Interreg 2 Seas Mers Zeeën
INCASE
European Regional Development Fund

2018

D. 1.5.1. Set of basic low-cost mobile robots using integrated design methods

ISEN-Lille : Annemarie Kokosy, Gilles Tagne, Bruno Stefanelli

Université de Lille : Jérémie Boulanger, Benjamin Mathon

KU Leuven : Philippe Saey, Frederic Depuydt, Mathieu Troch, Jos De Brabanter

31/03/2018

Overview

Introduction

This report presents the results of work on “integrated design” methods applied to low-cost mobile robots achieved by the teams from Yncréa Hauts-de-France/ISEN-Lille, KU Leuven and University of Lille.

In the first section, the methodology to design algorithms to control the robot actuators as well as the robot trajectory is described and implemented in Simulink. A S-function in Simulink has been developed to allow the connection of a robotic system as a slave in the PROFIBUS industrial network. Thanks to that block, it is now possible to connect a robot having a low-cost non-industrial hardware (for example NXT, Arduino or Raspberry) to an industrial network.

In the second section, 4 different robotic systems are presented. They use different low-cost hardware board for control: an Arduino, a NXT or a Raspberry Pi3. Each robotic system is controlled by MATLAB/Simulink using a rapid prototyping methodology. The first two robots run the same algorithm implemented in Simulink and deployed in their own hardware target in order to illustrate that is possible, with very few changes, to make ready for use robots having different hardware. This clearly illustrates flexibility and speed of adaptation, reducing “time to market” for new applications.

The last section describes two systems controlled by a PLC using an Industrial network, Modbus and PROFIBUS, which creates new opportunities for applications.

1. Robot control algorithms using MATLAB/Simulink

1.1. Basic motor control

The first goal when the aim is to design an autonomous robot, is to ensure the low-level control. It means that first of all, the robot's wheels need to keep the desired speed.

In order to ensure it, a controller needs to be design. The key steps to implement a controller are showed in the following figure

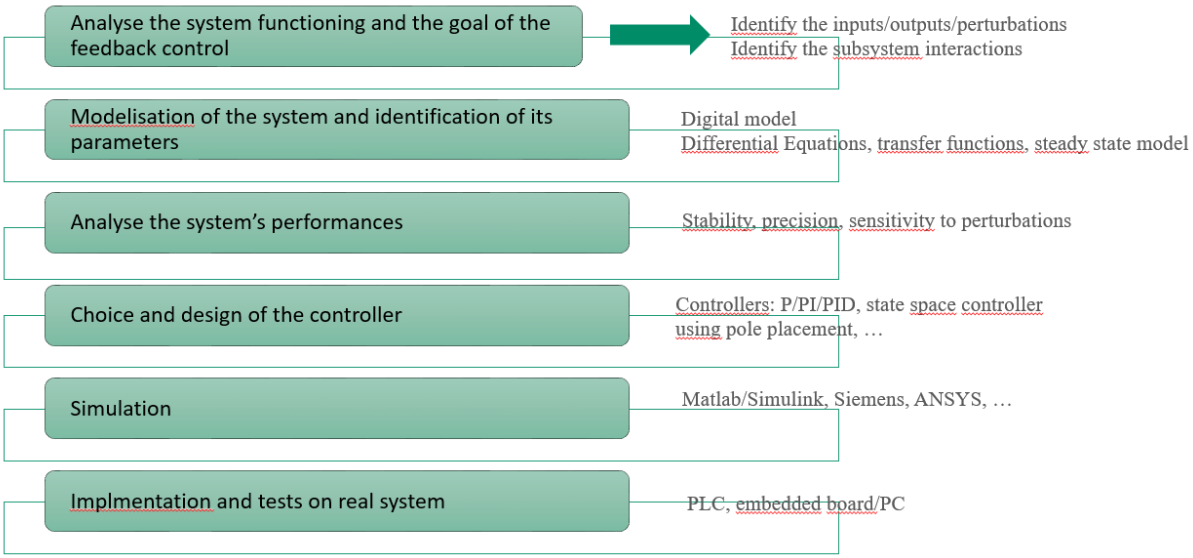


Figure 1: Key steps to implement a controller

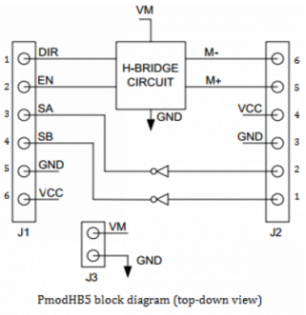
The small mobile robots mainly use DC motors. To control their speed, a PI controller should be designed using MATLAB/Simulink rapid prototyping method.

In order to illustrate how to implement a PI controller using MATLAB, Simulink, the following equipment is used:

- DC motor including an encoder TRENZ-24142, 6V, 240mA, ratio: 53, controlled by a PWM
- Motor Driver PMODHB5; the pinout description table is showed below
- An Arduino Uno
- A PC with MATLAB/Simulink, version 2016b

Pinout Description Table

Header J1 (pin 1 on the top)			Header J2 (pin 1 on the bottom)		
Pin	Signal	Description	Pin	Signal	Description
1	DIR	Direction pin	1	SB	Sensor B feedback pin
2	EN	Enable pin	2	SA	Sensor A feedback pin
3	SA	Sensor A feedback pin	3	GND	Power Supply Ground
4	SB	Sensor B feedback pin	4	VCC	Positive Power Supply (3.3/5V)
5	GND	Power Supply Ground	5	M+	Motor positive pin
6	VCC	Positive Power Supply (3.3/5V)	6	M-	Motor negative pin



PmodHB5 block diagram (top-down view)

The overall system is depicted in Figure 2.

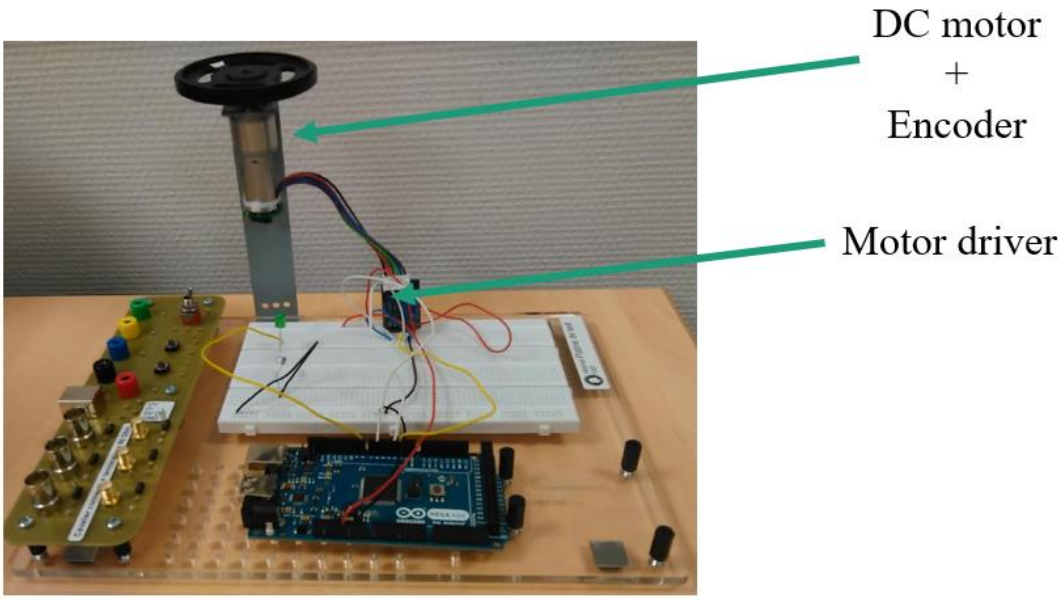
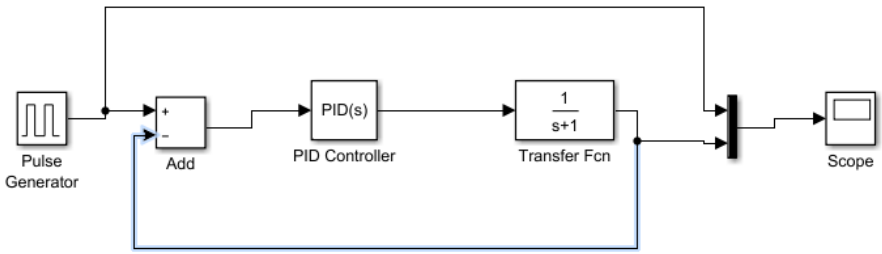


Figure 2: Bench with DC motor speed control using an Arduino Uno

To design a controller, it is necessary to determine a mathematical model of the DC motor and the driver. Once this model is available, the controller could be designed using Simulink. The simulation scheme is showed below. The controller should guarantee that the closed loop system reaches the requested performances.



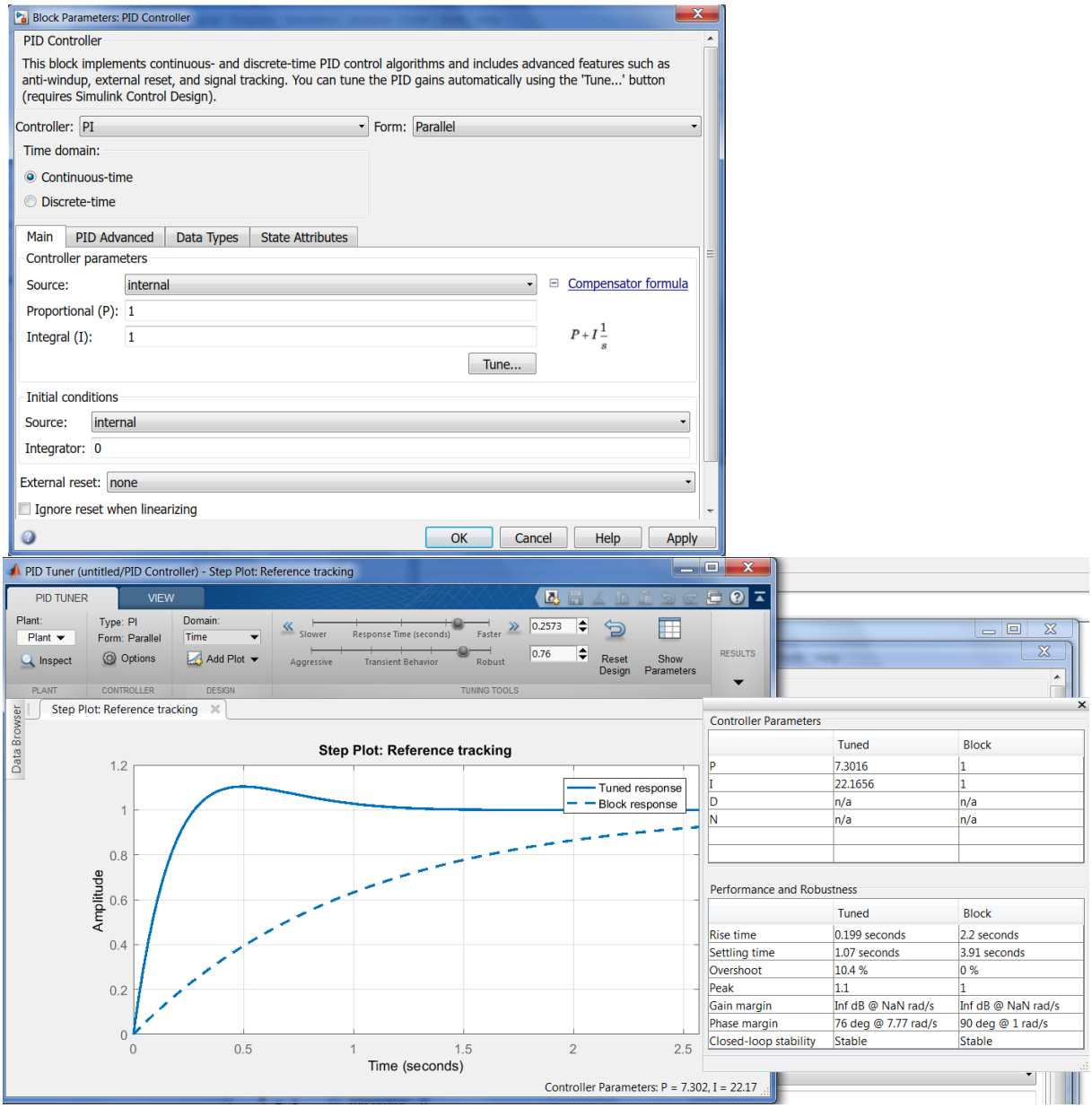


Figure 3: PI controller tune in Simulink

1.2. Trajectory control

Once the speed controller designed and implemented, it is now possible to control the robot's trajectory. The robot, which need to be controlled, is a non-holonom robot having two motorized wheels. The robot's direction is controlled by using the differential speed of the wheels.

The linear and angular speed, v and ω, of the robot are calculated using the following equation, where vr is the speed of the right wheel, vl is the speed of the left wheel and d is the distance between the 2 wheels.

$$v = \frac{v_l + v_r}{2}$$

$$\omega = \frac{v_l - v_r}{\mathrm{d}}$$

1.2.1. Straight-line travel

In order to move straight the robot, it is necessary to put the same speed to the 2 wheels. As, it is possible that the two wheels are not controlled in a synchronized way, a supplementary control needs to be implemented which will reduce the speed difference between the two motors.

The Simulink scheme implementing the straight-line travel algorithm is showed in Figure 4. The

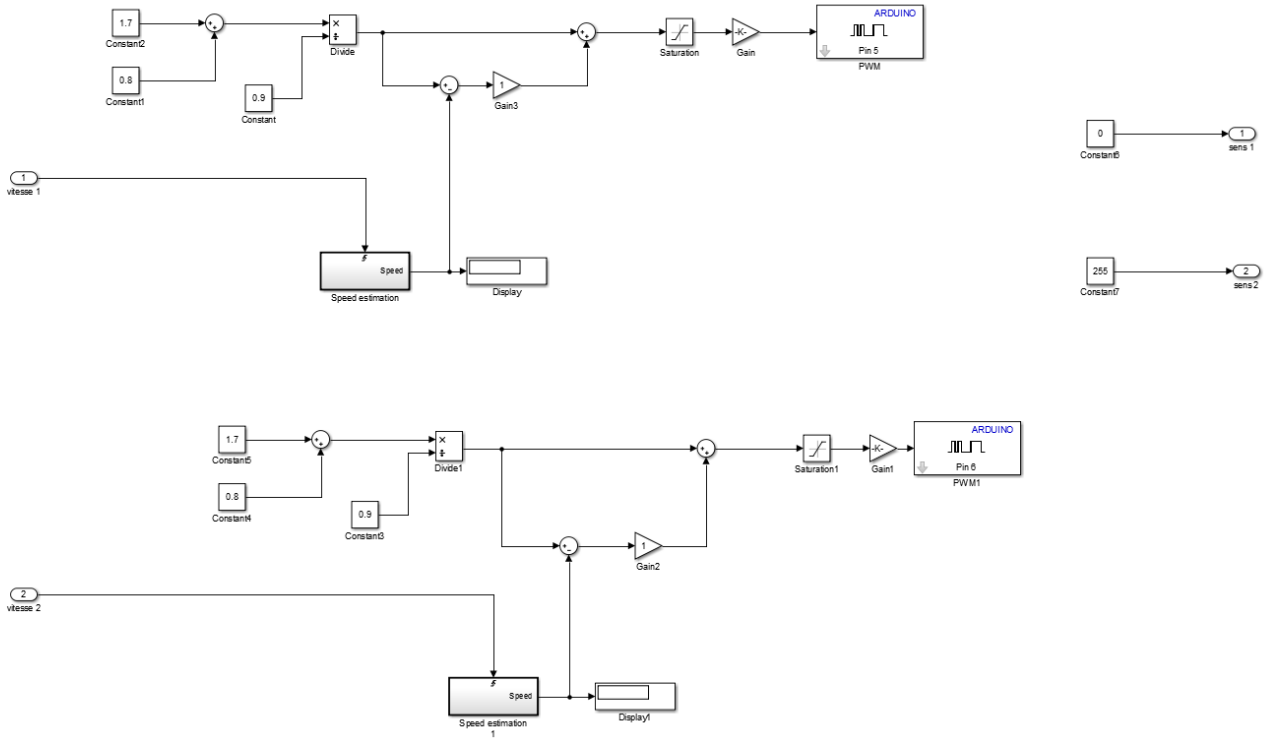


Figure 4: Algorithm of a robot straight-line travel implemented in Simulink

1.2.2. Line follower

If the robot needs to follow a line, it needs to be able to sense the line using suited sensors. One of those sensors is the IR sensor. Thanks to it, the robot could distinguishes between two different colours having a contrast good enough (for example black line on a light blue floor).

The robot needs to have at least 2 IR to be able to know if it is following the line or it starts to deviate to the left or to the right.

If the robot deviates from the line to the right, it needs to correct it's direction to the right. If the robot deviates from the line to the right, it needs to correct it's direction by going to the left. More the deviation is important, more the level of the correction needs to be important too.

In order to implement the line follower algorithm a state machine is used. The finite state machine is a representation of an event-driven (reactive) system. In an event-driven system, the system responds to an event by making a transition from one state (mode) to another. This transition occurs if the condition defining the change is true.

A Stateflow chart is a graphical representation of a finite state machine. States and transitions form the basic elements of the system.

The Stateflow chart provides aditional capabilities beyond traditional finite state machines, such as:

- Modelling hierarchical states for large scale systems;
- Adding to define decision logic;
- Define orthogonal states to represent systems with parallelism.

Figure 5 shows an example of a flow chart. The system switches from state 0 to state 1 if the condition of the transition **a** is satisfied. When it arrives in state 1, the system executes the action 1.

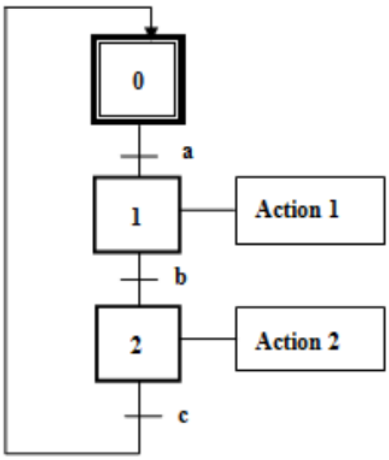


Figure 5: Example of a stateflow chart

The line follower algorithm is depicted in Figure 6. The transitions are defined by the data provided by the IR sensors. The robot has 4 different actions which it can do: **go straight**, **turn left**, **turn right** and **switch off**.

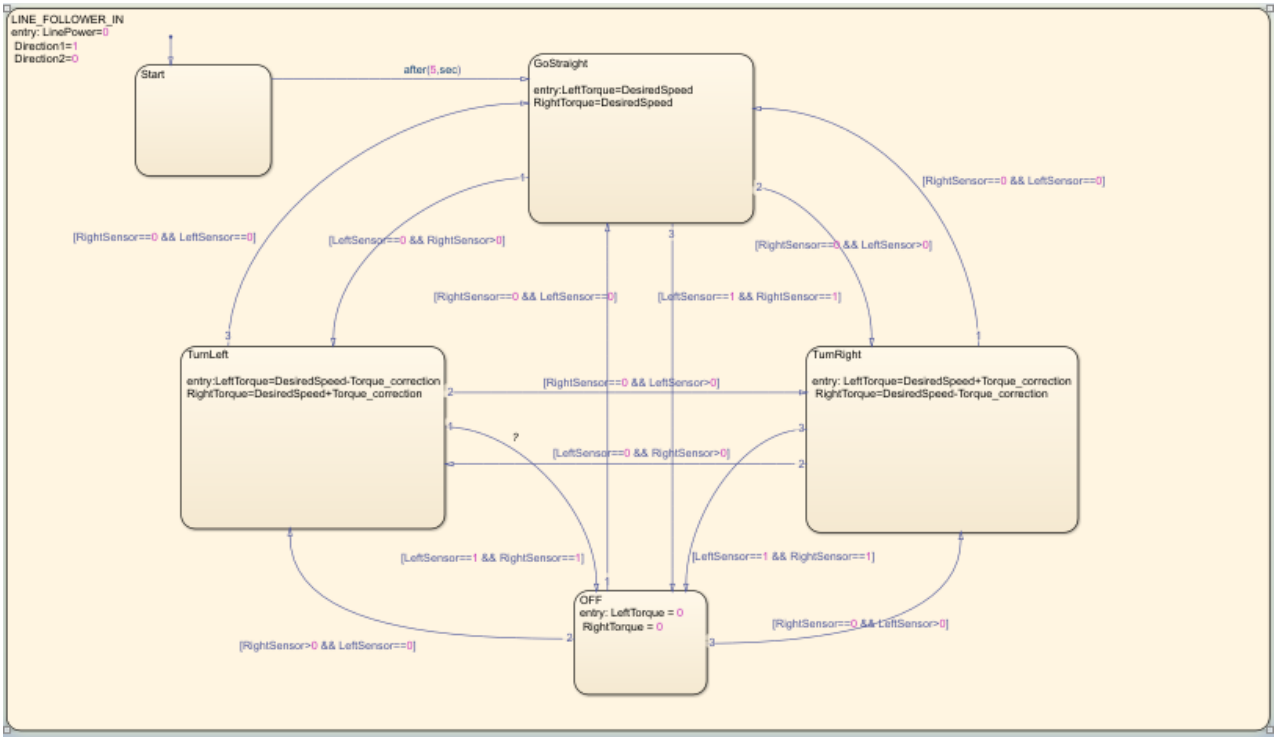


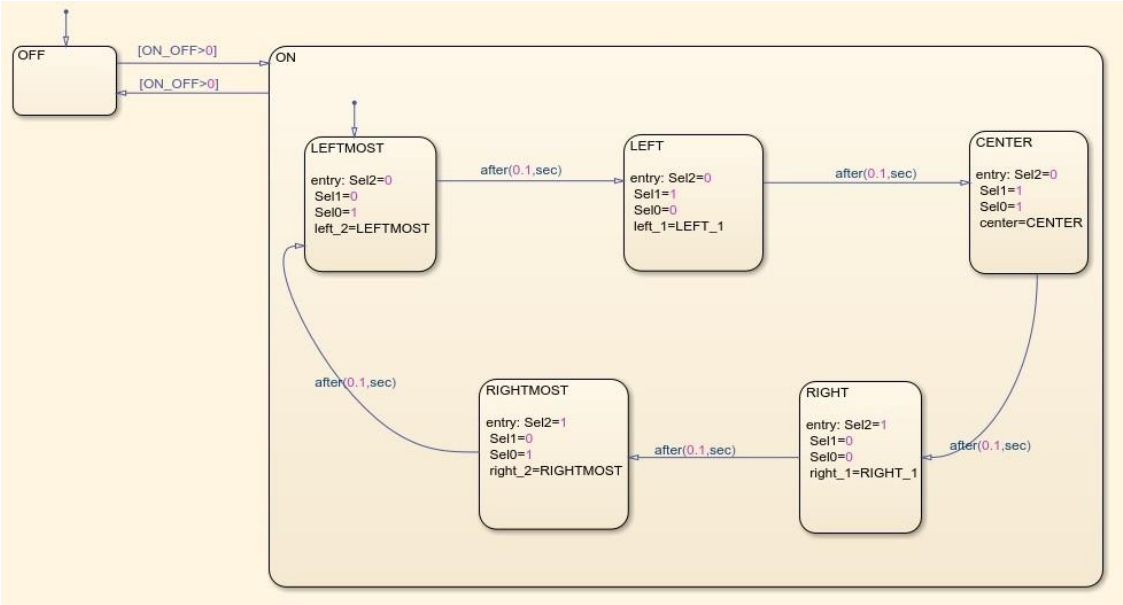
Figure 6: Line follower algorithm implemented in Simulink

1.2.3. Obstacle avoidance

If the robot needs to be autonomous, it is necessary to ensure a behavior free of collision. To satisfy this requirement, an obstacle algorithm needs to be implemented in the robot.

The robot needs to know if there are obstacles in its environment, the distance between the obstacles and itself and their positions. Among the sensors, which can be used to fulfill this task, we could find the US, IR, lasers and camera. Each sensor has some advantages and drawbacks in terms of precision, needs for data processing and price.

For the algorithm implemented in this report, the IR sensors are used. The robot has 5 IR posed in the front of the robot. The main state chart has five subsystem each one representing a sensor. The system switch from one subsystem to another one each 0.1sec. A filter is added to filter the disturbances due to the ambient light. The IR sensor detects an obstacle when the distance is less than 30 cm. The number of sensors determines the smoothness of the robot's movement.



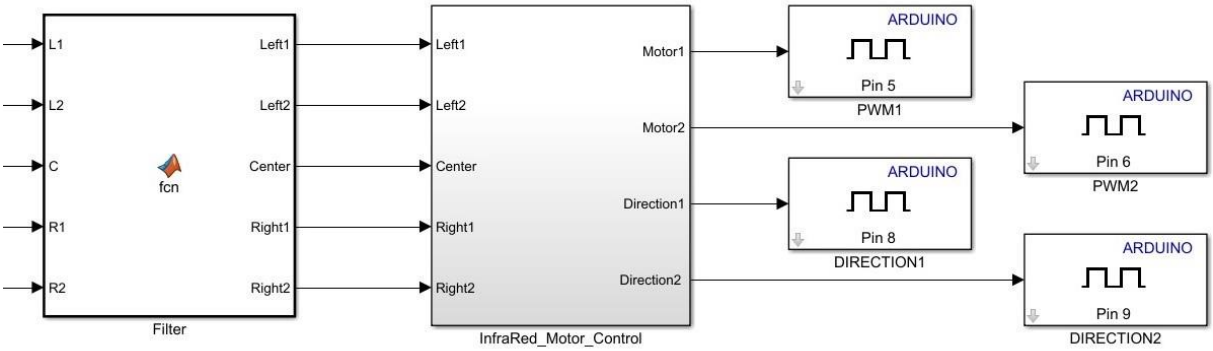


Figure 7: Obstacle avoidance algorithm implemented in Simulink

1.3. Code generation with MATLAB for LEGO NXT

1.3.1. System Requirements

- MATLAB 2016a or higher with Simulink
- LEGO Mindstorms NXT with Firmware version 1.28

1.3.2. Features

- PROFIBUS DP Slave
- Supported Baud rates:
  - 9.6 kbps
  - 19.2 kbps
  - 93.75 kbps
  - 187.5 kbps
  - 500 kbps
- Supports Sync mode

1.3.3. S-function Block Usage

The complete software stack is written in an S-function Block, which can be imported in a Simulink Model. S-functions (system-functions) provide a powerful mechanism for extending the capabilities of the Simulink environment. An S-function is a computer language description of a Simulink block written in MATLAB, C, C++, or Fortran. C, C++, and Fortran S-functions are compiled as MEX files using the mex utility (see Build MEX File). As with other MEX files, S-functions are dynamically linked subroutines that the MATLAB execution engine can automatically load and execute.

S-functions use a special calling syntax called the S-function API that enables you to interact with the Simulink engine. This interaction is very similar to the interaction that takes place between the engine and built-in Simulink blocks.

S-functions follow a general form and can accommodate continuous, discrete, and hybrid systems. By following a set of simple rules, you can implement an algorithm in an S-function and use the S-Function block to add it to a Simulink model. After you write your S-function and place its name in an S-Function block (available in the User-Defined Functions block library), you can customize the user interface using masking (see Block Masks).

If you have Simulink Coder, you can use S-functions with the software. You can also customize the code generated for S-functions by writing a Target Language Compiler (TLC) file. For more information, see S-Functions and Code Generation.

In Figure 8 is an example shown of the S-function Block.

Figure 8: S-function Block

The S-function for PROFIBUS has 2 Inputs and 2 Outputs.

1.3.4. Inputs

1. Ready for Data Exchange: a boolean that must be True for the Device to go into Data Exchange mode.
2. I/O Input Data as a multi-dimensional matrix.

1.3.5. Outputs

1. PROFIBUS DP Slave State: this output gives the current State of the PROFIBUS Slave Stack:
   - 0 = Power On: The Slave is powered on and the program is running and initialising)
   - 1 = Waiting for Parameters: initialisation is finished and the Slave now waits for parameterisation by a Master)
   - 2 = Waiting for Configuration: after a Master has parameterised the Slave, the Slave waits for the I/O Configuration.
   - 3 = Data Exchange: The Slave is fully configured and is sending and receiving its I/O data.

2. I/O Output Data as a multi-dimensional matrix.

When invoking a Level-1 MATLAB S-function, the Simulink engine always passes the standard block parameters, t, x, u, and flag, to the S-function as function arguments. The engine can pass additional block-specific parameters specified by the user to the S-function. The user specifies the parameters in the S-function parameters field of the S-Function Block Parameters dialog box (see Passing Parameters to S-Functions).

If the block dialog specifies additional parameters, the engine passes the parameters to the S-function as additional function arguments. The additional arguments follow the standard arguments in the S-function argument list in the order in which the corresponding parameters appear in the block dialog. You can use this block-specific S-function parameter capability to allow the same S-function to implement various processing options.

To configure the PROFIBUS S-Function Block right click the block and open 'Block Parameters (S-function)'. The Block has 3 additional parameters besides the standard t, x, u and flag block parameters:

- The PROFIBUS DP Slave Address (uint8)
- The number of Input Bytes (uint8)
- The number of Output Bytes (uint8)

An example of the properties window is shown in Figure 9 on the next page, the PROFIBUS DP Slave Address is set to 40 and we have 4 Input bytes and 4 Output Bytes. Changing these parameters doesn't affect the S-function blocks appearance.

To further connect the S-function block, we need to multiplex and demultiplex the In- and Output signals as shown in Figure 10 on the following page.

For the simplicity of the overall Simulink Model we include all this within a single Subsystem. A connected example of the Subsystem is shown in Figure 11.

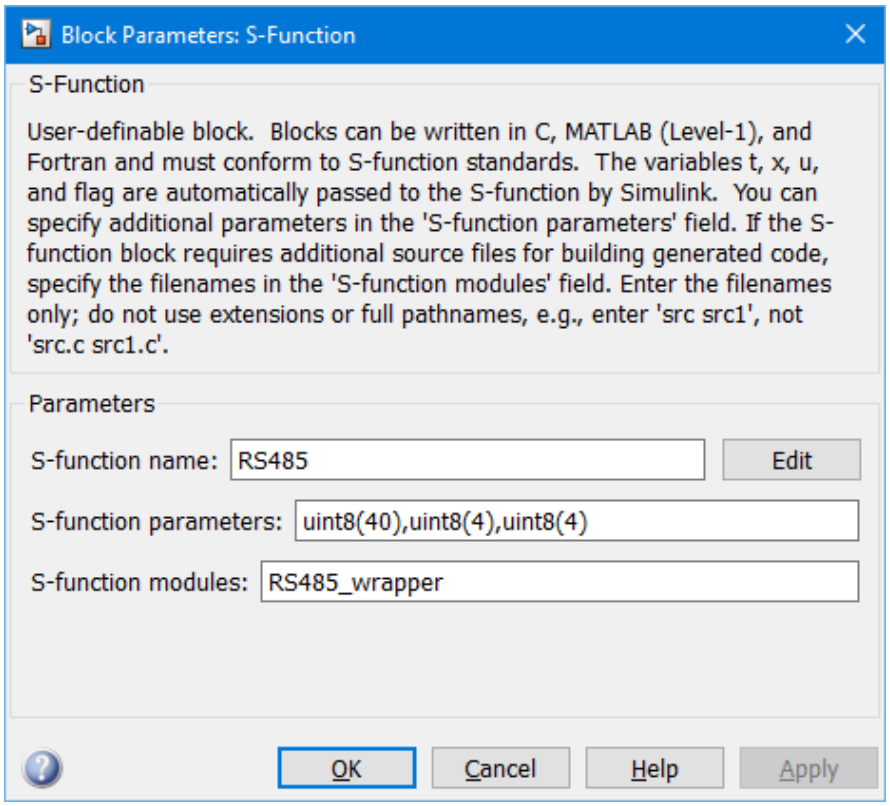


Figure 9: S-function - Block Parameters

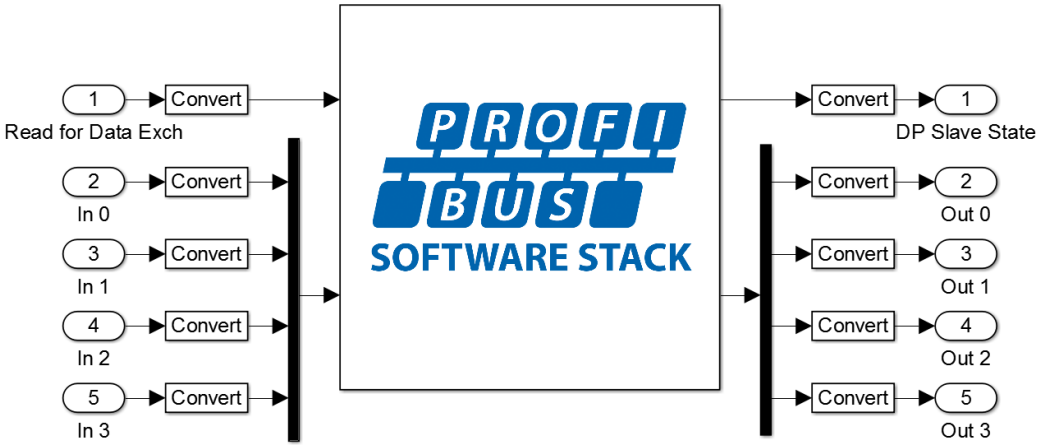


Figure 10: PROFIBUS - Software Stack with 4 Input Bytes and 4 Output Bytes

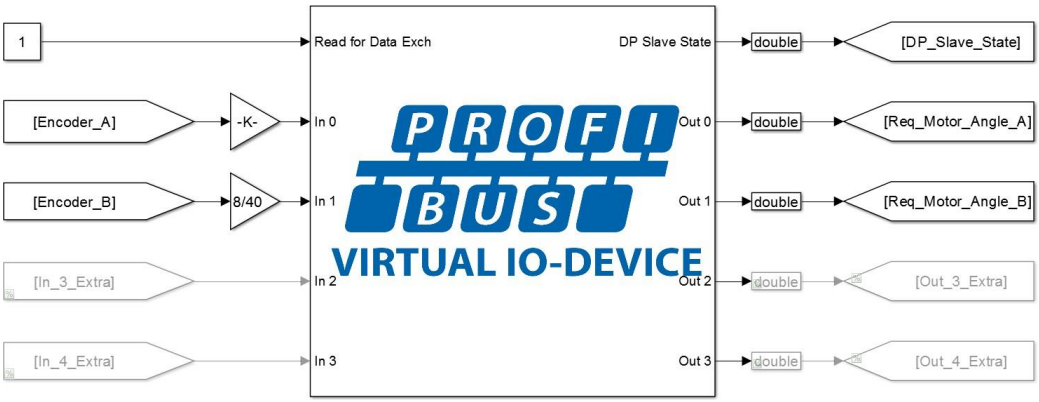


Figure 11: Subsystem connected via labels to other Simulink Blocks

2. Small robotic demonstrators

2.1. ISEN Robot controlled by Arduino

2.1.1. Description of the robot

The ISEN robot controlled by Arduino (see Figure 12) is a non-holonom robot using two driving wheels actuated by DC motors 6V, 240mA. It has 5 IR sensors, the TCRT5000, in charge of obstacle avoidance and 2 for line following function.

Some buttons, power off/on and for the joystick are mounted in the robot, to control it manually as well as to select the functioning mode of the robot. It has 3 LED matrices used to show the state of the robot as well as a LCD screen to write the error messages of the robot. An Arduino mega 2560, contains all the intelligence of the mobile base.

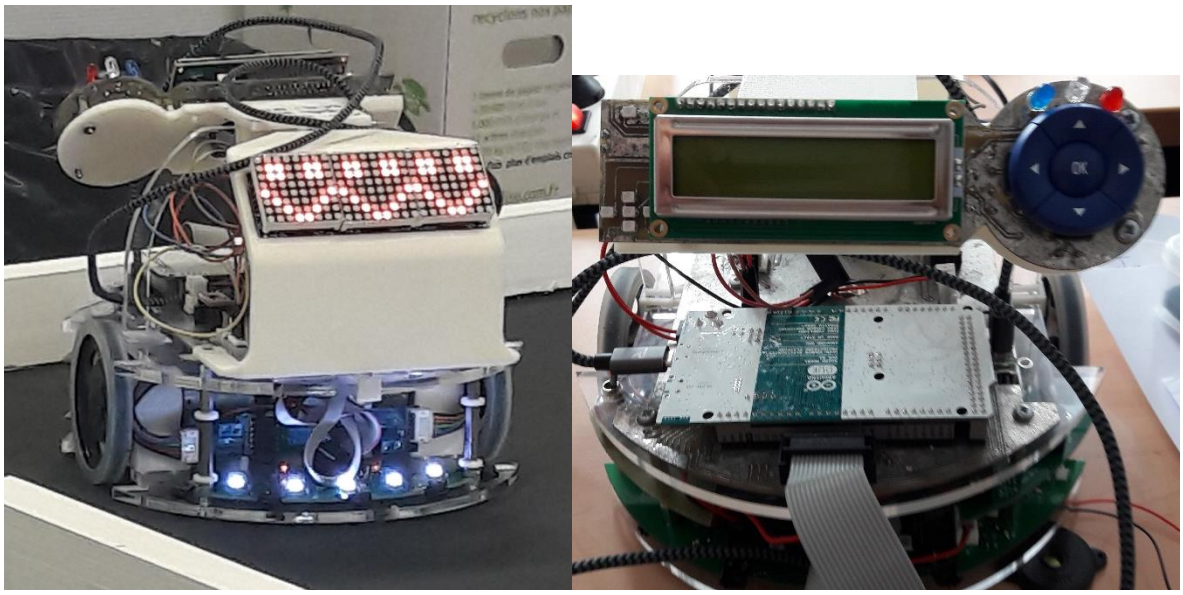


Figure 12: ISEN robot controlled by Arduino

The schematic of the robot is illustrated by Figure 13.

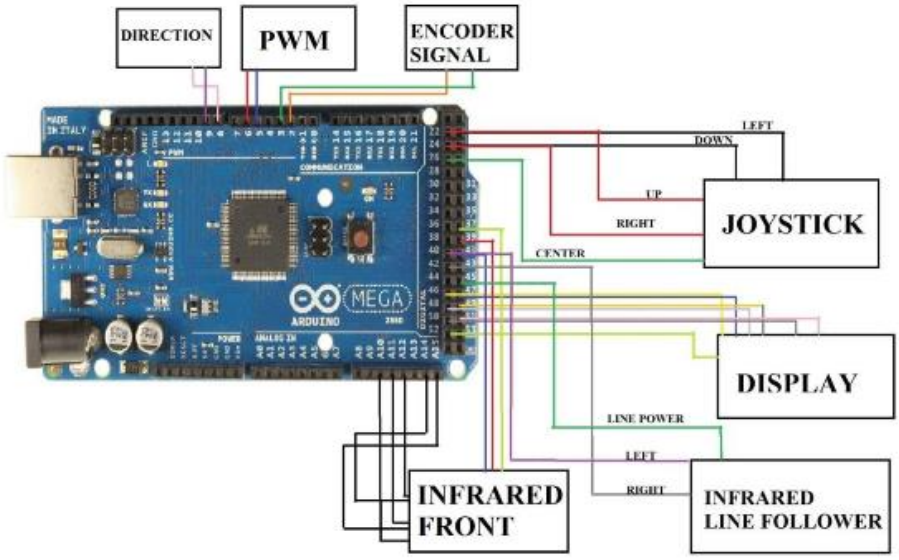


Figure 13: Schematic of the Arduino board connected the robot components

2.1.2. Rapid prototyping using MATLAB/Simulink

The algorithms developed in Simulink can be deployed into the target using the appropriate toolboxes provided by MATLAB/Simulink (see Figure 14).

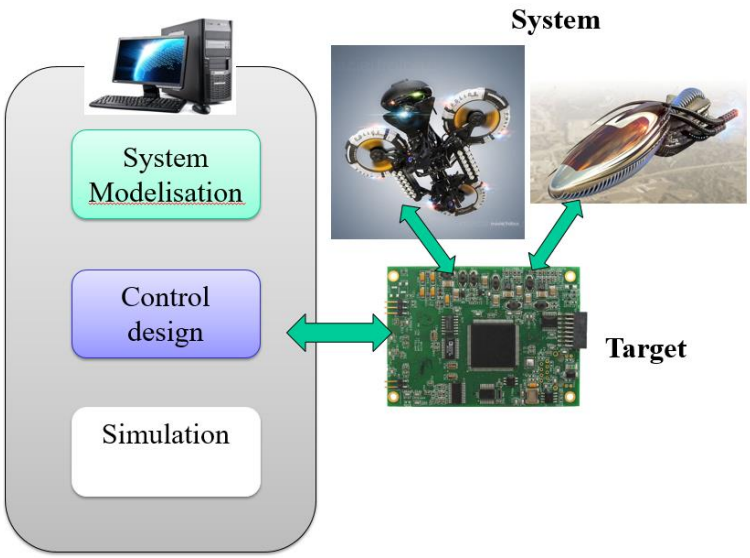


Figure 14: Rapid prototyping schematic using MATLAB/Simulink

The target can be an NXT (Lego Midstorm), an Arduino or a Raspberry Pi.

If the target is an Arduino, the Simulink Support Package for Arduino Hardware needs to be installed.

For that, in MATLAB Command Window, tape supportPackageInstaller and chose the suited hardware (see Figure 15).

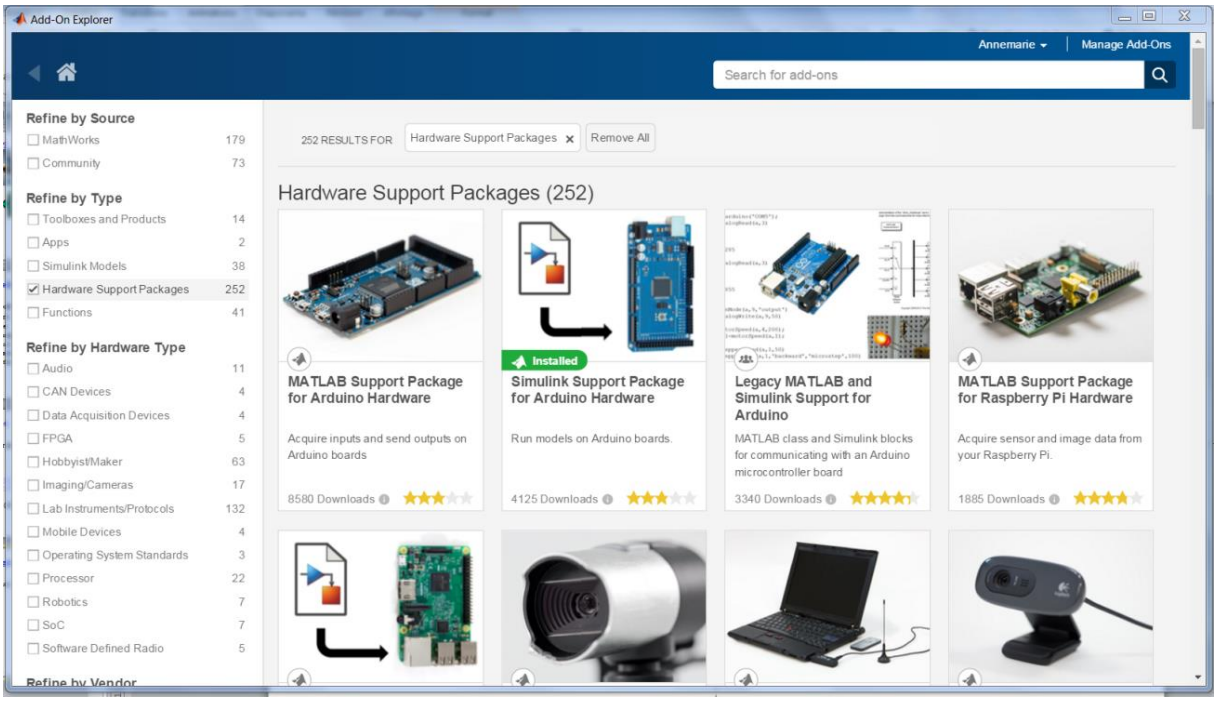


Figure 15: Hardware support packages for MATLAB/Simulink (version 2012b or more)

The Simulink Support Package is now available on Simulink Library (see Figure 16).

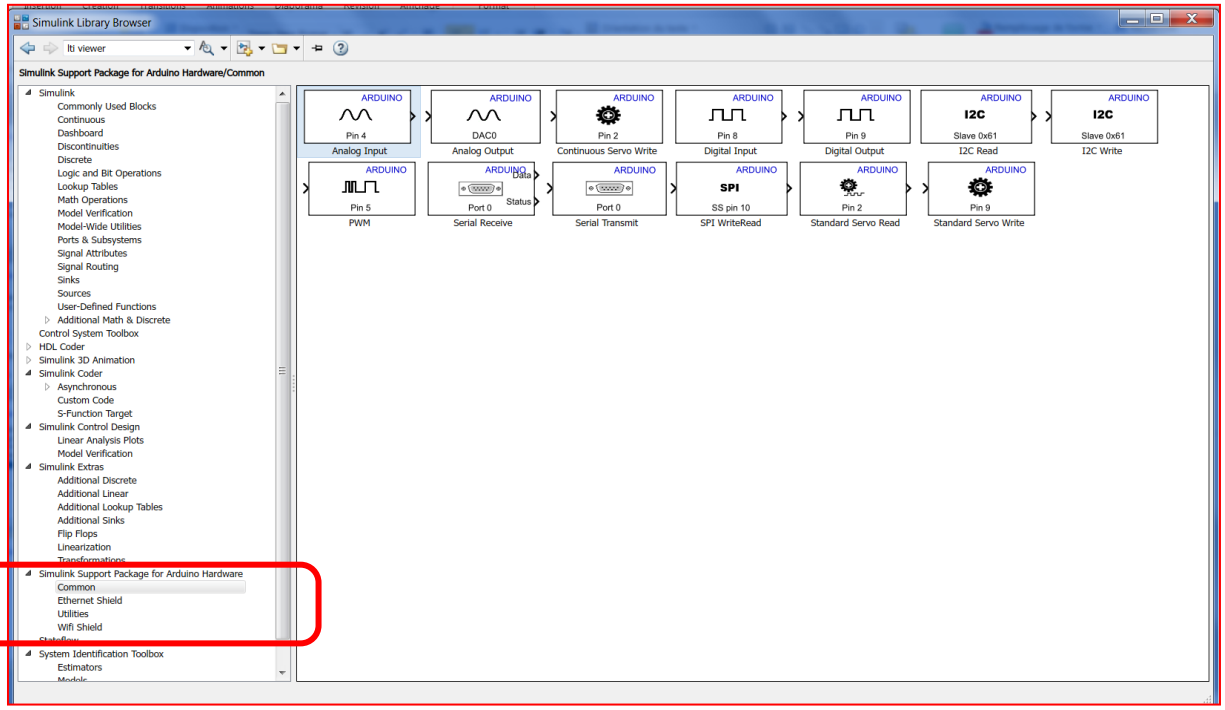


Figure 16: Simulink library browser with Simulink support packages for Arduino Hardware

Once the package for Arduino is installed and ready to use in Simulink, the Arduino board can be connected to the computer using the USB cable. If the computer cannot automatically detect the board driver, update the board driver manually (see *Figure 17*). To do that, Open Windows Device Manager and locate the board either in Other Devices or in COM Ports. Right-click on the board and select 'Update Driver Software...' Next, select 'Browse my computer for driver software'. Specify the search location as

C:\ProgramData\MATLAB\SupportPackages\R2016b\3P.instrset\arduinoide.instrset\arduino-1.6.7\drivers and click 'Next'.

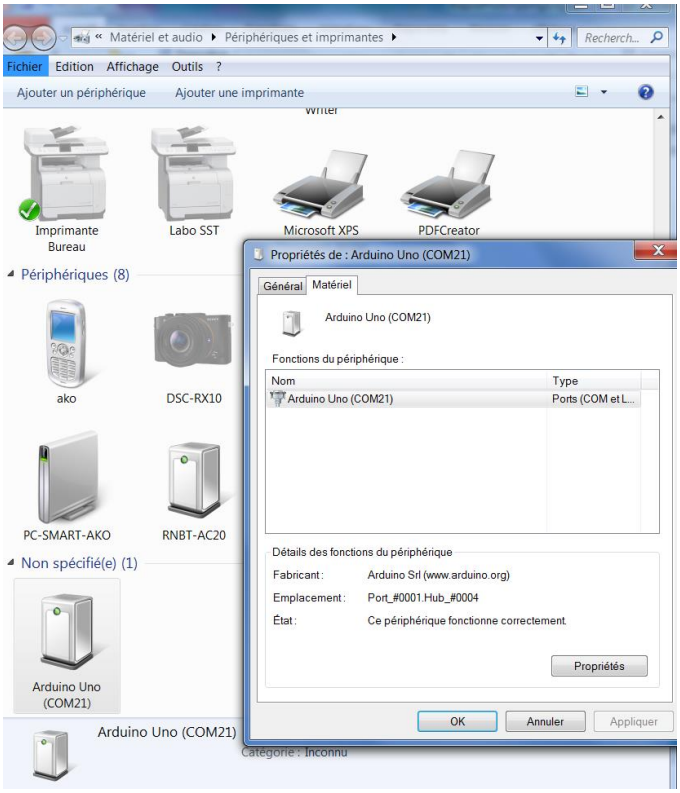


Figure 17: Manual update of the Arduino board driver

Once the Simulink algorithm is ready to be deployed in the target, Open the Configuration parameters window from ***Tools → Run on Target Hardware → Options*** and select the Arduino board model (for example Arduino Due, see *Figure 18*).

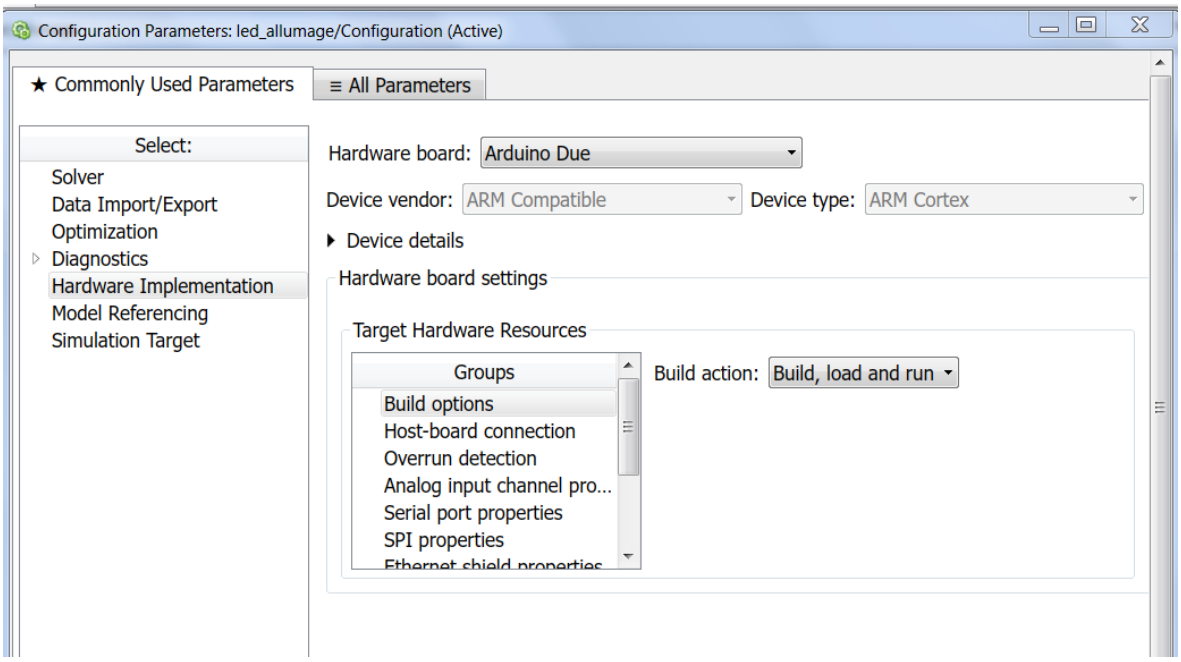


Figure 18: Choice of the hardware Arduino board model

The Simulink algorithm is now ready to be deployed on the target.

2.1.3. Robot control algorithms

All the algorithms described in the previous chapter are implemented into the robot thanks to Rapid Prototyping using MATLAB/Simulink. The robot could also be controlled by a joystick using the buttons mounted in front of it (see Figure 12). Using the left or right buttons, it is possible to choose among the 4 demonstration algorithms: obstacle avoidance, line follower, geometric shape (Straight Line over a requested distance of travel, triangle, circle, square) or a joystick. The central button is used to select one of the program. When the button is pressed, the algorithm enters in the selected function and executes it.

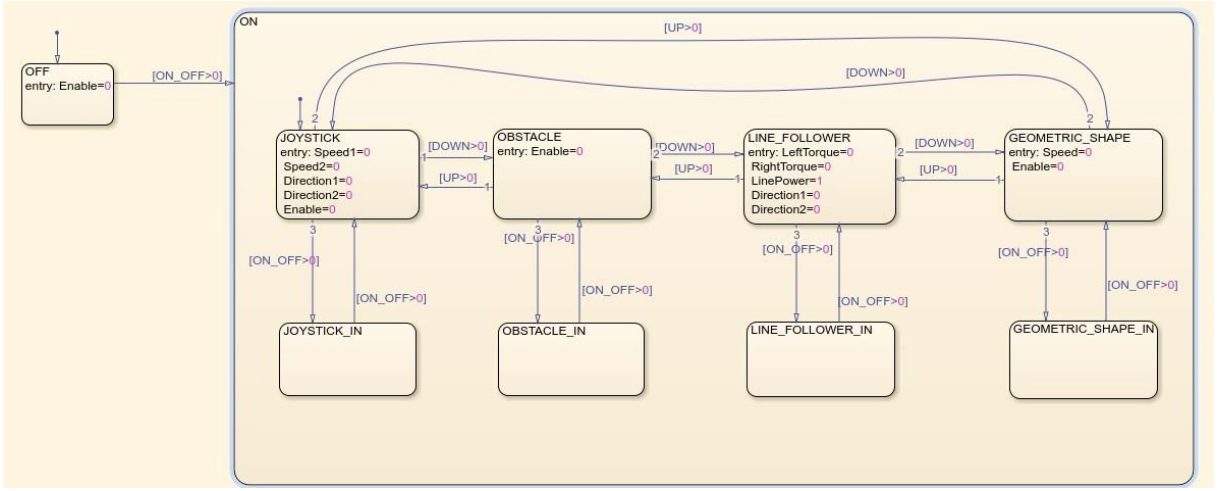


Figure 19: Overall robot demonstration algorithm implemented in Simulink

2.2. ISEN Robot controlled by Raspberry Pi3

2.2.1. Description of the robot

The robot using a Raspberry Pi3 uses the same motors and sensors than the one using the Arduino.

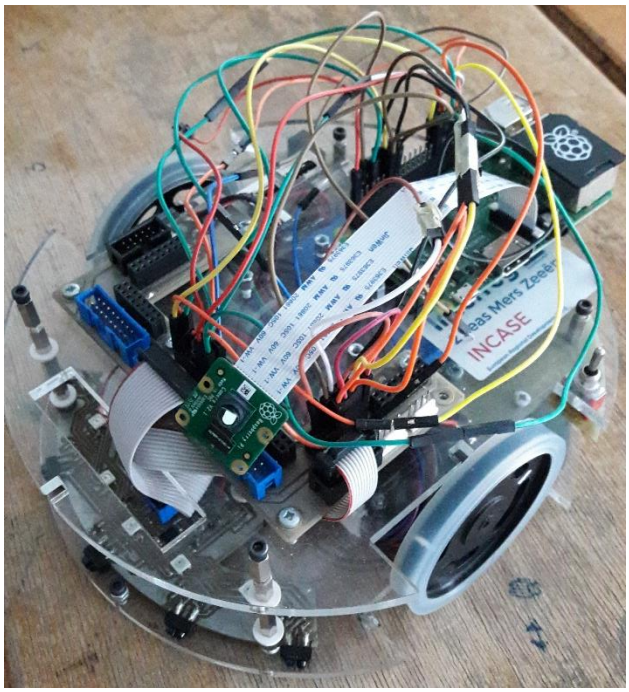


Figure 20: ISEN robot controlled by Arduino

In the following figures, the general view of pinout of the Raspberry Pi 3 is described.

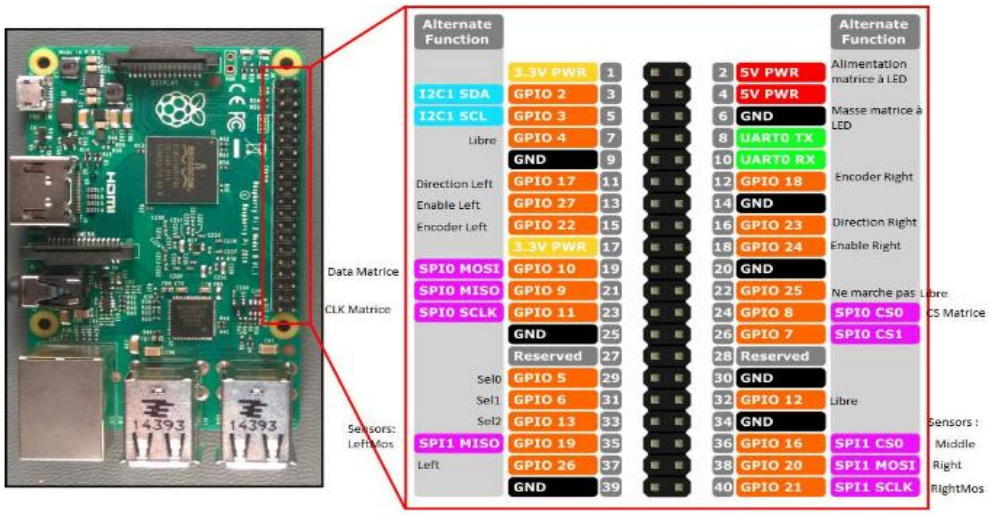


Figure 21: Raspberry Pi pinout

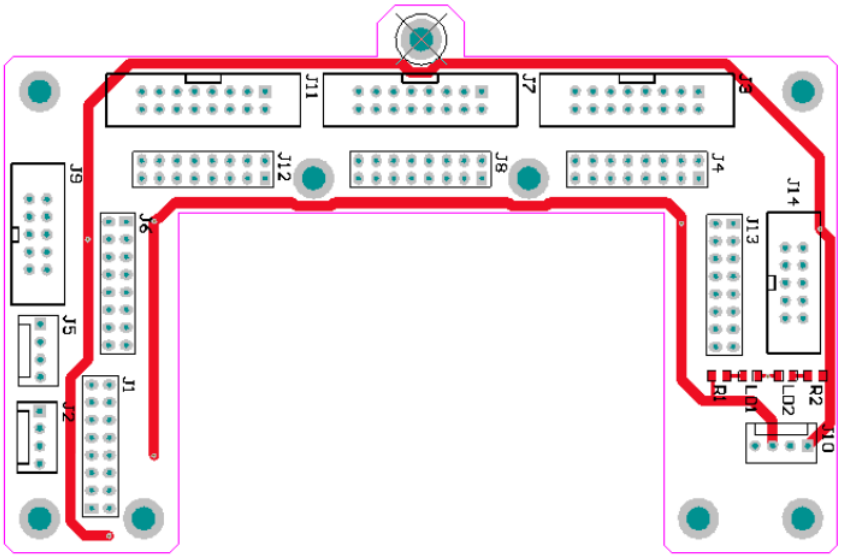


Figure 22: Connectors of the mobile platform

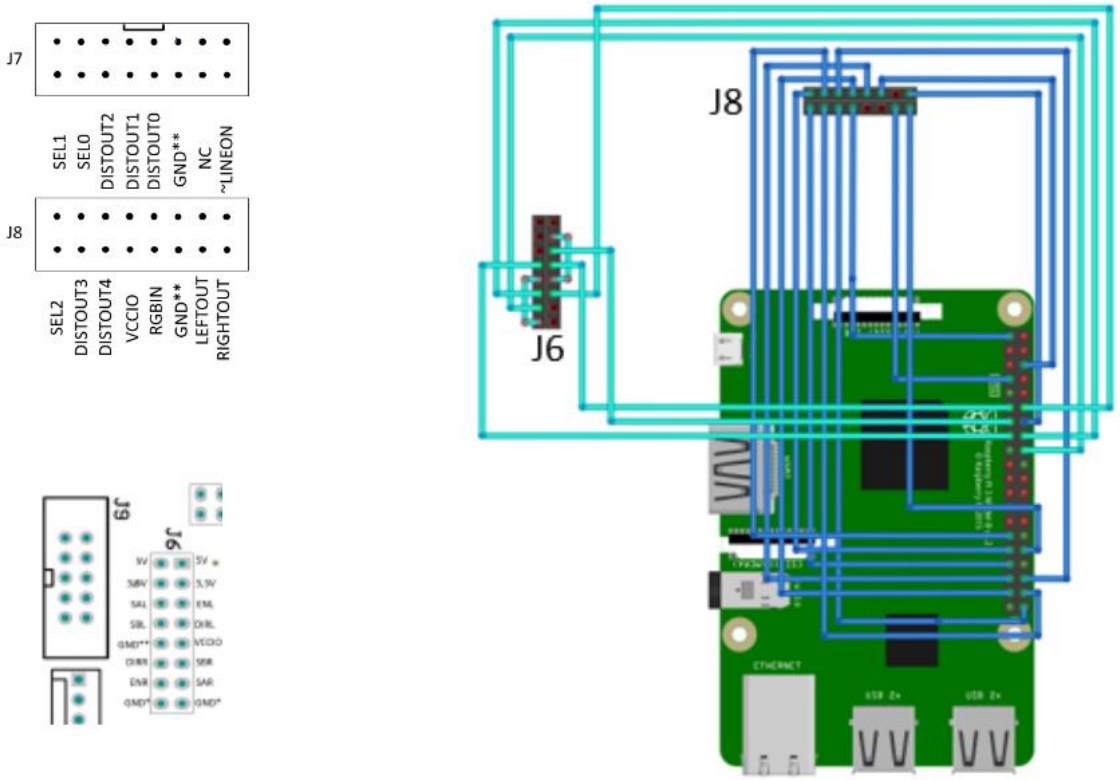
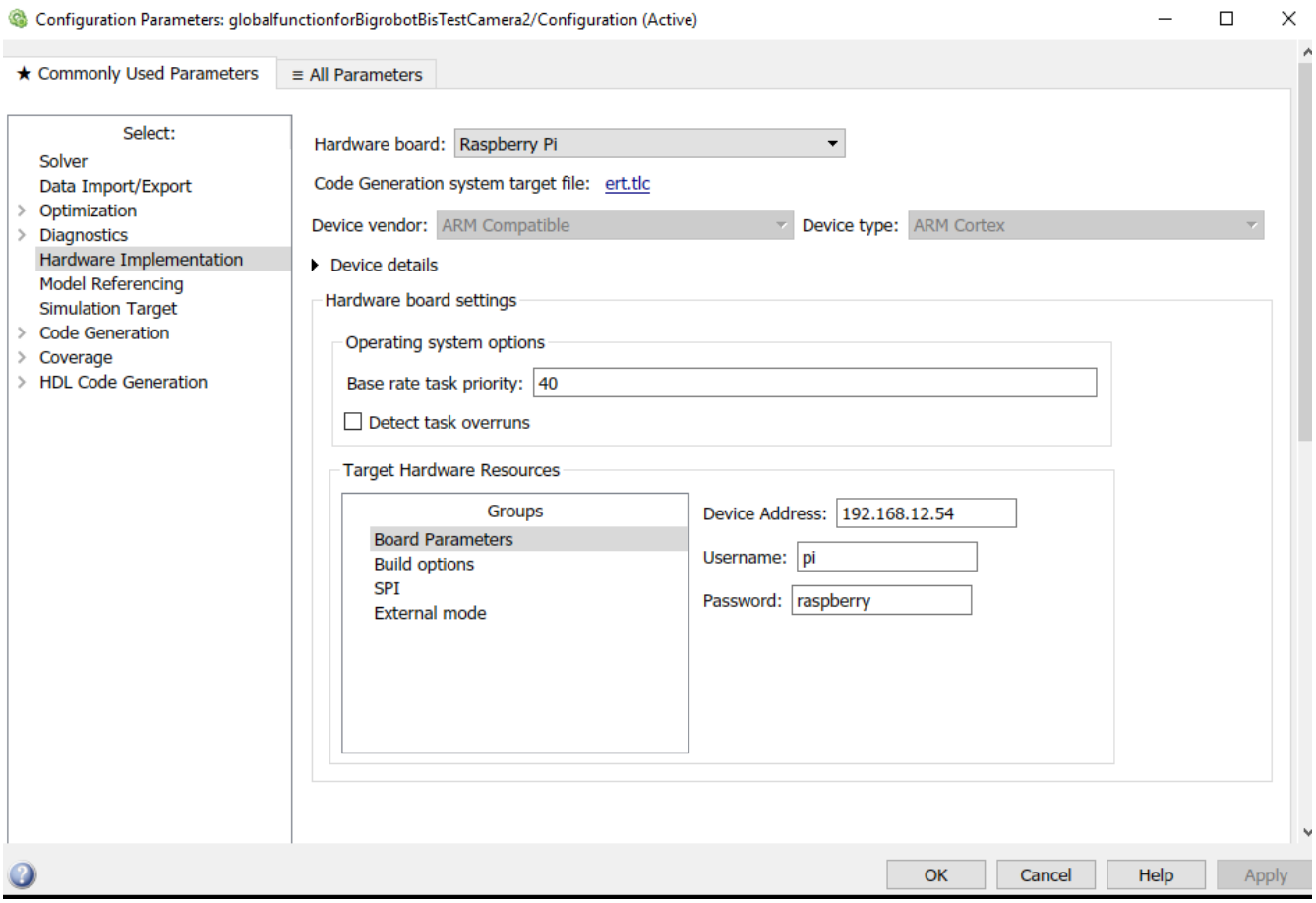


Figure 23: Connections between the Raspberry Pi and the mobile platform connectors

2.2.2. Rapid prototyping using MATLAB/Simulink

Once the appropriate Simulink Support Package for Raspberry is installed, to be able to run Simulink models we need to set some final configurations in Simulink settings. As the figures below shows, the hardware target, the IP address and the raspberry pi login & password need to be set.



2.2.3. Robot control algorithms

All the algorithms used for the Arduino board are also deployed in the Raspberry Pi3 target.

The main algorithm controls the robot. Using the Joystick, one of the following functions can be sellected:

Left : “ObstacleAvoidance” is the function which use the infrared sensor at the front of the robot to detect and avoid obstacle. It changes direction when he detects one obstacle.

Down: “Joystick” is the function which use the joystick to control the robot (Physical or software). It make the robot go straight forward or back with Up/down.

Right: “GeometricalShape” is the function which draw geometrical shape with the robot, you select the shape while inside by using the joystick:

· Left: Square

· Down: Circle

· Right: Straight Line

· Up: Triangle

Up: “LineFollower” is the function which use the ground infrared sensors to detect a line and follow it. You have to put the robot on a black line on a white surface to make it work well, since we have only digital inputs.

Middle: The button used to go back to the principal menu, and to stop the actual function.

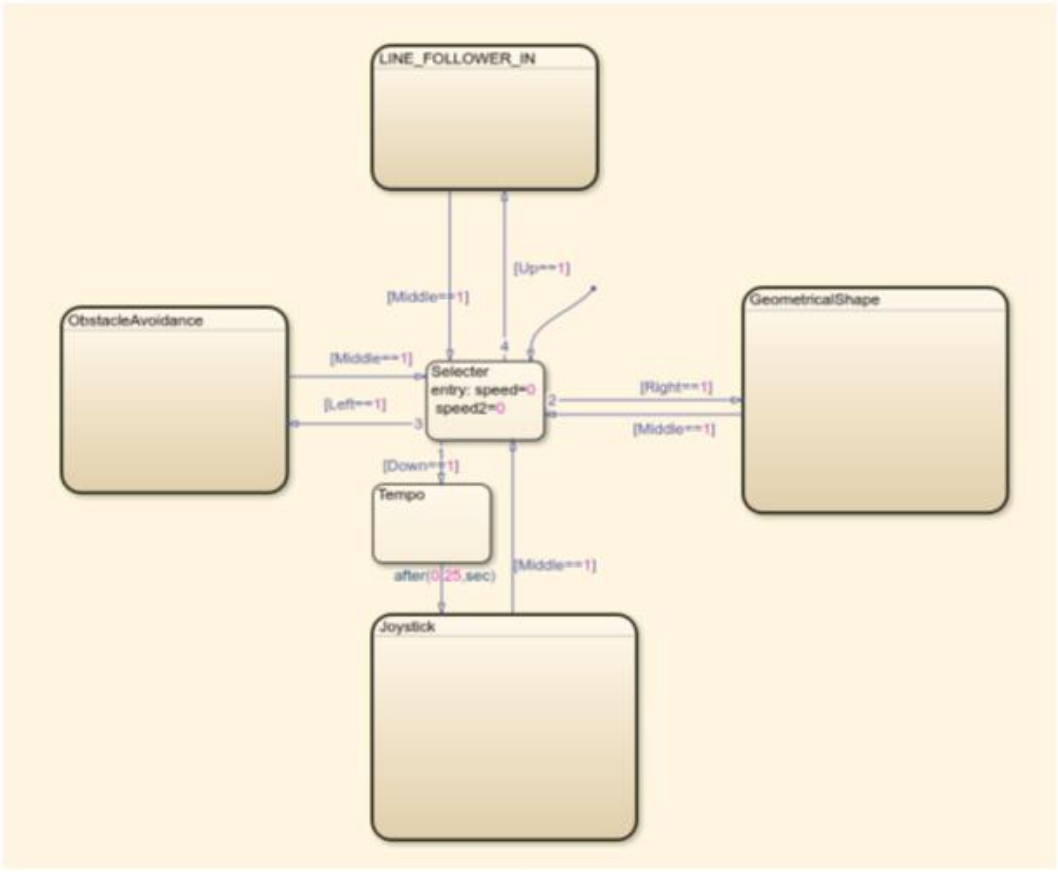


Figure 24: Overall robot demonstration algorithm for Raspberry Pi 3 target implemented in Simulink

2.3. Gantry Crane Description

2.3.1. Introduction

The purpose of the gantry crane is to load and unload supplies for the warehouse of a water production plant. The crane is built using LEGO, LEGO Mindstorms NXT CPUs running code generated from MATLAB/Simulink, and a Siemens PLC[1].

[1] PLC: Programmable Logic Controller

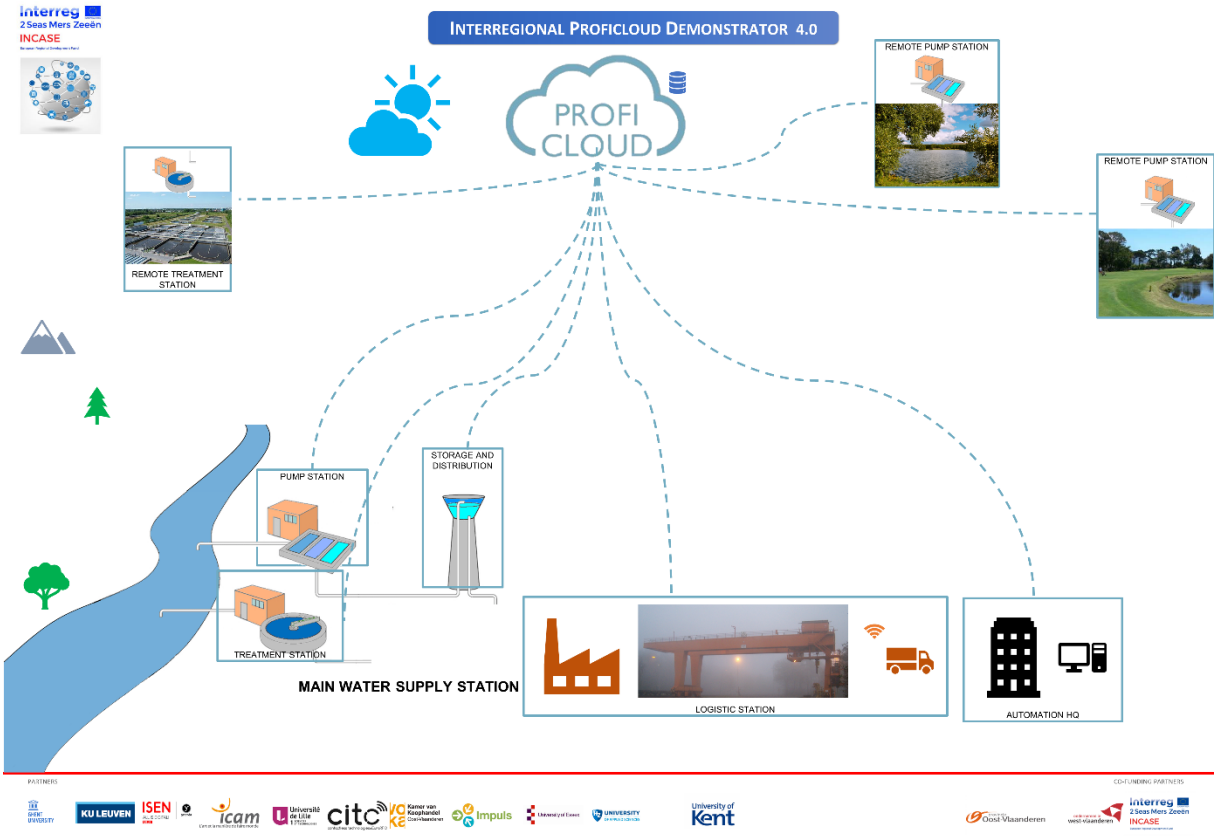


Figure 25: Interregional PROFIcloud Demonstrator

2.3.2. Physical construction of the gantry crane

The gantry crane is built on top of a metal frame that functions as a rail. In the picture below, you can see the design of the gantry crane.

Trucks with a payload can drive to one of the two truck positions. When a truck is detected, the crane moves to that position and automatically picks up the payload. The payload is then placed in the stockroom on the appropriate position.

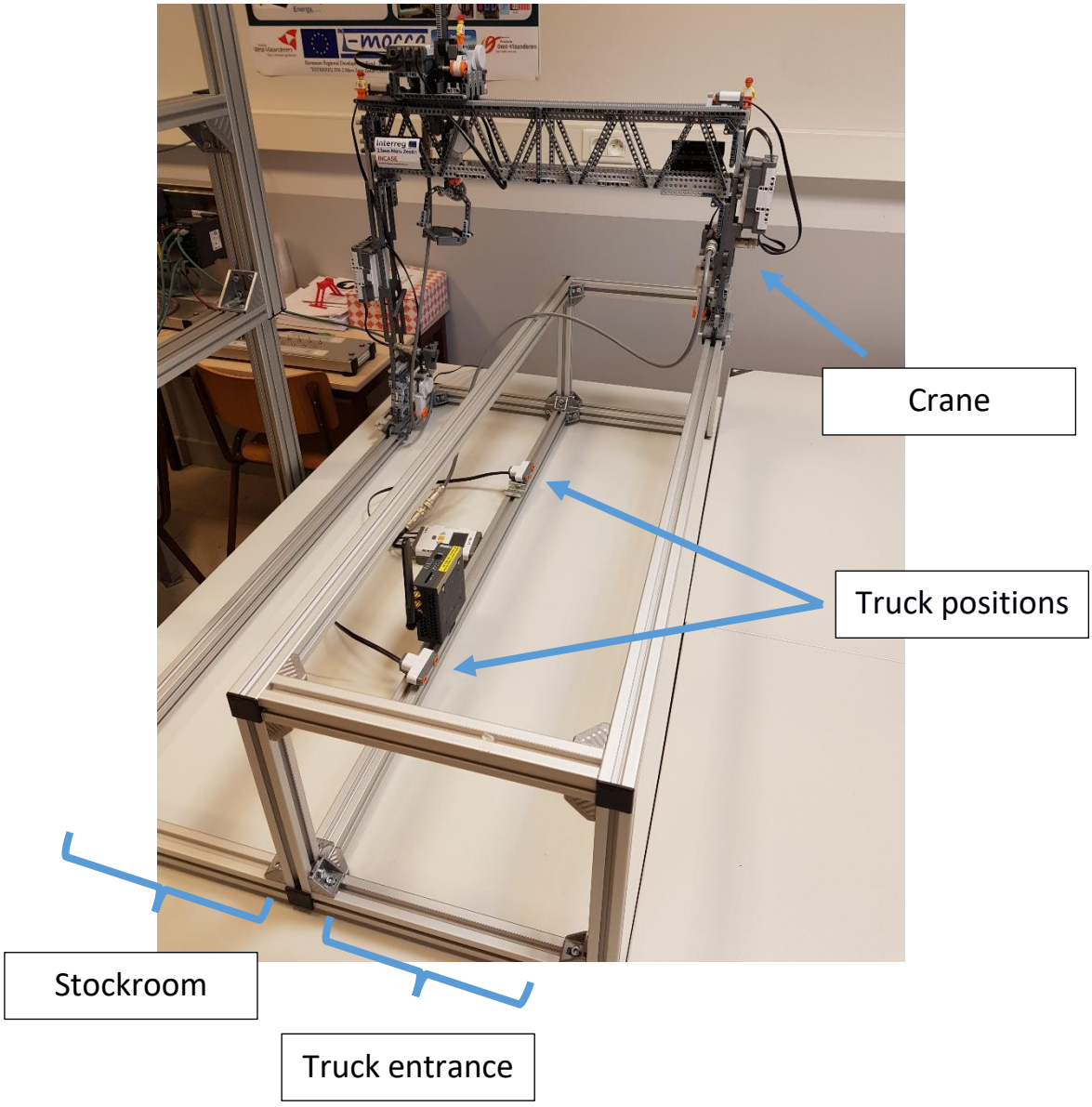


Figure 26: The gantry crane on top of the metal frame

2.3.3. LEGO Mindstorms NXT

NXT Brick

Figure 27: LEGO Mindstorms NXT brick

This is the unit that controls the motors and reads the necessary sensors. The NXT has the ability to control 3 NXT motors. On the same side is the USB type B port with which the NXT can be programmed. This USB connection can take up to 12 Mb/s and the NXT also has a Bluetooth module.

The NXT has 4 input ports on which you can connect a range of sensors. The connection is made via a custom RJ12 connector. Furthermore, the NXT has a speaker, a 100 x 64 pixel LCD screen where up to 8 different lines of data can be displayed.

Processor	Atmel 32-Bit ARM AT91SAM7S256 48 MHz 256 KB FLASH-RAM 64 KB RAM
Co-Processor	Atmel 8-Bit AVR, ATmega48 8 MHz 4 KB FLASH-RAM 512 Byte RAM
Operating System	Proprietary
Sensor ports	4, Analog Digital: 9600 bit/s (IIC)
Motor ports	3, with encoders
USB Communication	Full speed (12 Mbit/s)
USB Host	n/a
SD-Card	n/a
Communication with Smart devices	Android
User-Interface	4 Buttons

Display	LCD Matrix, monochrome 100 x 64 Pixel
Communication	Bluetooth USB 2.0

Table 1: LEGO NXT Specifications

The NXT is not programmed via LEGO's own supplied software, Simulink Support Package for LEGO MINDSTORMS NXT Hardware (MathWorks, sd) has been used here. This toolbox allows programming of the NXT by means of block diagrams in Simulink.

The 4th input port also allows to communicate via RS485. Via a software implementation of the PROFIBUS DP stack, developed at KU Leuven, it is possible to receive and send PROFIBUS DP messages.

NXT motor

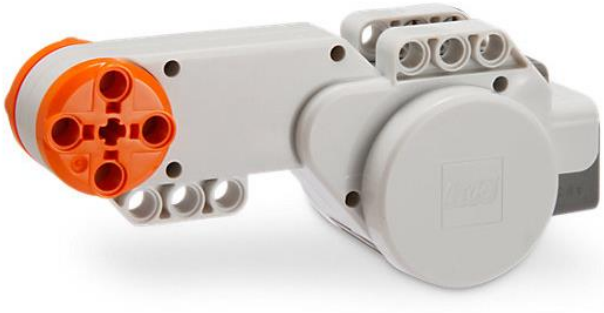


Figure 28: LEGO NXT motor

This is a DC motor that is fed from the NXT by a differential pulse width modulation voltage. This together with 2 encoder signals, ground and 4.3 V power supply are in one cable. The theoretical speed is up to 170 rpm. The incremental encoder has a resolution of 1°. This data can be accessed via a separate block in Simulink. Here you can choose to continue counting, with each sample time or by an external signal reset. The sample time is also adjustable. This time must be long enough to count enough pulses over this period at sufficiently low speeds.

To test the motor, a step from 0% to 100% is applied to the motor voltage. As shown below, a delay of 4 ms is measured.

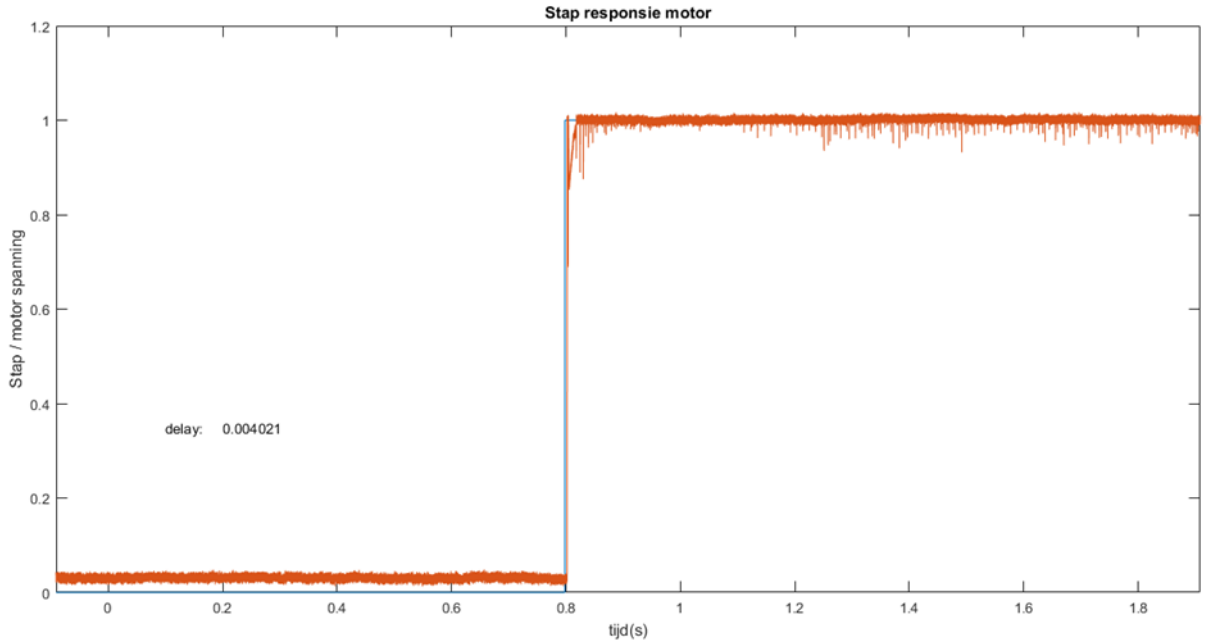


Figure 29: Step response of the motor voltage

The motors have incremental encoders, so the speed is measured with pulses. Through convolution the pulses are converted into the speed to measure a discrete step response for the encoders. The convolution was executed with a window of 14000 samples, which was calculated by the following formula: $\frac{Period\ 1st\ pulse}{Sample\ interval} * 4 = \frac{7*10^{-3}\ s}{2*10^{-6}\ s} * 4 = 14000$. This will shift a frame of 14000 samples over time. This frame size ensures that when starting 3 pulses are seen and at full speeds 9 pulses. The measured delay is 10 ms.

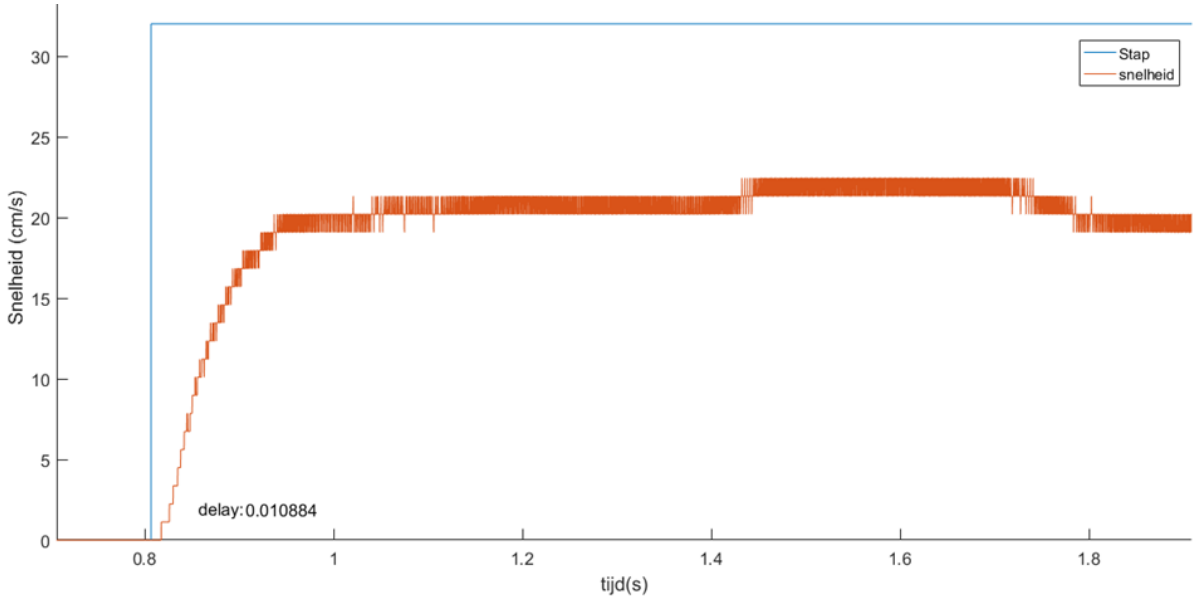
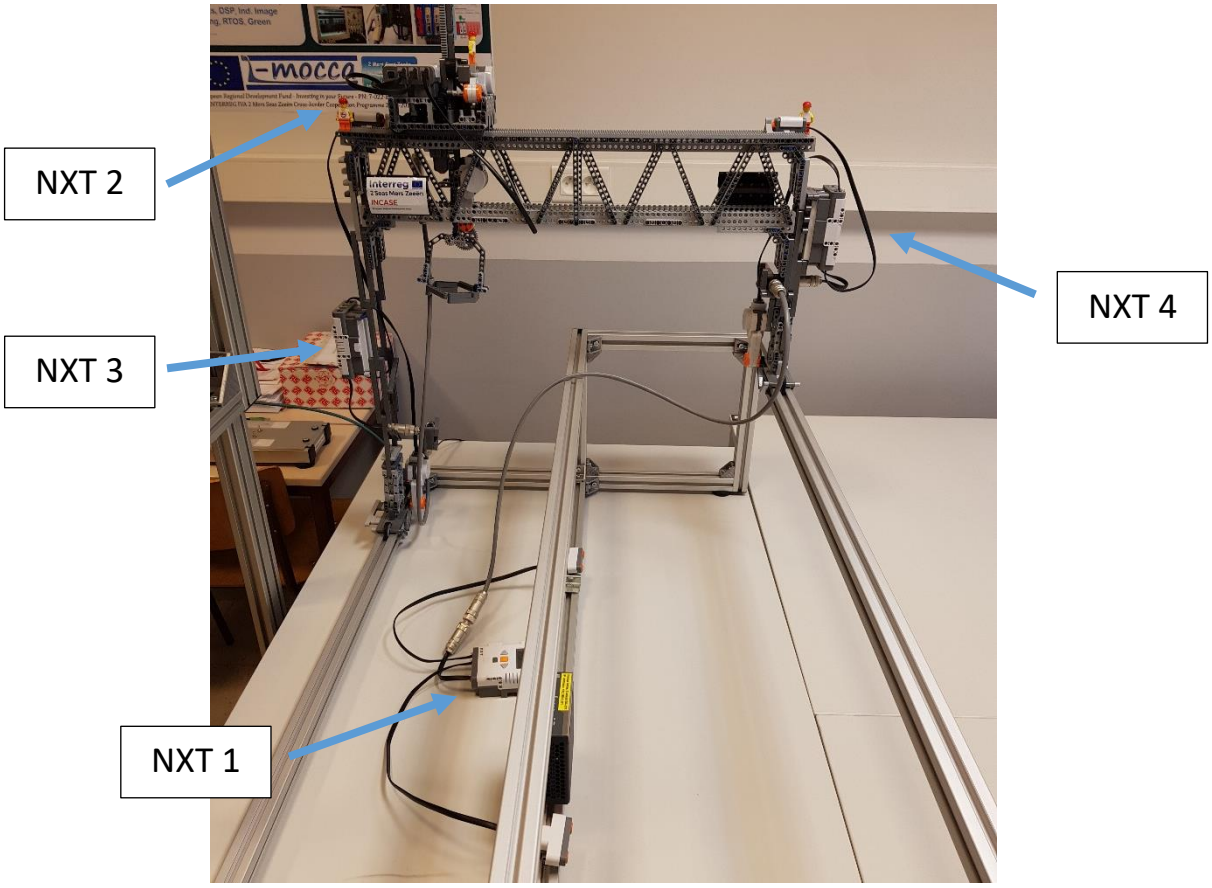


Figure 30: Step response of the incremental encoders

Use of LEGO Mindstorms NXT

There are four NXT bricks built into the gantry crane, each one has its own purposes. The location and functions of each NXT are displayed in the schematic below.



NXT	Function	Sensors and motors
1	Detects the trucks End of rail detection	Two ultrasonic sensors and one limit switch
2	Moves the arm, the cabby Controls the gripper	Three motors and one limit switch
3	Moves the left leg End of bridge detection	One motor and one limit switch
4	Moves the left leg End of bridge detection	One motor and one limit switch

Figure 31 and Table 2: a schematic overview of the functions of each NXT.

2.3.4. Automation of the gantry crane

Controller

The controller used for the automation of the gantry crane is a Siemens PLC, more specific, a CPU 315F-2 PN/DP.

Figure 32: Siemens CPU 315F-2 PN/DP

The PLC program controls the crane. In manual mode, the crane is controlled by switches connected to the digital inputs of the PLC. The automatic mode still needs to be designed, but the idea is that the crane automatically loads and unloads trucks when they are detected.

Communication between the PLC and the NXT bricks

The PLC communicates with the NXT bricks using PROFIBUS. The NXT bricks have a RS485 compatible port and thanks to a software implementation of the PROFIBUS DP stack, the NXT bricks are able to send and receive PROFIBUS DP messages. The NXT bricks can be added as a device in the PLC programming software using a GSD-file. Once added to the device, the NXT bricks can be used as a PROFIBUS DP Slave.

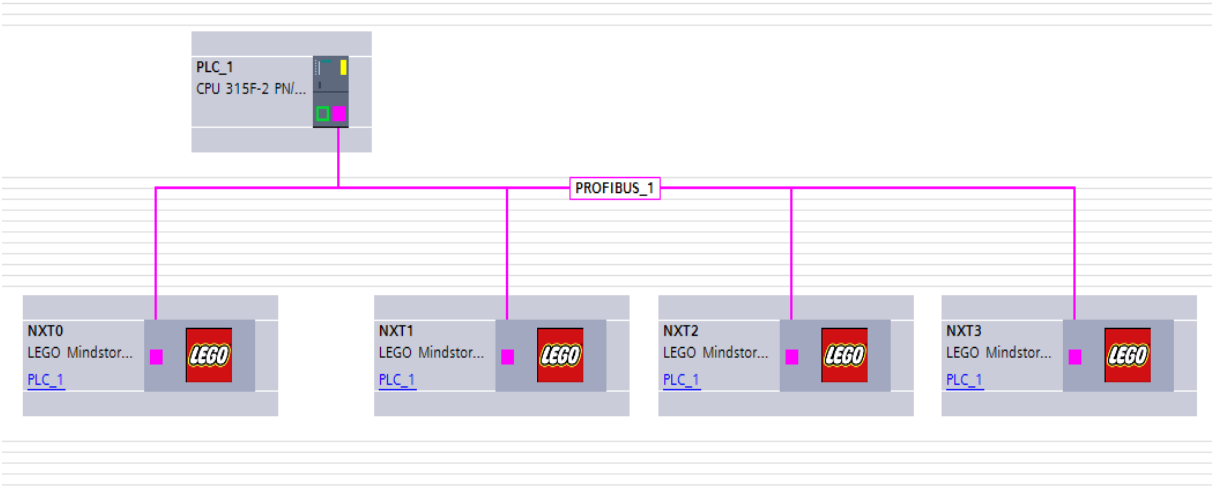


Figure 33: PROFIBUS connection between the PLC (Figure 32) and the NXT bricks

NXT Program

Since the NXT bricks are added to the PLC program as PROFIBUS DP Slaves, the logic to control the motors and read in the sensors must be added to NXT bricks themselves. They are programmed using code generation by MATLAB/Simulink. This is possible by using the "Simulink Support Package for LEGO MINDSTORMS NXT Hardware".

Simulink must be configured to be able to generate the code and deploy it on the NXTs. The settings are available through "Tools" → "Run on target hardware" → "Options".

Settings:

- Hardware board: LEGO MINDSTORMS NXT
- Connection type: USB connection

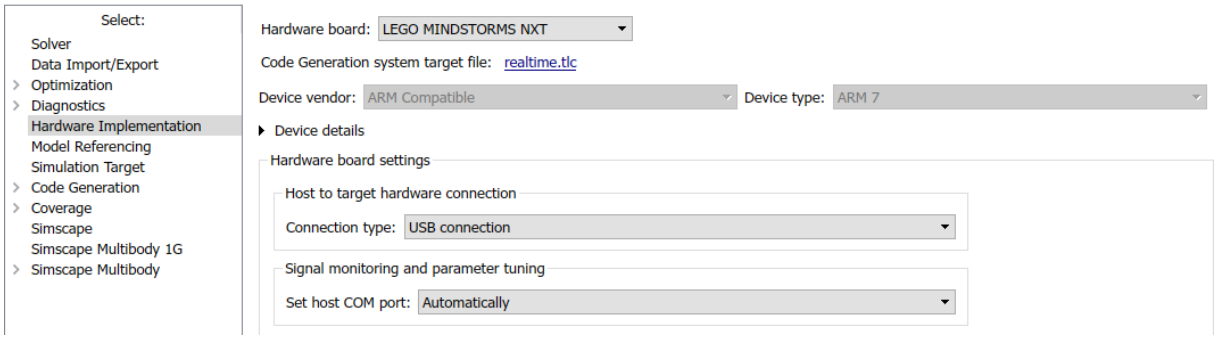


Figure 10: Simulink settings for programming the NXTs

To generate and deploy the code to the NXTs, "External" must be selected in "Real-Time Simulation" (1). When "Deploy to hardware" (2) is clicked, the program is sent to the NXTs.

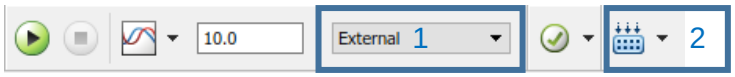


Figure 34: Deploying Simulink to the NXTs

When the program is called for the first time every actuator value and internal value are put at 0. Just after the crane initializes itself: the arm on the top, the mobile part on the right part of the bridge and the crane at the beginning of the rail. It initializes on the same internal variable required for the software part. After that the PLC waits for the first truck detection to begin to move. Each movement of the crane is calculated with the encoder values and implemented to never touch the limit switch.

A speed controller is used to be able to regulate the speed. An acceleration limitation is used to have smooth velocity changes, so the crane can stay balanced. A saturation limits the voltage applied to the motor to don't apply more voltage than the brick can apply on the motor. And a security is implemented to apply 0 on the motor when the brick is not connected to the PLC.

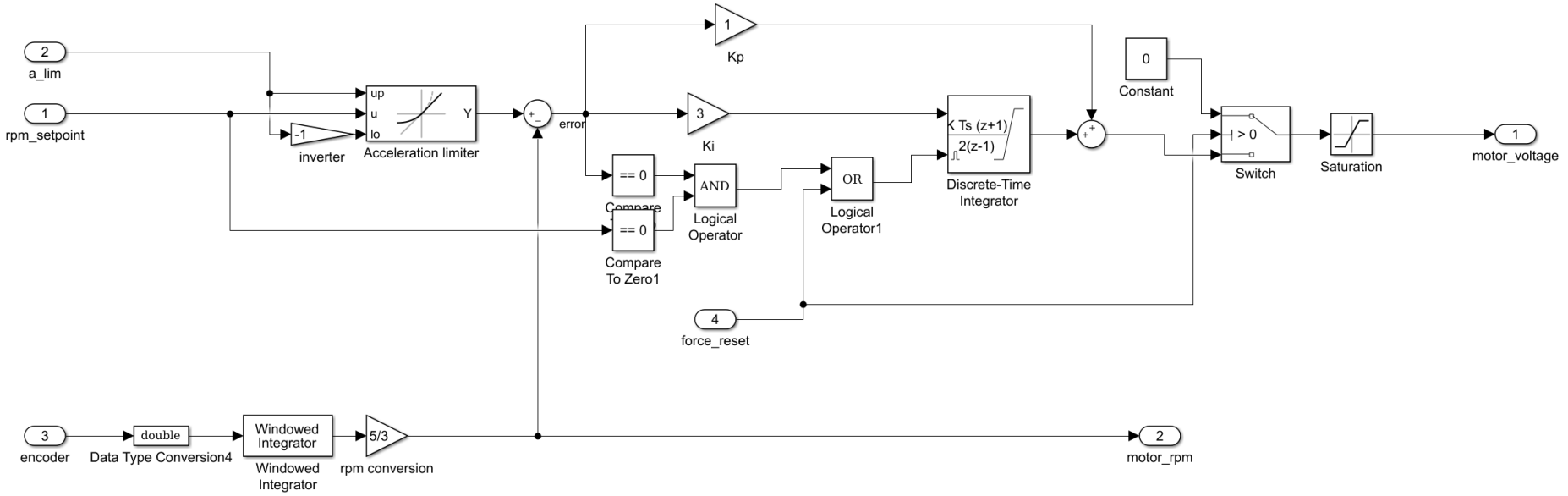


Figure 35: Screenshot of the speed controller

A position controller is also used to determine the position of the crane and deduce the speed required to finish the move. As before a saturation is used to limit the speed and don't have too much voltage applied on the motor. The speed delivering is proportional to the distance between the reference position and the actual one.

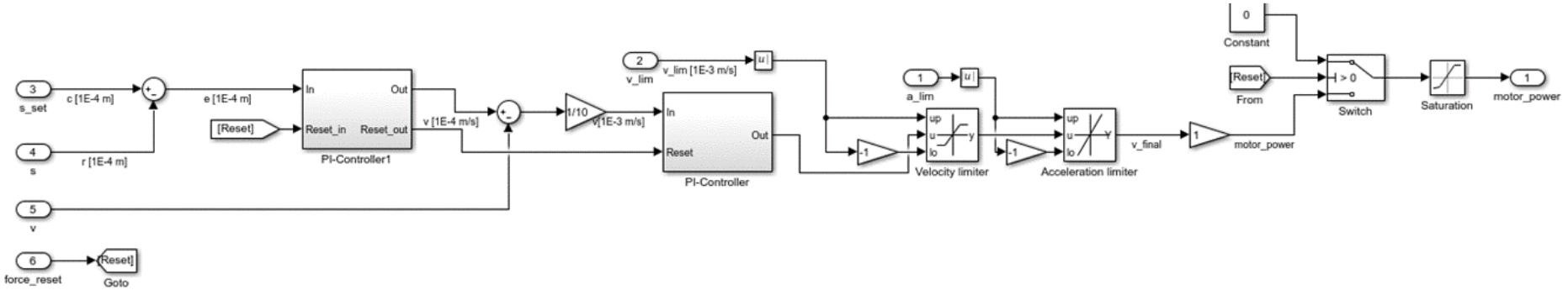


Figure 36: Screenshot of the position controller

The estimation of the actual distance is done with the value of the encoder and a gain.

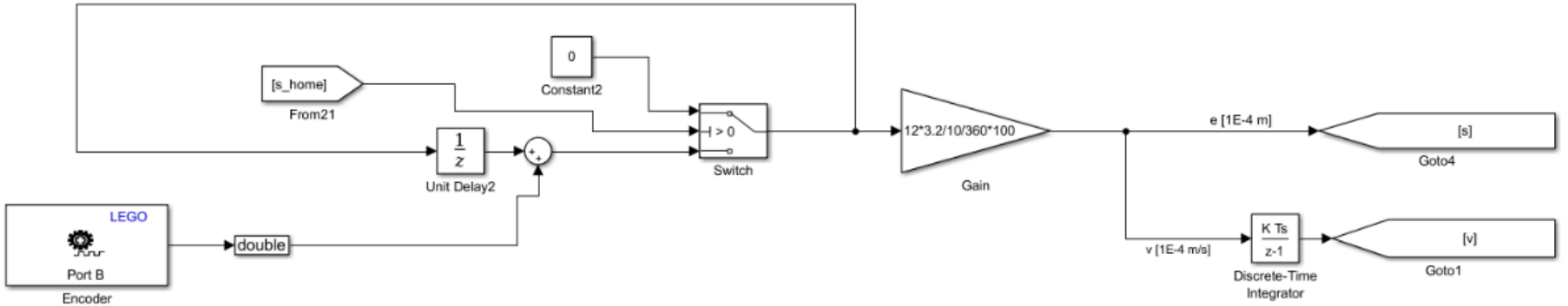


Figure 37: Calculation of the position

Aside from controlling the motors, the sensor values are also read by the NXT. The sensor values are used in the NXT to control the motors, but are also sent to the PLC.

Like explained earlier, the PROFIBUS DP Stack is used inside the NXT bricks, so it is able to send and receive PROFIBUS DP messages.

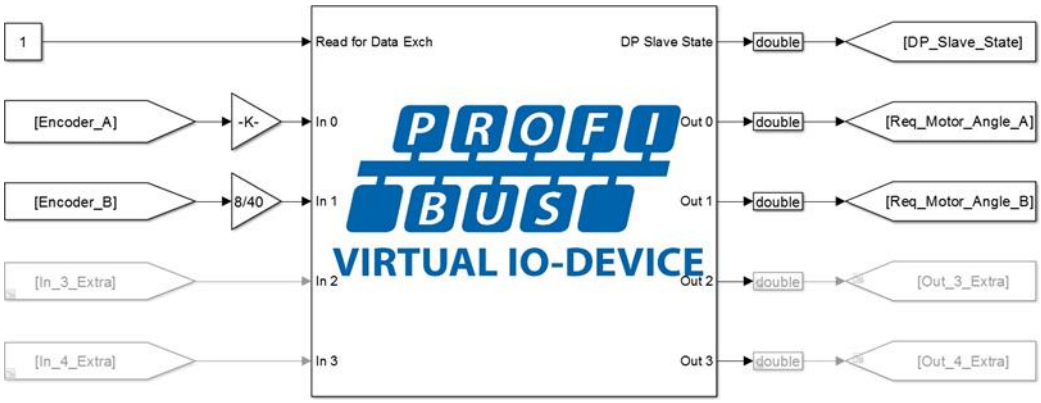


Figure 38: Subsystem connected via labels to other Simulink Blocks

2.3.5. Future work

The gantry crane will be part of the large PROFIcloud demonstrator (Output 1).

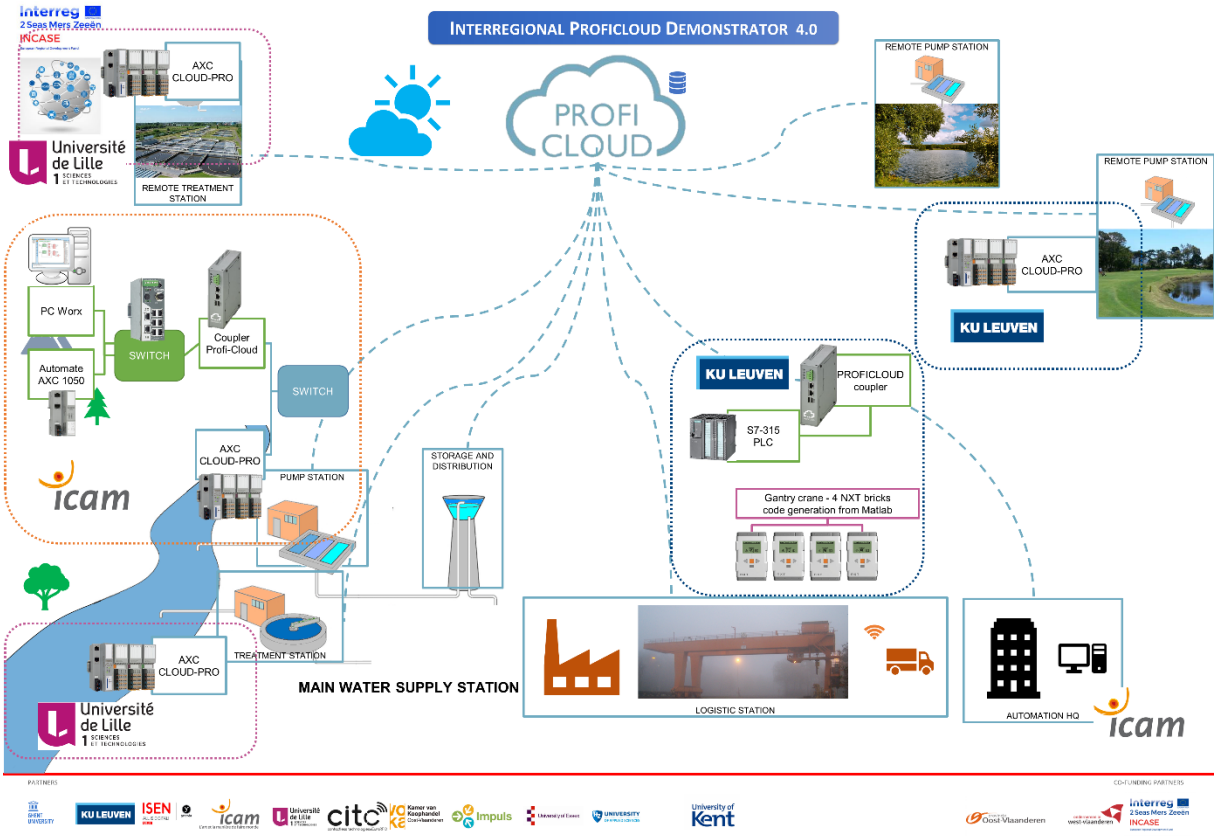


Figure 39: Interregional PROFIcloud Demonstrator

2.4. LEGO Vehicle

2.4.1. Introduction

This LEGO vehicle drives on two tables with a combined surface of 2 by 2 meters. It uses LEGO Mindstorms NXT CPUs running code generated from MATLAB/Simulink. It can be manually controlled from an Android tablet through a PLC and via a wireless link.

Figure 40: Schematic of the setup

2.4.2. Physical construction of the LEGO Vehicle

The LEGO vehicle is designed with four caterpillar tracks, each one driven by a motor. By using four caterpillar tracks, the vehicle is able to rotate around its axis on the spot. It also has four ultrasonic sensors to make sure the vehicle doesn't drive off the table.

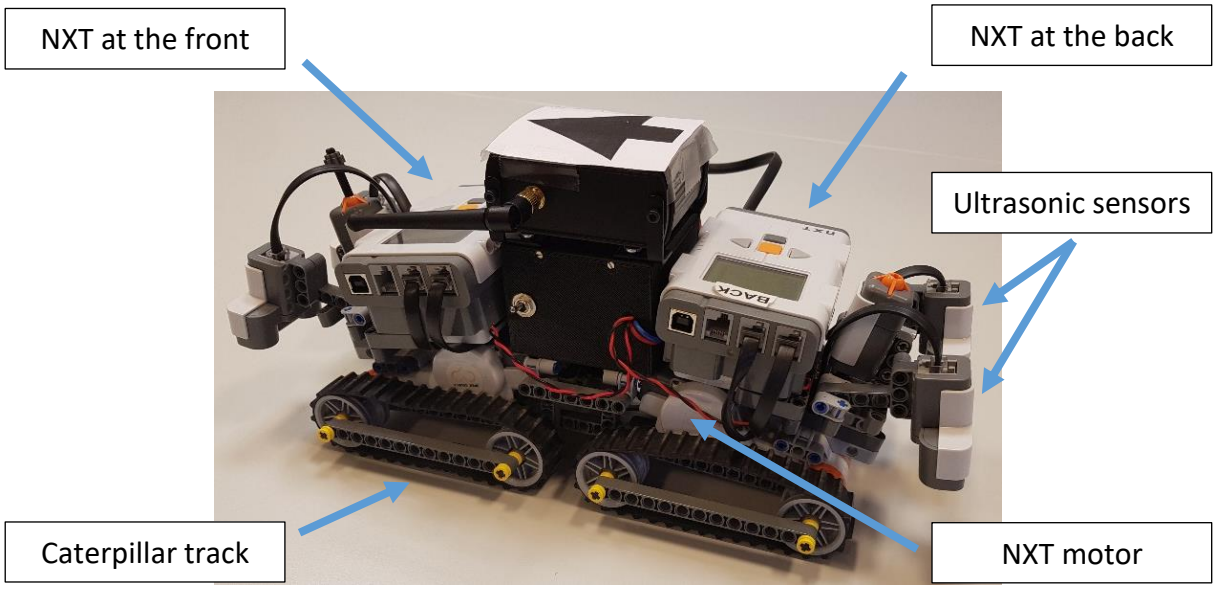


Figure 41: The LEGO vehicle

2.4.3. NXT execution speed

Measurement without PROFIBUS communication

With a step size of 1 ms in Simulink, the execution time ranges between 0.904 ms and 1.076 ms, with an average of 1.000 ms. The range varies 0.172 ms.

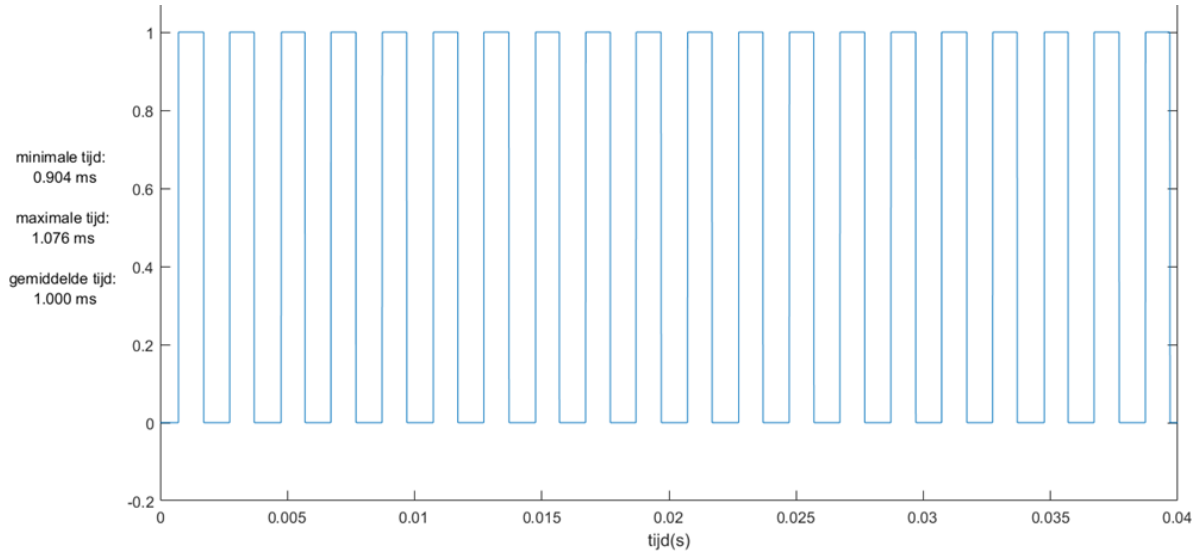


Figure 42: Execution time of the NXT without PROFIBUS communication

Measurement with PROFIBUS communication

With a step size of 1 ms in Simulink, the execution time ranges between 0.917 ms and 1.071 ms, with an average of 1.000 ms. The range varies 0.154 ms.

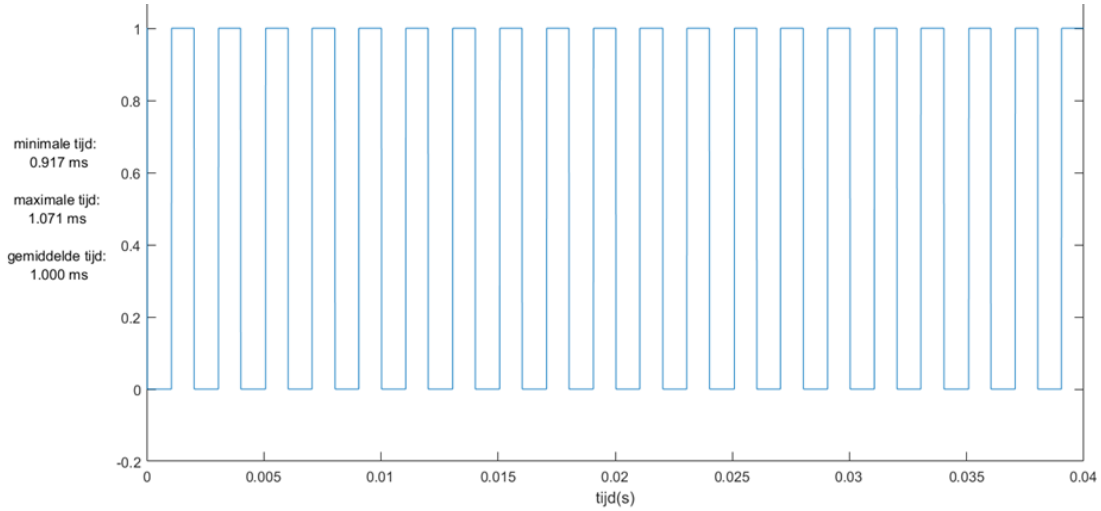


Figure 43: Execution time of the NXT with PROFIBUS communication

The use of the PROFIBUS communication doesn't influence the performance of the NXT.

Measurement for the fastest execution speed of the NXT

For this measurement, the NXT counts to 200 four times and switches an output each time. Then the output is switched on and off three times. This is programmed as S-function in C-code (see paragraph 2d) and deployed to the NXT using Simulink.

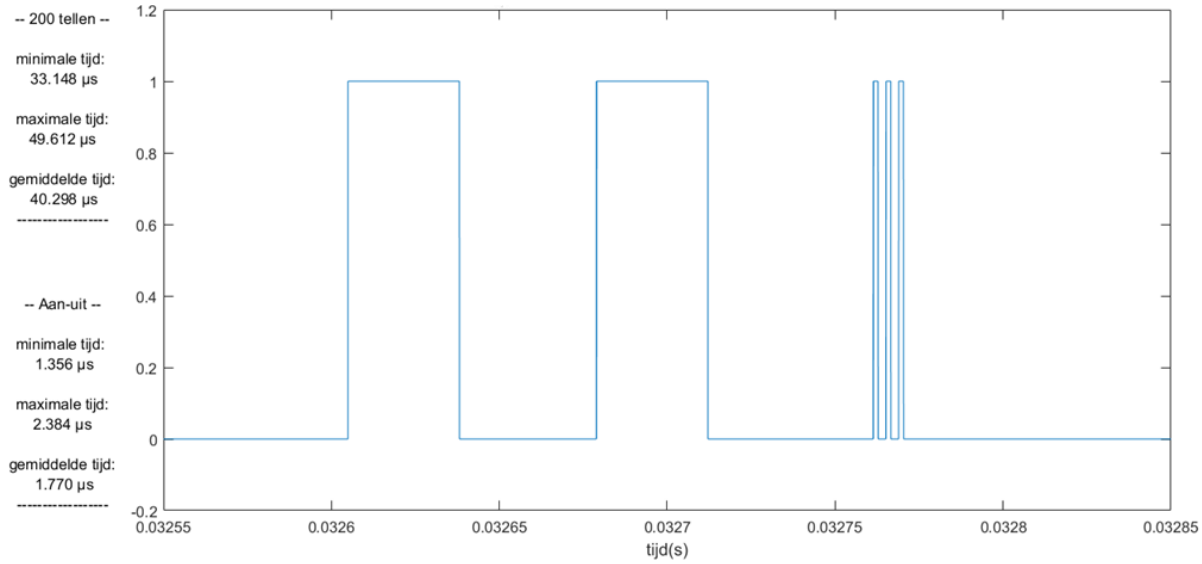


Figure 44: Fastest execution time of the NXT

The time needed to count up to 200 ranges between 33.148 µs and 49.612 µs, with an average of 40.298 µs. The range varies 1.028 µs.

The time needed to switch the output ranges between 1.356 µs and 2.384 µs, with an average of 1.770 µs. The range varies 16.464 µs.

2.4.4. Use of LEGO Mindstorms NXT

Because four motors are used and one NXT brick only supports three motors, there are two NXT bricks used in this vehicle. One NXT brick is connected to the motors and the ultrasonic sensors at the front, the other one at the back.

Figure 45: The LEGO vehicle

2.4.5. Automation of the LEGO vehicle

Controller

The controller used for the automation of the LEGO vehicle is a Siemens PLC, more specific, a CPU 1516F-3 PN/DP.

The PLC program will receive commands from the Android tablet and it will send the appropriate setpoint for each motor to the LEGO vehicle.

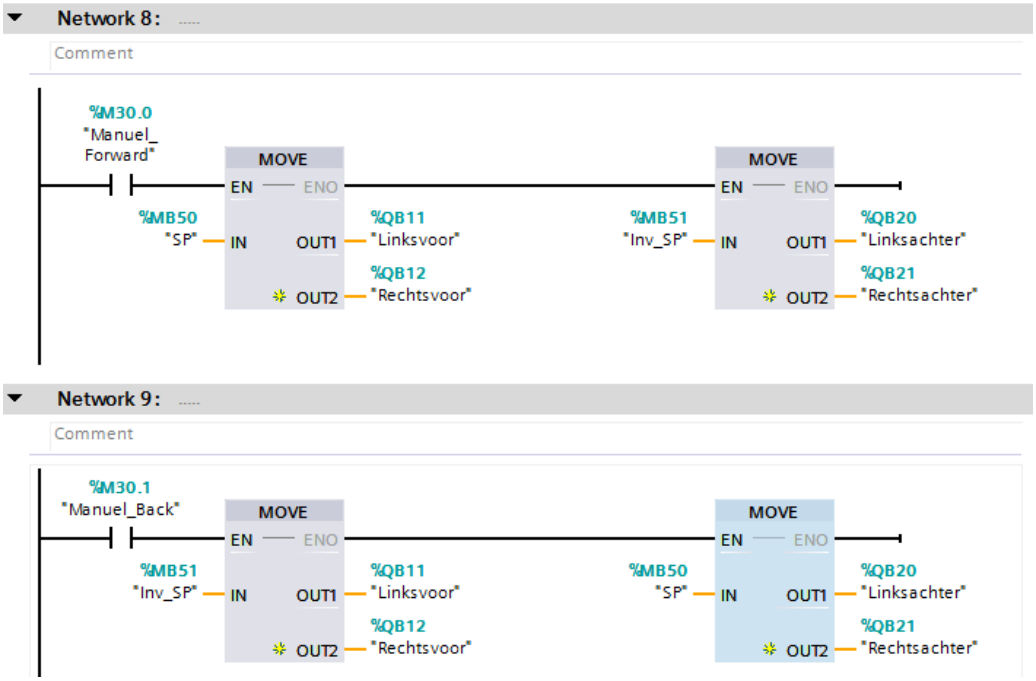


Figure 46: PLC program for driving forward or backwards

2.4.6. Communication between the PLC and the NXT bricks

TCP-connection

The commands from the Android tablet will be send wirelessly over a TCP connection to the PLC. The device used for this wireless connection is the Phoenix Contact FL WLAN 5100, this is a WLAN-access point.

Figure 47: Phoenix Contact FL WLAN 5100

PROFIBUS DP

The PLC will send the received commands wirelessly over PROFIBUS DP to the LEGO vehicle. The device used for the wireless transmission is the Anybus Wireless Bridge – Serial – Bluetooth (Anybus, sd). It

provides a wireless connection over Bluetooth for serial devices with an RS-232/422/485 interface. For the connection two devices are needed, one at the PLC and one on the LEGO vehicle.

Figure 48: Anybus Wireless Bridge – Serial – Bluetooth

Power supply

The Anybus Wireless Bridge operates with a power supply between 8 and 30 V DC. Since the batteries of the NXTs only supply 9V when fully charged, a boost or step-up converter has been used, more specifically a Velleman LM2577. This brings the supply voltage to 12 V. It has been decided to connect the two batteries in parallel so that the capacity of both drops equally. To eliminate consumption during inactivity, a switch is provided to switch the boost converter of the supply voltage.

Velleman LM2577:

- Input voltage: 3.5 to 35 V DC
- Output voltage: 5 - 55 V DC
- Max. input current: 3 A
- Constant input current: 2 A

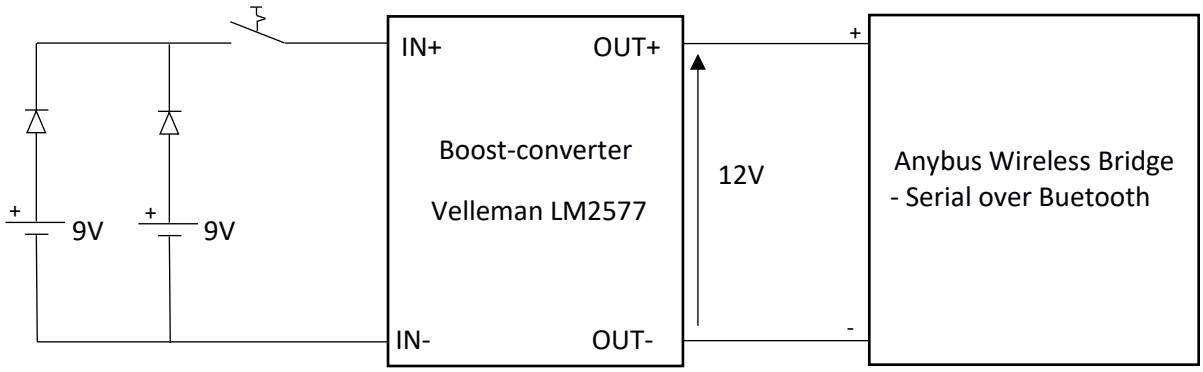


Figure 49: Connection diagram for the Velleman LM2577 and the Anybus Wireless Bridge

Physical connection

In order to establish the connection between the NXT and the Anybus Wireless bridge, there is no direct (straight through) connection possible. The I²C communication of the NXTs is converted to RS485 via MATLAB functions. The communication pins are pin 5 and 6. On the Anybus Wireless Bridge, the send and receive lines must be connected externally to allow half-duplex communication.

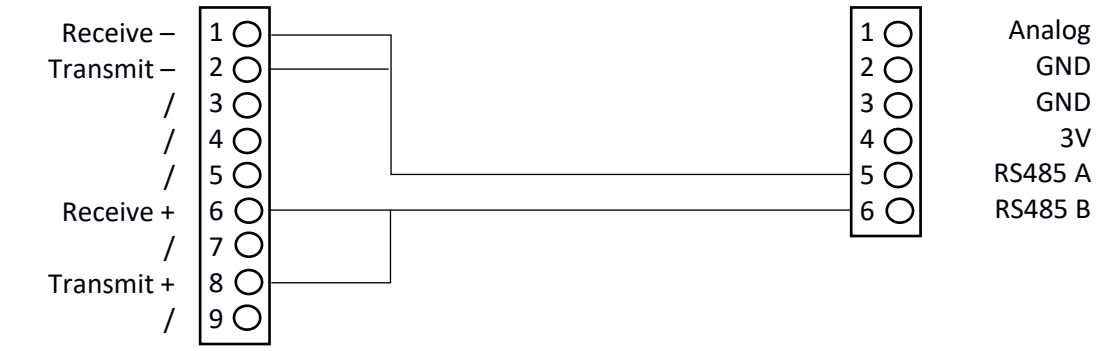


Figure 50: Connection diagram for the Anybus Wireless Bridge and the LEGO NXT

2.4.7. NXT Program

A PID controller – realised with Simulink generated code – is used to control the motors.

Optimised control parameters:

$$K_r = \frac{0.8\ .\ 0.9}{\left(0.011+\frac{0.005}{2}\right).\ \frac{21}{32}} = 1.59$$

$$\tau_i = 3\ .\left(0.011 + \frac{0.005}{2}\right).= 0.2025$$

$$\tau_d = \frac{1}{2}\ .\left(0.011 + \frac{0.005}{2}\right) = 0.00675$$

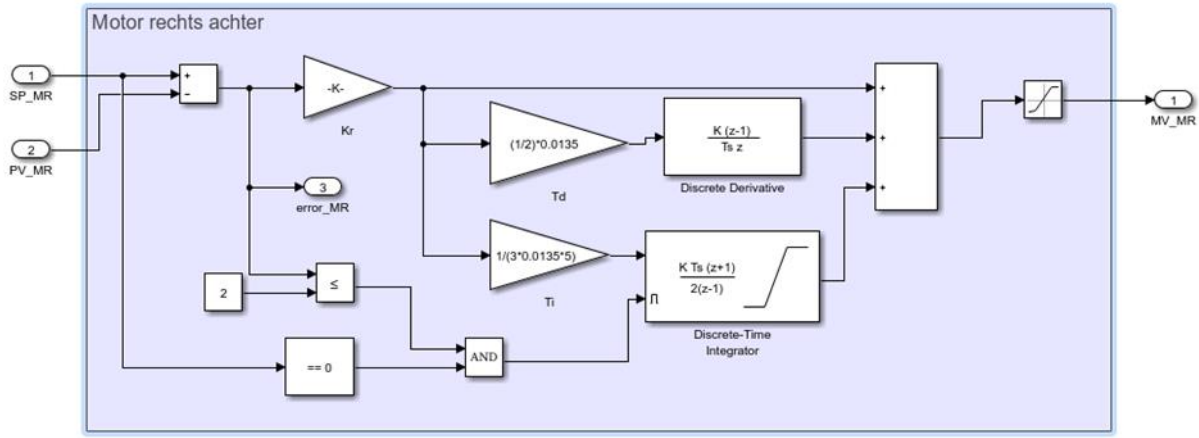


Figure 51: PID controller

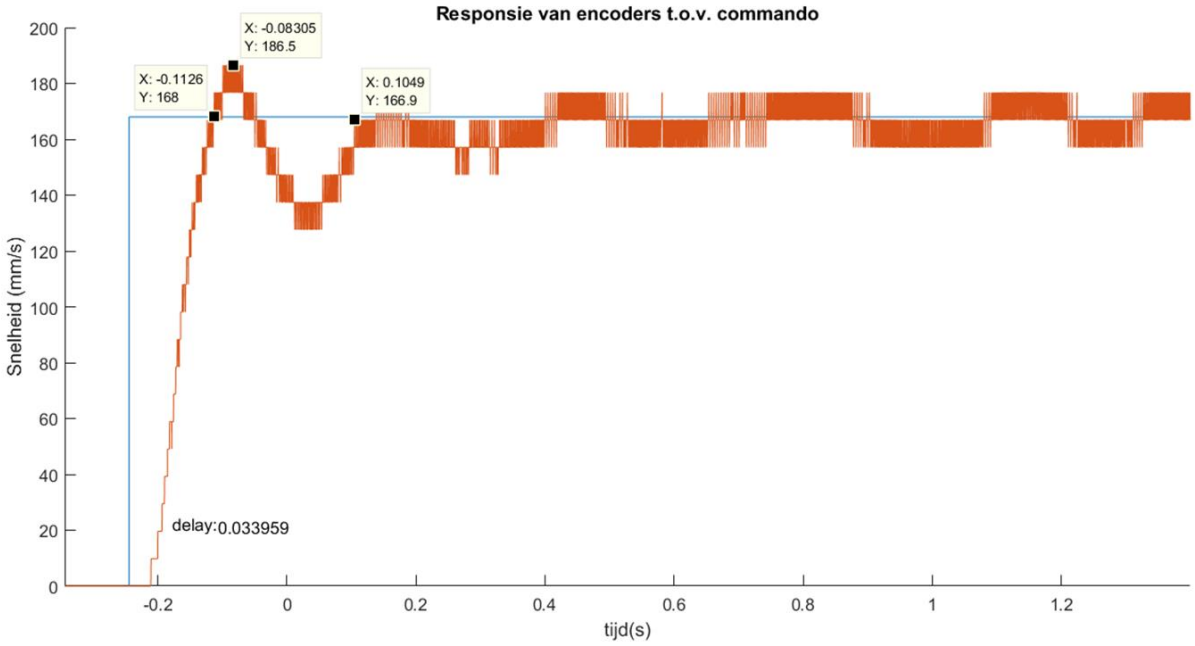


Figure 52: Step response with the PID controller implemented

Step response results for the PID controller:

- Rise time: 97.84 ms
- Overshoot: 11%
- Settling time: 348.93 ms.

2.4.8. Futue work

It is kept as an option to integrate a similar mobile vehicle in the crane application.

3. Low-cost hardware connected to industrial networks

3.1. Small robot controlled by PLC and Raspberry

3.1.1. Introduction

The aim of the experiment was to control an open source small robotic arm via an industrial PLC. A raspberry PI was used as a bridge between the robot (controlled via GPIO pins) and the PLC (via Modbus). An additional color camera was used to determine the action to perform. It was then possible to command the arm to grab, move and drop different blocks depending on their color detected by the camera. An illustration of the chain of control can be find in Figure 53.

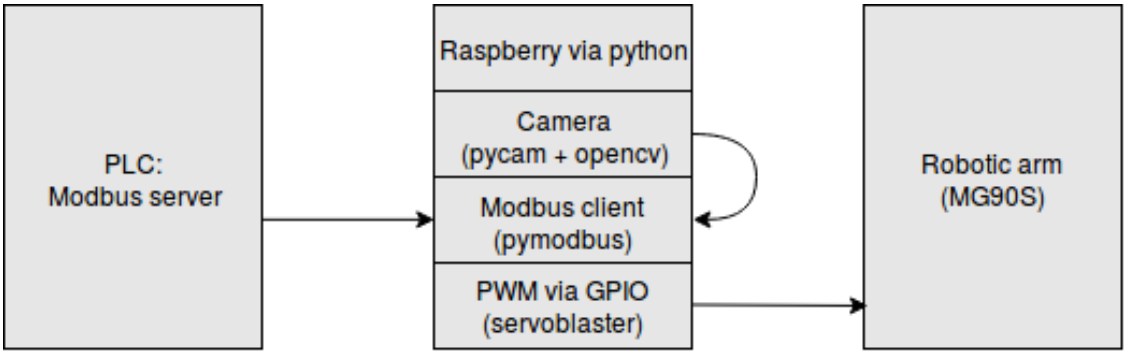


Figure 53: Controlling a robotic arm via a PLC

3.1.2. Equipments:

- Siemens PLC S7-1200: This was the PLC used for the demonstration. However, any modern PLC can be used as the experiment do not rely on any Siemens specific component.

- Raspberry PI: Used as a gateway between the PLC and the robot. Instructions from and to the PLC comes from an Ethernet RJ 45. The robot is controlled via the GPIO pins and the camera via the CSI port. The main advantage of the Raspberry PI is its low cost and the easy integration via python of several different libraries.

- Mearm (https://shop.mime.co.uk/collections/frontpage/products/mearm-pocket-sized-robot-arm): A kit for building a robotic arm. The aim of this device is purely educational. Despite that no proper industrial application would be feasible, this kit has the advantage to easily illustrate a proof of concept.

- Pi Camera Module (v2) (https://www.raspberrypi.org/products/camera-module-v2/): A Color Bayer Camera which is attached via a ribbon cable to the CSI port on the Raspberry Pi and on the waist of the robotic arm.

Figure 54 shows a picture of the global framework, where all equipments are connected together.

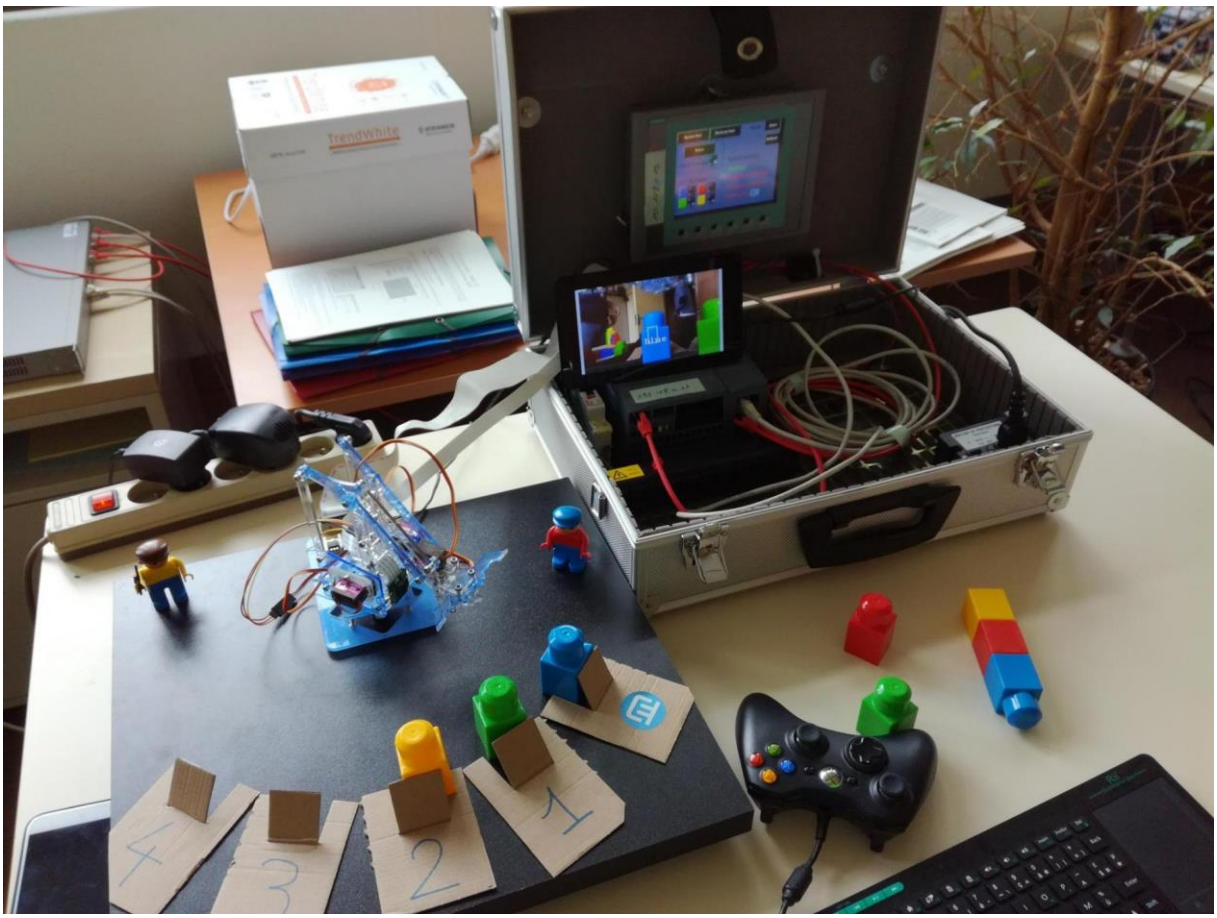


Figure 54: Global framework

3.1.3. Protocols and communication

PLC

The connection between the PLC and the Raspberry PI is based on a Modbus TCP link. This protocol can easily be found in modern PLCs, A server block is started on the PLC to control the arm. Despite that holding registers have been used for the prototype, inputs and outputs can also be used. However, using registers is more adequate to our demonstration as it requires integers to be send. Using input and output would require binary conversion and wouldn't bring anything useful for our demonstration.

Different commands can be send to the robotic arm:

- Grab: The arm will grab a piece in front of him.

- Drop: The arm will drop the piece he's holding in front of him

- Position to move: An integer representing fixed position for the robotic arm

- Move: The arm will move to the position specified in the "Position to move" variable.

From the arm, the PLC received different informations:

- Busy: During activities like grabbing, dropping or moving, the arm signals that it is busy doing a task. Only when this value is no more activated, the PLC can continue its program.

- Code of color: Represents the code of the detected color of the piece in front of the arm. Depending on the code, different actions can be chosen by the PLC.

Both the variables “Position to move” and “Code of color” are coded via integers. As holding registers are transmitted in Modbus protocol as a 2 bytes variables per registers, integers can be directly transmitted without using a bits to bytes conversion.

Raspberry-Pi

Different Modbus TCP client already exists for several programming language. It can also be easily coded as its protocol is fairly described and proper documentation exists, contrary to Profinet. This represents the main reason of using a Modbus protocol with respect to other ones.

In this case, we used the pymodbus package (https://pymodbus.readthedocs.io/en/latest/) which is a python library implementing a TCP Modbus client. The same data described in the section 3.1.1 from the PLC were then sent (received) to (from) the PLC. For the sake of simplicity, data were exchanged for every iteration of the program. As the different motions of the arm last around 500ms, the scanning frequency was high enough with respect to the demonstration. Clearly, depending upon the application and the equipment used, one can prefer to have a fixed timer for exchanging the data and this can easily be done via multithreading (as python allows it quite easily).

The robotic arm is basically controlled by its servos, in our case 4 MG90S controlled via PWM. No additional high-level layer is used to control the arm and there wasn’t any inverse cinematic used in our case. The different positions used (pulsations in microseconds) were fixed and recorded offline. Servos are controlled via a dedicated library (servoblaster) implementing PWM on the GPIO of the Raspberry Pi. Note that the values of pulsations of the servos can be modified manually using a Xbox controller connected to the Raspberry Pi and a dedicated python module.

Frames acquired by the camera module are displayed on a 7” touchscreen connected to the DSI port of the Raspberry Pi and processed using the OpenCV (Open Source Computer Vision) library. The detection algorithm of the color is quite easy since 5 colors can be detected here (blue, red, green, yellow and black): the average RGB color of a region of interest is automatically computed a each iteration of the program (each acquisition of a frame) and the detected color is then given by the color channel(s) with the maximal value(s), see Figure 55.

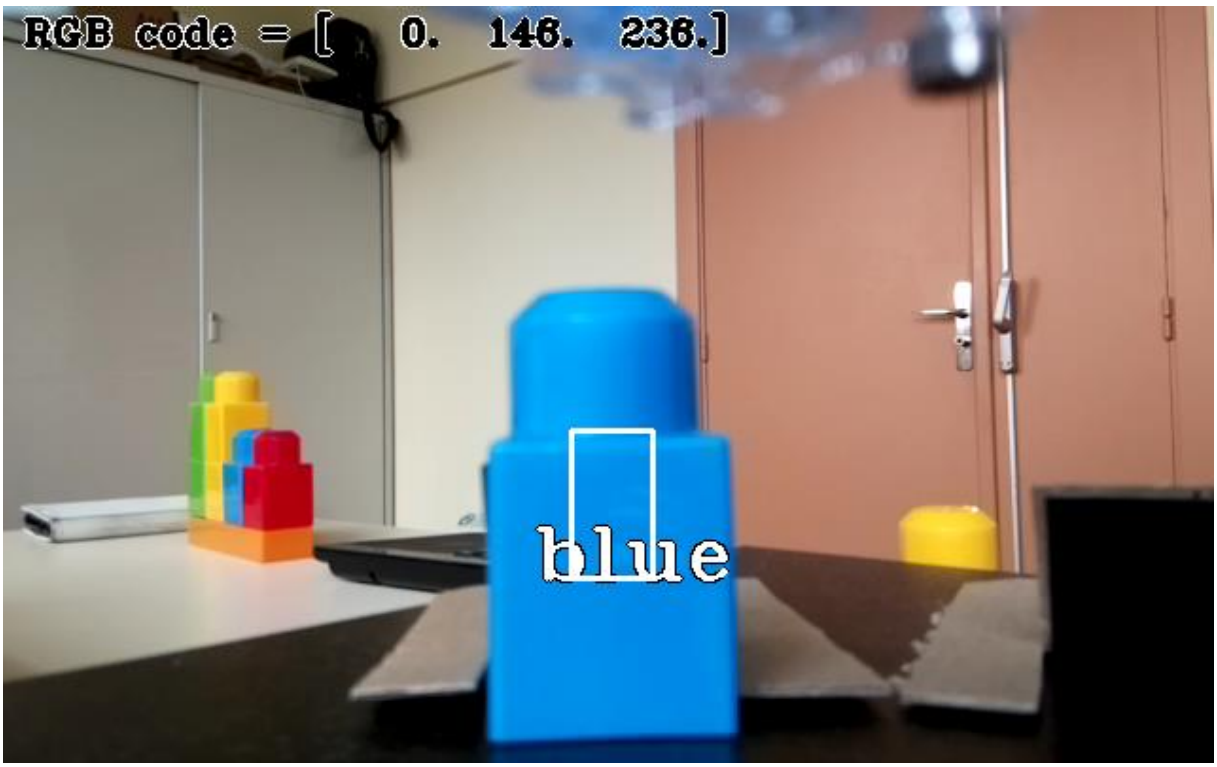


Figure 55 : Image of a block acquired by the camera. The ROI (white rectangle), the average color (RGB code) and the detected color (blue) are displayed/computed using the OpenCV library

3.2. Wireless PROFIBUS DP over Bluetooth

3.2.1. Introduction

A brief description of the wireless PROFIBUS DP link, used for the mobile vehicle (paragraph 2.4.2) can be found in this paragraph.

The code and connections can be found in paragraph 2.3.4.

3.2.2. Device

Figure 56: Anybus Wireless Bridge – Serial – Bluetooth

To transmit PROFIBUS DP wirelessly, the Anybus Wireless Bridge – Serial – Bluetooth (Anybus, sd) can be used. It provides a wireless connection over Bluetooth for serial devices with an RS-234/422/485 interface. Of course, two of these devices are needed to setup the wireless connection.

3.2.3. Setup

While setting up the PROFIBUS connection in the programming software (e.g. TIA Portal), the wireless bridge doesn't need to be added or configured. It is invisible for the PROFIBUS devices in the network.

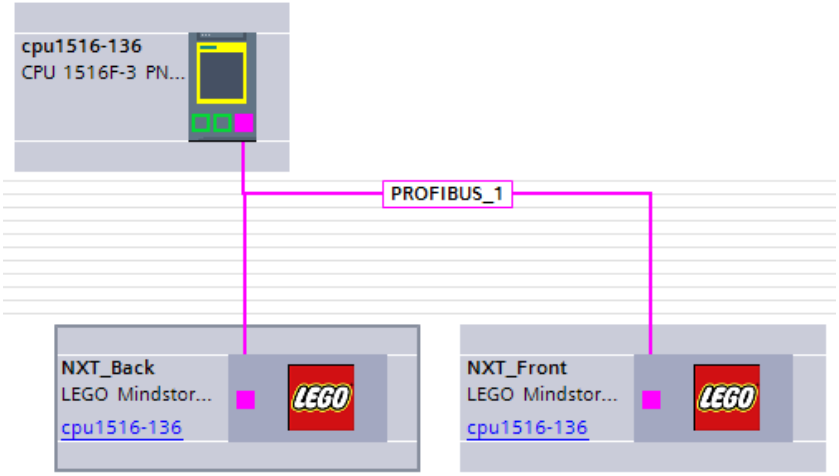


Figure 57: PROFIBUS configuration in TIA Portal

3.2.4. Delay measurement

Of course, a wireless link introduces some delay. This delay can be determined by measuring the PROFIBUS DP messages on one side and on the other side and calculate the time difference between the same PROFIBUS DP message. On average, the transmission delay is 14.74 ms, the maximum measured was 21.7 ms.

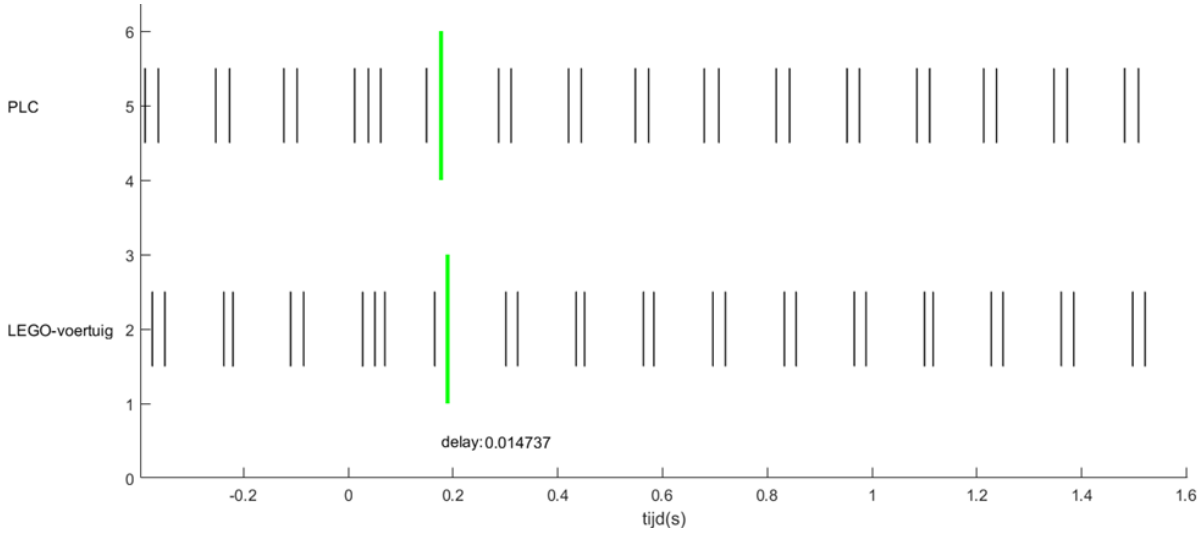


Figure 58: Transmission delay over the wireless bridge

3.2.5. Example of PROFIBUS DP messages

Communication between the master (address 76) and both NXT slaves (addresses 41 and 42). Only the frames from the master to the slaves are shown.

27	0.5530	SD2	76 -> 42	SRD HIGH	Data Exchange	93	0 0
36	0.6630	SD2	76 -> 41	SRD HIGH	Data Exchange	125	0 0
32	0.6790	SD2	76 -> 42	SRD HIGH	Data Exchange	125	0 0
42	0.7905	SD2	76 -> 41	SRD HIGH	Data Exchange	93	80 80
37	0.8064	SD2	76 -> 42	SRD HIGH	Data Exchange	93	7F 7F
48	0.9180	SD2	76 -> 41	SRD HIGH	Data Exchange	125	80 80
42	0.9339	SD2	76 -> 42	SRD HIGH	Data Exchange	125	7F 7F
54	1.0430	SD2	76 -> 41	SRD HIGH	Data Exchange	93	80 80
47	1.0614	SD2	76 -> 42	SRD HIGH	Data Exchange	93	7F 7F
60	1.1714	SD2	76 -> 41	SRD HIGH	Data Exchange	125	80 80
52	1.1878	SD2	76 -> 42	SRD HIGH	Data Exchange	125	7F 7F
66	1.2989	SD2	76 -> 41	SRD HIGH	Data Exchange	93	80 80
57	1.3164	SD2	76 -> 42	SRD HIGH	Data Exchange	93	7F 7F

Figure 59: PROFIBUS DP messages

3.2.6. Future work

Further analysis and another application in mobile ISEN robot is planned for end of summer 2018, during the traineeship of an ISEN student in KU Leuven.

4. Bibliography

Anybus. (n.d.). *Anybus Wireless Bridge - Serial - Bluetooth*. Retrieved from Anybus - Multi-network connectivity within Fieldbus and Industrial Ethernet: https://www.anybus.com/products/wireless-index/anybus-wireless-bridge/detail/anybus-wireless-bridge---serial---bluetooth

Halloy, H. (2017). M1 Internship Report.

MathWorks. (n.d.). *Simulink Support Package for LEGO MINDSTORMS NXT Hardware - File Exchange - MATLAB Central*. Retrieved from MathWorks - Makers of MATLAB and Simulink - MATLAB & Simulink: https://nl.mathworks.com/MATLABcentral/fileexchange/40311-simulink-support-package-for-lego-mindstorms-nxt-hardware

Simon Magiar, Mirela Gale(2017). Lowcost mobile robot pilot using integrated design: Rapid prototyping using MATLAB, Simulink and Stateflow for Arduino. Internship report, ISEN-Lille.

Hugo Carpentier, Mustapha Bouziane, Thomas Tarroza, Edouard Beudaert (2017). Lowcost mobile robot pilot using integrated design: Rapid prototyping using MATLAB, Simulink and Stateflow for Raspberry Pi3. M1 Project report, ISEN-Lille.