
2015-12-08 06:38:34	Dominik	Order Finish	Nothing delivered, but it kind of work. right?	0:04:54
2015-12-08 06:43:17	Dominik	Order Start		
2015-12-08 06:59:05	Dominik	Order Finish	Some minor traffic issues	0:15:48
2015-12-08 07:22:35	Thor	Order Start		
2015-12-08 07:26:35	Dominik	Error	dropped one brick	
2015-12-08 07:30:20	Dominik	Order Finish	Otherwise fine	0:07:45
2015-12-08 07:42:55	Dominik	Order Start		
2015-12-08 07:54:40	Dominik	Order Finish	Initialization in the beginning was off, then it worked fine	0:11:45