FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

IAAM
Institut für angewandte
Automation und Mechatronik

University of Applied Science Aachen

Faculty of Mechanical Engineering and Mechatronics

Project Report

Autonomous Mover:

Development of an autonomous guided vehicle for shop floor delivery

Submitted by:

Aadithya Ramamurthy

Michael Knoll

Rachana Kodila

Sameer Tuteja

Vedhashruthi Harinath

Supervised by:

Prof. Dr.-Ing. Jörg Wollert

Victor Chávez-Bermúdez

March 2022

Declaration of Independence

We hereby certify that we have written this paper independently and have not used any sources other than those listed in the bibliography. Passages taken verbatim or in spirit from published or as yet unpublished sources are identified as such.
Drawings or illustrations in this work have been prepared by us or are accompanied by an appropriate source reference.
This work has not been submitted in the same or similar form to any other examination authority.

Place, Date Signature

Place, Date Signature

Place, Date Signature

Place, Date Signature

Place, Date Signature

Table of Contents

List of Figures

List of tables

Acronyms

PLC	Programmable Logic Controller
AGV	Autonomous Guided Vehicles
AR	Augmented Reality
ERP	Enterprise Resource Planning
VDA	Verband der Automobilindustrie
ROS	Robotic Operating System
IR	Infrared
PSD	Position Sensitive Detector
IRED	Infrared Emitting Diode
LDO	linear voltage regulator
NC	Normally Closed
IO	Input Output
MEMS	Micro Electro-Mechanical Systems
DI	Digital Input
DO	Digital Output
ISO	International organization of Standardization
CAN	Controller Area Network
CSMA/CR	Carrier Sense Multiple Access/Collision Resolution
CRC	Cyclic Redundancy Check
RTR	Remote Transmission Request
DLC	Data Length Code
SPI	Serial Peripheral Interface
IODD	IO Device Description
PD	Process Data
SLAM	Simultaneous Localization and Mapping
eMMC	Embedded Multimedia Card
RANSAC	RANdom SAmple Consensus
k-d	k-Dimensional
LIDAR	Light Detection and Ranging
PWM	Pulse Width Modulation
USB	Universal Serial Bus
UART	Universal Asynchronous Receiver-Transmitter
GPIO	General Purpose Input/Output
ASCII	American Standard Code For Information Interchange
OPC UA	Open Platform Communications United Architecture
DBC	Data Base Container
URDF	Unified Robot Description Format
RVIZ	ROS VIZualization
DIN	Deutsches Institut für Normung
ADC	Analog to Digital Converter
DLC	Digital Loop Carrier
I2C	Inter-Integrated Circuit

MC	Motor Controller
MTR	Motor
IC	Integrated Circuit
GND	Ground
RFID	Radio Frequency Identification
FMEA	failure modes and effects analysis

Structure

CHAPTER 1: In this chapter, introduction to the AGV Mover project and its components is given along with the contributions of the batch 2020-2021.

CHAPTER 2: In this chapter, the work of batch 2021-2022 and their results is recorded.

CHAPTER 3: This chapter consists of the conclusion and further developments.

CHAPTER ONE – SYSTEM ANALYSIS

1.1 Introduction

1.1.1 Overview

In March 2018, the new Industry 4.0 Fab was opened at FH Aachen. In this Fab, technical stations are implemented for the construction of Lego cars and E-longboards, with more complicated scenarios being introduced in the coming years. To achieve connections between the warehouse, packing stations, and various production stations, the shopfloor delivery will require autonomous guided vehicles. These autonomous guided vehicles (AGVs) will function similarly to Amazon Robotics' Kiva mover and should be able to transport a payload of 100 kilograms with PLC as main controller and ROS on High level interface which can detect stations with AR Tags.

In this chapter we will be seeing what the main goal of the project is and how the project was handed over to the group of students working on the project from the period April 2021-March 2022.

1.1.2 Goal

Several tasks are encompassed in the design and development of a system for an AGV. The first, and perhaps obvious, is to design and manufacture the construction of the mover. The team must also develop a concept for automation based on either an embedded controller or PLC technology. In parallel with the aforementioned task, the AGV must also be fitted with the necessary sensors and actuators to implement and assist with autonomous movement. At a higher level, ROS should be implemented to achieve free navigation. Furthermore, the team should conduct research on how to optimize the movement and control of the AGV. Finally, the mover should be fully integrated into the industry 4.0 Fab's Enterprise Resource Planning (ERP) System. The main requirement is that the mover should be within 1500C. And due to previous experience and unreliability of Chinese cost-effective components, it would be recommended to select reliable (preferably German components for AGV).

This list provides the information about the goal, vision of the project which we will try to realize in this project phase.
To finish the mechatronics project, the following tasks has to be fulfilled:

1. Documentation of the final version of the rover

2. Detailed cost evaluation of the rover

3. Generation of a CAD model of the version "model factory" including parts list, drawings, etc. that we are able to manufacture and order all required components without your help.

4. Simulation of the path finding & movement in ROS along with a detailed software documentation.

1.1.3 State of the art

To be able to understand the needs of the bot and to identify which direction to pursue and how to improve on what the competition offers, the team investigated the existing systems and movers currently available in the market Bito's Leo Locative, SSI Schäfer's AGV Weasel, and Locus Robotics' LocusBot, Kiva and the Hercules are the firms and their movers that were primarily focused on. Despite looking at the Kiva, the team decided that its applicability was too particular (and limited) to Amazon's needs to be considered and it was also determined that Kiva was used to carry shelves worth 450kgs which was not something required by our AGV to do. However, its hardware and integration with sensors were researched. The physical characteristics and specifications, the battery, the mechanism for carrying objects, and the navigation/level of autonomy were the four primary variables investigated (including types of sensors used). The range of the load mass was between 20 and 45 kg, and with both the Leo and Weasel base shaped as a rectangle. The LocusBot was unique in that it resembled a Segway more than a cart.

All batteries provided at least 8 hours of operating time with the Weasel even reaching up to 16 hours, with a minimum charging time requirement of 4 hours. All the researched movers made use of bins to hold the items for transport. The LocusBot allowed for compartmentalizing the bins for separate orders. As for navigation, the Leo Locative and AGV Weasel use line following and QR codes on the ground to navigate to the necessary station for pickup/delivery. They also make use of IR sensors for any obstacle detection. The LocusBot's navigation is much more complex, where it communicates between the Warehouse Management Server and employee prompts with a server, being able to achieve full autonomy. The LocusBot uses LIDAR, alongside IR sensors, to both detect obstacles and to map the warehouse, allowing it to constantly update the map in the server. An algorithm is also used to plan a path, reroute, and prioritize orders. Price is also a factor that was considered; the Leo Locative and AGV Weasel both costs less than 10.000C, while the LocusBot is expected to cost between $30.000 and $35.000. The team believes that the mover technology of Leo Locative and AGV Weasel can be replicated in the time allocated, while the LocusBot functionality would be a stretch goal.

Figure 1 Bito's Leo Locative, LocusBot, SSI Schäfer's AGV Weasel, Amazon's Kiva Systems

1.2 Lastenheft

1.2.1 Specific requirements

The main task is to develop & manufacture the mover construction with the most economical and modular solution (a goal of less than 1500 euros) without losing functionality, with the ultimate goal of full autonomy. The design of the rover was based on the priority of transporting containers manufactured by Auer Packaging GmbH that conform to the VDA 4500 standard for industrial stacking containers. These containers are designed to allow easy stacking & storage. This particular design also suited for conveyor belt systems found commonly in industrial factories, making it easily integrable for logistics solutions. With that in mind, the rover was dimensioned to transport a base container with the maximum base area of 60x40 cm. The CAD design of container is mentioned below:

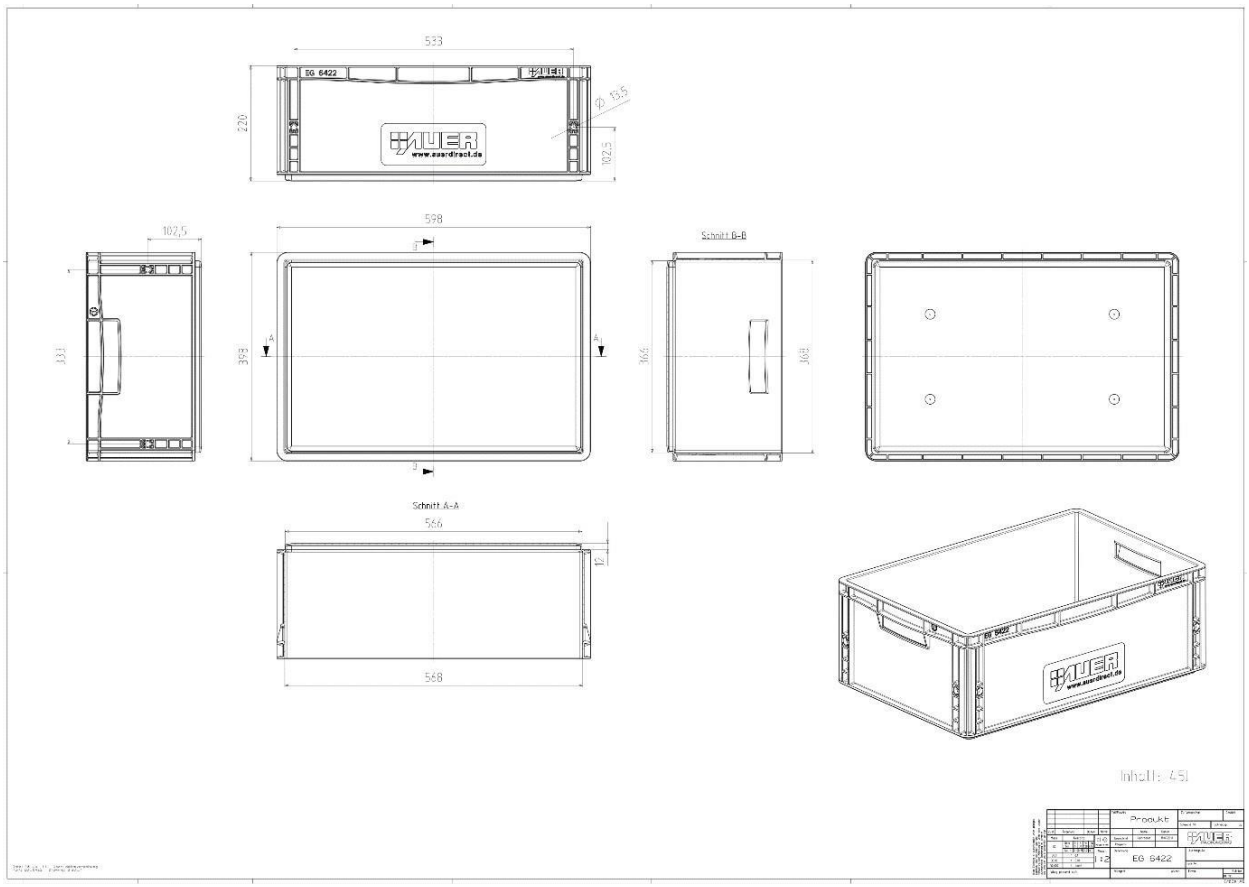


Figure 2 Technical drawing of the container by Auer GmbH.

1.2.2 Functional Requirements

Functional Requirements					
Serial Number	Require-ments	Description	Remark	Proposed Solution	Required Components
1	Robust	It should be sturdy and stable enough to support the needed weight but also move swiftly.	ANYS analysis of the robot has been conducted to analyze the total impact of the load.	A heavy metal body needs to be used by the stack-based bot.	Compact enclosure by Rittal International
2	Safety	The AGV must be safe for human handling.	All safety measures followed as per the EU standards needs to be followed.	PLC is used which shuts down the system during hazards.	
3	HMI	User friendly and intuitive interactions between the AGV and a human user.	Real time system with ROS of high-level interface helps to have a good user interface.	WAGO Ecockpit and ROS on high level.	WAGO PLC, ROS
4	Compact ness	A size that allows the AGV to travel about a workplace and into tight locations.	Need to take care if the accessibility of the components during maintenance.	Stack version is used to mount the components inside the enclosure.	
5	Ease of Manufac turing and Assembl y	Easily replicable model for the future groups.	Care needs to be taken as to not compromise on the robustness.	Assembly is done and analyzed with ANSYS software.	

6	Autonomous movement.	Using Line follower to allow maneuver through the path.	Reflection of light from the line and surface needs to be used.	IR sensor is used to follow the marked line.	IR sensor and USB camera
7	Obstacle avoidance	The rover has to avoid obstacles such as humans, machines, other rovers etc.	Detection of obstacle is a necessity as the AGV is supposed to move autonomously.	24V Ultrasonic sensor is used for this purpose.	Ultrasonic sensor
8	Step Detection	The system should avoid edges, steps etc.	Detection of the steps for safety measures	Sharp IR sensors are used for this purpose.	Sharp IR sensors
9	Autonom-ous loading and unloading	The AGV should load and unload from a point or a station.	The bot should support the loading pallet on the top.	Top pallet is supported into the bot.	8 wandbefestigungschalter

Table 1 List of functional requirements

1.2.3 Non-functional requirement

Non-functional requirement			
Serial Number	Requirements	Description	Remark
1	Environmental consideration	The intended environment of the target system is a factory or warehouse.	It has to travel at the temperature at the particular warehouse.
1.1	Conditions	The bot is to operate, and be stored, in dry environments, with a tolerance for small spills	
1.2	Floor Types	Operational on different floor types and surfaces •Concrete •Epoxy •Rubber •Tiles •Metal Plate •Wood	Maintain a coefficient of friction of 0.4 that would allow the bot to run on all surfaces.

2	Battery Level	Removable lithium batteries. Capacity: 7Ah, Voltage: 24V Box Dimensions: 151 x 65 x 99 mm Weight: 2.05kg	At least 2 hours of charge time at extreme load (24V).
3	Full Dimension	60 x 40 cm in length and width	Based on the Auer Packaging Gmbh industrial containers VDA 4500 standards.
4	Kill Switch	Manual switch that will totally shutdown the AGV	Manual switch is required during hazardous situations.
5	Operating Speed	The speed at which the bot moves.	Min: 0.1 m/s , Max:1 m/s
6	Edge Detection	To ensure the bot doesn't fall over an edge	
7	Transport unit weight	80 Kgs payload and 20 Kgs Bot	
8	Transport unit dimension	Rittal Compact enclosure 500 x 500 x 300 mm	Stability of bot, Rittal Compact enclosure
9	Max Acceleration	1 m/sˆ2	To prevent jerking which will affect/tilt the balance of load.
10	Runtime	At least 1hr	Depends upon battery capacity, 12V, 7Ah Rechargeable battery
11	Slope	Max 15deg	
12	Turning radius	1m	Bot trackwidth is 500mm

Table 2 List of Non-functional requirements

1.3 Pflichtenheft

There could be a variety of ways for manufacturing and constructing an open path AGV, depending on the necessity (Lastenheft). We've detailed our approach to this problem statement in this section.

1.3.1 Abstract

With the advancement of technology, industrial automation is getting more attention of the researchers to make life of mankind easier and more comfortable. To achieve that, the open path steering system came through to cover the problems inherent in the former.

With the help of various sensors, the AGV is automated without physically changing the system. Using grids, paths are assigned in the supervisory control, such as ROS, which in turn have real time control with PLC controlling system. Therefore, the location of the AGV and also the probable deviation is controlled and supervised by the supervisory unit. To achieve this, PLC control system for motion control with ROS for navigation has been implemented.

1.3.2 Scope of delivery

There are two alternatives for developing and producing an AGV: one that is within 1500 euros and the other that is compliant with industrial standards and more reliable, referred to as the Model factory version.

1.3.3 Possible saving measure

Use of low-cost Logitech USB camera instead of readily available USB camera can be used as a possible saving measure. Use of Motor Hub wheel instead of Mecanum wheels is also a low-cost alternative which still suits the needs of the AGV.

1.4 Motor and motor controller selection

1.4.1 Brushless DC Motors

The brushless DC motor has expanded into numerous applications due to its dependability and lifespan. It's used in a wide range of industries, including manufacturing, computers, and much more.

Advantages:

-Less overall maintenance due to lack of brushes.

-Operates effectively at all speeds with rated load.

-High efficiency and high output power to size ratio; -Reduced size with far superior thermal characteristics.

-Higher speed range and lower electric noise generation.

Figure 3 Wheel along with motor inside it

Figure 4 Planetary gear drive in Motor Hub

There are three Hall sensors spaced approximately 120 degrees apart. It is estimated that it contains 20 permanent magnets thereby 10 permanent magnet pairs.

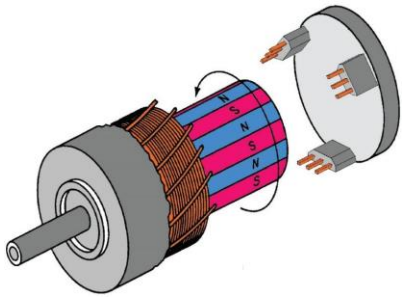


Figure 5 Planetary gear drive in Motor Hub

1.4.2 Motor calculations

While doing the initial motor calculations, the group had to make some assumptions in order to find initial values and be able to order a motor.

Assumptions:

- Two driven wheels
- Assume two pairs of caster wheels as support
  The design is for the worst-case scenario, meaning that the supporting caster wheels are not in contact with the floor and the drive wheels are bearing all of the weight.
- floor friction: $\mu = 0.4$
- driven wheel diameter $d = 165\ mm$, we will also look through a range of diameters:$d_{range} = 100mm, 101mm,$ (the 165mm is compared to wheels used by hoverboards. Our logic is that the hoverboards carry similar weights to our bot.)
- Assuming we're using the motors available in the storage room, the motor mass is $m_{motor} = 1.186\ kg$ and the gearbox mass is $m_{gearbox} = 0.76\ kg$ for a combined mass of $m_{drive} = m_{drive} + m_{gearbox} = 1.964\ kg$
- Maximum load mass is $m_{maxload} = 40kg$, and minimum load mass is $m_{minload} = 20kg$
- Rover mass will be $m_{rover} = 50kg$ (this is a conservative value, higher than the competition, but can be subject to change depending on our final design and the materials used)
- Total mobile drive unit mass:

$$m_{unit} = 2m_{drive} + m_{maxload} + m_{rover} = 93.892\ kg$$

Next, the group made an assumption of the speeds that the AGV would be driving at.

$$V_{max} = 1\ m/s$$

$$a_{max} = 1\ m/s^2$$

$$rpm_{max} = v_{max} \cdot \frac{1}{\frac{d_{wheel}}{2}}$$

In order to visualize the contact of the wheels with the ground and any forces acting on the objects, free-body diagrams were drawn, which can be seen in Figures 7 and 8. Next, the torque of the wheels to maintain traction was calculated, considering both scenarios presented in the free-body diagrams.

$$F_{Ntotal} = m_{unit} \cdot g = 920.766\ N$$

$$F_{Nwheel} = \frac{F_{Ntotal}}{2} = 460.383\ N$$

$$\tau_{trac} = \mu \cdot F_{Nwheel} \cdot \frac{d_{wheel}}{2} = 15.193\ N \cdot m$$

$$\tau_{tracincline} = \mu \cdot F_{Nwheel} \cdot \frac{d_{wheel}}{2} \cdot cos\ (15deg) = 14.675\ N \cdot m$$

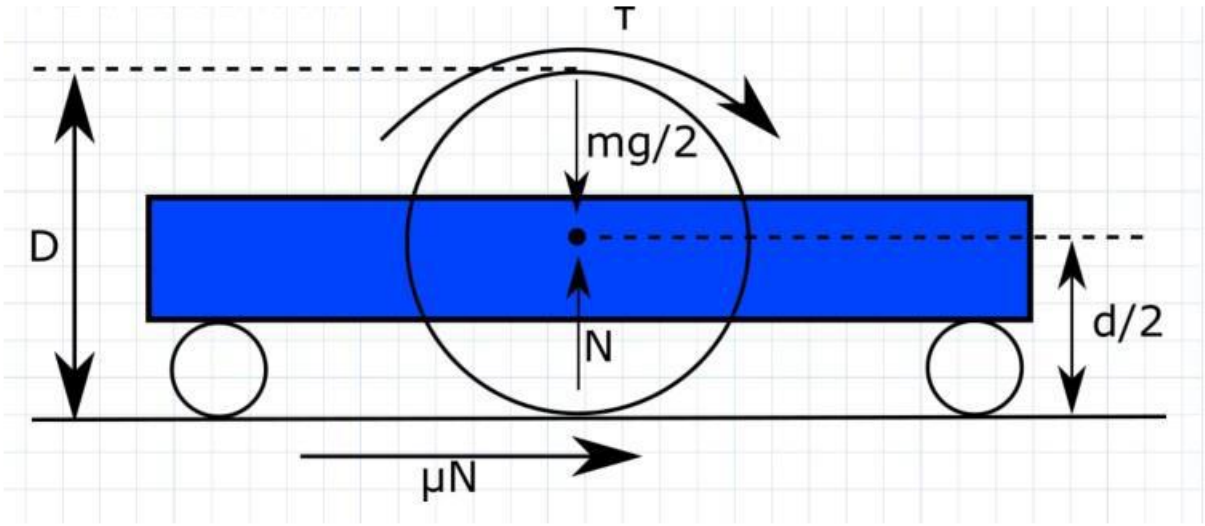


Figure 6 Free-body diagram of the AGV on a flat surface

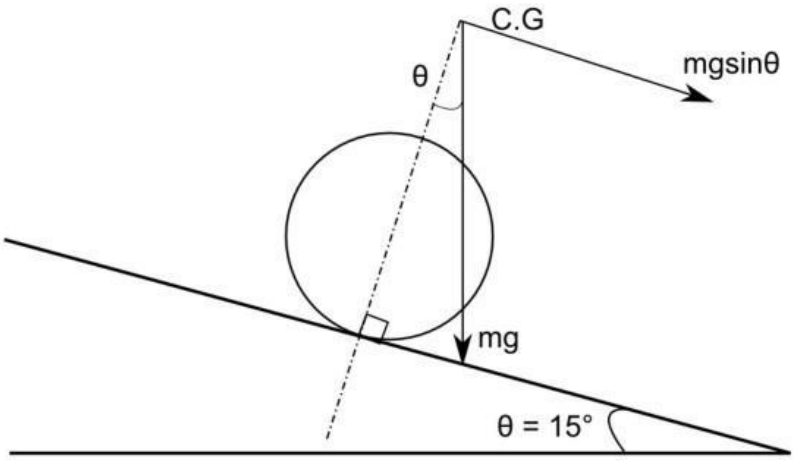


Figure 7 Free-body diagram of the wheel in contact with an inclined surface

The torque at an initial assumed acceleration of $2\ \frac{m}{s^2}$ was then calculated:

$$F_{acc} = a_{max} \cdot m_{unit} = 187.78\ N$$

$$\tau_{flat} = F_{acc} \cdot \frac{d_{wheel}}{2} = 15.492\ N.m$$

$$F_{incline} = m_{unit} \cdot g \cdot sin\ (15deg) = 238.312N$$

$$F_{max} = F_{acc} + F_{incline} = 426.096N$$

$$\tau_{max} = F_{max} \cdot \frac{d_{wheel}}{2} = 35.153N.m$$

$$\tau_{maxwheel} = \frac{\tau_{max}}{2} = 17.576\ N.m$$

17.576 Nm is not feasible since it exceeds the maximum traction torque the wheels can transmit. To determine what the maximum achievable acceleration for one wheel is, Eqn. $T_{maxwheelrange}$ (a_{max}) was plotted against the two traction torque values in Eqns. T_{trac} and $T_{tracincline}$ in Fig. 10.

$$F_{accrange}(arange) = a_{range} \cdot m_{unit}$$

$$\tau_{flatrange}(arange) = F_{accrange}(arange) \cdot \frac{d_{wheel}}{2}$$

$$F_{maxrange}(arange) = F_{accrange}(arange) + F_{incline}$$

$$\tau_{maxrange}(arange) = F_{maxrange}(arange) \cdot \frac{d_{wheel}}{2}$$

$$\tau_{maxwheelrange}(arange) = \frac{\tau_{maxrange}(arange)}{2}$$

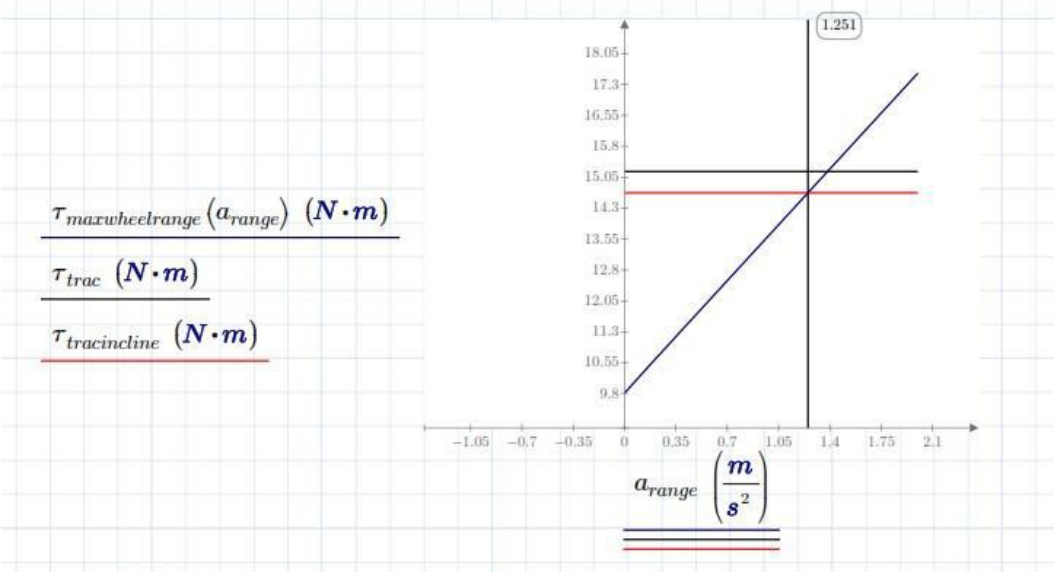


Figure 8 Plot showing the achievable acceleration to traction on a flat inclined surface.

The maximum torque that can be transferred by the wheel is the $\tau_{tracincline}$.

As well as considering that the rover has to travel an inclination of 15deg., we can conclude that the maximum acceleration that is achievable is a_{max}= 1.251 m/s2.

$$\tau_{provMotor} = 0.5\ N \cdot m$$

$$N = 25\ \ Gearbox\ ratio$$

$$\tau_{maxprov} = \tau_{provMotor} \cdot N = 12.5\ N.m$$

$$\tau_{ranged}(d_{range}) = \mu \cdot F_{Nwheel} \cdot \frac{d_{wheel}}{2}$$

$$\tau_{maxwheelrange}\left(1.25086\frac{m}{s^2}\right) = 14.675\ N \cdot m$$

$$m_{unitrange}(m_{loadrange}) = m_{loadrange} + 2 \cdot m_{drive} + m_{rover}$$

$$F_{Nange}(m_{loadrange}) = m_{unitrange}(m_{loadrange}) \cdot g$$

$$F_{wheelrange}(m_{loadrange}) = \frac{F_{Nrange} \cdot (m_{loadrange})}{2}$$

$$\tau_{rangem}(m_{loadrange}) = \mu \cdot F_{wheelrange}(m_{loadrange}) \cdot \frac{d_{wheel}}{2}$$

Final requirements:

Torque required per wheel, τmax wheel range (1.25086 m/s2)=14.675 Nm Required motor rpm for maximum velocity of 1 m/s , rpm(max)=115.749 rpm

To comply with our requirement of torque, a cheaper motor Hub wheel along with a controller has been selected.

Power calculations

A PNOZ X5 relay is available, which requires 24V and 4A and has a maximum output power of 100W per output channel. We'll assume there's a total weight of 100kg. A linear velocity of 1 m/s (125.3 rpm) and 0.5 m/s (125.3 rpm) are the two speed examples (63 rpm). When two wheels are in contact with the ground, a torque of 15 Nm is required per wheel, but when three wheels are in contact, a torque of 10 Nm is required per driving wheel.

Based on these assumptions, the power is determined to be

		Wheels contacting ground	
		2	3
Linear Velocity	1 m/s	200W	130W
	0.5 m/s	100W	66W

$$F_{Ntotal} = 96\,kg \cdot 9{,}81\,m/s^2 \approx 920\,N$$

$$F_{Nwheel} = \frac{F_{Ntotal}}{2} = 460\,N$$

Characteristic behavior of motors

As the motors are controlled with PWM signals provided by the Arduino to throttle of motor driver, the voltage at which the motor starts to overcome its inertia needs to be checked along with what the maximum achievable RPM is with highest control voltage of 5V supplied to the driver. The tests are conducted for each motor separately for both forward and reverse direction and can be seen as follows. The RPM range which can be achieved with these BLDC motors is from 70 RPM to 170 RPM.

1.4.3 Motor Controller

To control the brushless DC motor and wheel hub, a motor controller had to be used. The motor controller, shown in Fig. 12, takes 24V, 17A, and 200W. The 200W is enough power to provide the required torque to the wheel. Several wires come out of the controller, each with different functions. Some basic information was given by the manufacturer for what each wire does, however, most of the functions and uses were tested through trial and error. Below are the detailed descriptions for each wire.

Hall sensor lines: These connect directly to the hall sensor lines on the motor and are able to check the speed of the motor.

3 phase line: These are used to modulate the current going to the motor and hence the speed of the motor.

Battery negative/positive voltage: These lines are connected to the negative and positive of the supply.

Assist sensor: There was no conclusive function observed.

Low potential brake: These lines when toggled (using a relay switch), activate the brake of the motor.

Intelligent Identification: It was found that toggling of this switch (using a relay) lets us switch between forward and reverse direction of rotation of the motor.

Furthermore, if this switch is toggled and then left in the toggled state, the motor continues to run and when a change of speed input is provided, the motor only changes its direction of rotation, and at a constant speed.

Since we wish to control the speed and direction at all times, we are using the first (toggle on and off immediately) mode.

Speed Steering handle: This connection is used to connect to the potentiometer or DAC in order to control the speed of the motor.

One may have noticed that if one were to run both motors while mounted to the AGV, the vehicle would just spin in a circle. Both are technically running forward, but since one is the mirror opposite of the other in the assembly, it is spinning backwards. Hence one motor has to be running in the reverse direction while one has to be moving in the forward direction. This motor controller only responds to input values between 1.2 and 3.8 but could differ for different motor controllers. In our case, both the motor controllers respond in a similar way.

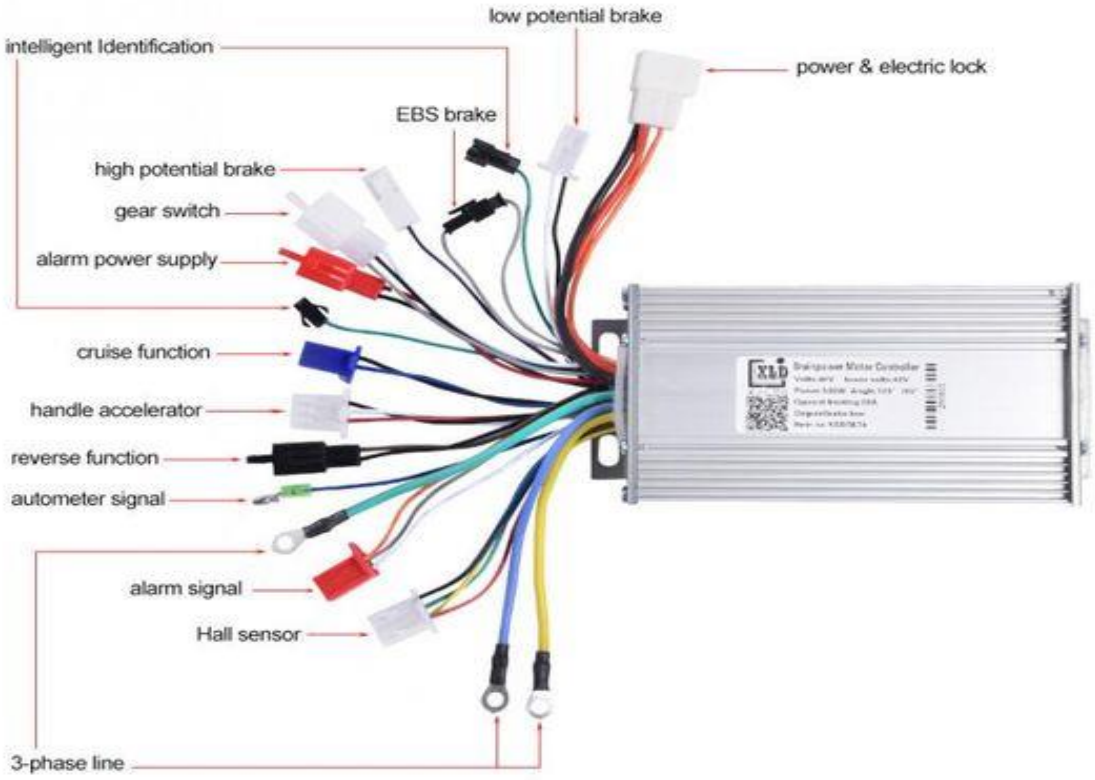


Figure 9 Motor controller cable diagram

Figure 10 Motor controller used

Visualization of the "Reverse" mode using an oscilloscope:

1. When the intelligent identification is triggered – Rising edge on speed steering changes the motor direction (Yellow line).

2. When the intelligent identification is disconnected – DAC connected to the speed steering input changes the speed of motor (blue line).

** PWM signal does not work to control the motor speed.

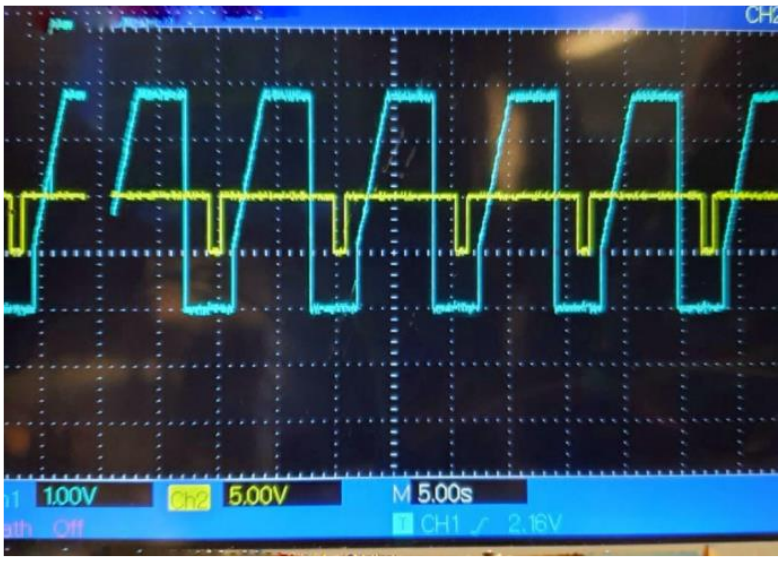


Figure 11 Visual output seen on oscilloscope

**

1. Rising edge detected on "intelligent sensor" (relay) line

2. Input given to speed control line in reverse direction by the DAC.

Toggling of REVERSE using "intelligent Identification" (Code):

```
dac_setvalue = 0;
dac.setVoltage(dac_setvalue , false);
delay(2000);
digitalWrite(reverseLine,LOW);
delay(1000);
digitalWrite(reverseLine,HIGH);
delay (1000);
```

Toggling of BRAKES using "Low potential brake line" (Code):

```
if (i == 0)
  digitalWrite(brakepin, LOW);
else if (i == 1)
{
  digitalWrite(brakepin, HIGH);
  dac_setvalue = 1000;
}
```

1.5 Wheel selection:

Motor hub wheels

These wheels were selected since they are inexpensive and fit inside our budget. It was purchased at www.acjdd.com, and the specifications are listed below. Aside from this, there is no datasheet accessible for this China-purchased device.

Dimensions: 6 inches Shaft type: -double shaft

-2 driven wheels

-Assume 2 pairs of caster wheels as support. The design is for the worst-case scenario, meaning that the supporting caster wheels are not in contact with the floor and the drive wheels are bearing all of the weight.

-floor friction: $\mu = 0.4$

-driven wheel diameter d =152.4 mm, we will also look through a range of diameters {d} {range} =100 mm, 101 mm (the 152.4 mm is compared to wheels used by hoverboards. Our logic is that the hoverboards carry similar weights to our bot)

A differential drive is a very simple driving mechanism that is quite often used in practice, the same is implemented for AGV. The AGV has one more castor wheel to support the vehicle and prevent tilting. Both main wheels are placed on a common axis.

1.6 MCP4725 12-Bit DAC

A digital-to-analog converter (DAC, D/A, D2A, or D-to-A) is a system that converts a digital signal into an analog signal.

A DAC is required as the motor controller used only accepts speed control in the form of an analog signal and not via a PWM signal.

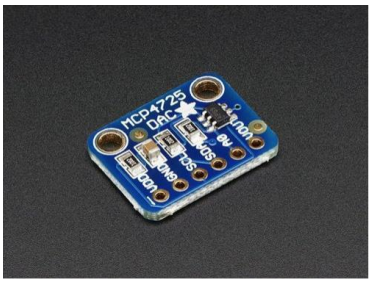


Figure 12 Analog to Digital converter

PIN Connections:

The VDD (power) is initially connected to a 3-5V power supply, and GND to ground.

The DAC uses I2C, a two-pin interface that can have up to 127 unique sensors attached (each must have a different ADDRESS).

SDA to I2C Data (on the Nano, this is A4 on the Mega it is 20 and, on the Leonardo, digital 2).

SCL to I2C Clock (on the Nano, this is A5 on the Mega it is 21 and, on the Leonardo, digital 3).

A0 allows you to change the I2C address. By default (nothing attached to A0) the address is hex 0x62. If A0 is connected to VDD, the address is 0x63. This lets you have two DAC boards connected to the same SDA/SCL I2C bus pins.

VOUT is the voltage out from the DAC. The voltage will range from 0V (when the DAC value is 0) to VDD (when the DAC 'value' is the max 12-bit number: 0xFFF).

IDE Setup:

Next up, download the Adafruit MCP4725 library. This library does all of the interfacing, so you can just "set and forget" the DAC output.

Check that the Adafruit_MCP4725 folder contains Adafruit_MCP4725.cpp and Adafruit_MCP4725.h.

Place the Adafruit_MCP4725 library folder your sketchbookfolder/libraries/ folder.

Restart the IDE.

CODE:

Library initialization:

```
#include <Adafruit_MCP4725.h>

Adafruit_MCP4725 dac;
```

DAC Setup:

```
// For Adafruit MCP4725A1 the address is 0x62 (default) or 0x63 (ADDR pin tied to VCC)
// For MCP4725A0 the address is 0x60 or 0x61
// For MCP4725A2 the address is 0x64 or 0x65
dac.begin(0x62);
digitalWrite(10, HIGH); //release brakes
```

Setting Analog values:

```
dac.setVoltage(i, false);
```

Here, i is the 12-bit value for the corresponding analog voltage output required from the DAC.

1.7 Optocoupler

An opto-isolator (also called an optocoupler, photocoupler, or optical isolator) is an electronic component that transfers electrical signals between two isolated circuits by using light. Opto-isolators prevent high voltages from affecting the system receiving the signal.

When a high voltage is used for one component and a smaller voltage is required for the rest of the system, an optocoupler is generally used to isolate the two parts of the system requiring varying voltage levels.

Commercially available opto-isolators withstand input-to-output voltages up to 10 kV and voltage transients with speeds up to 25 kV/µs.

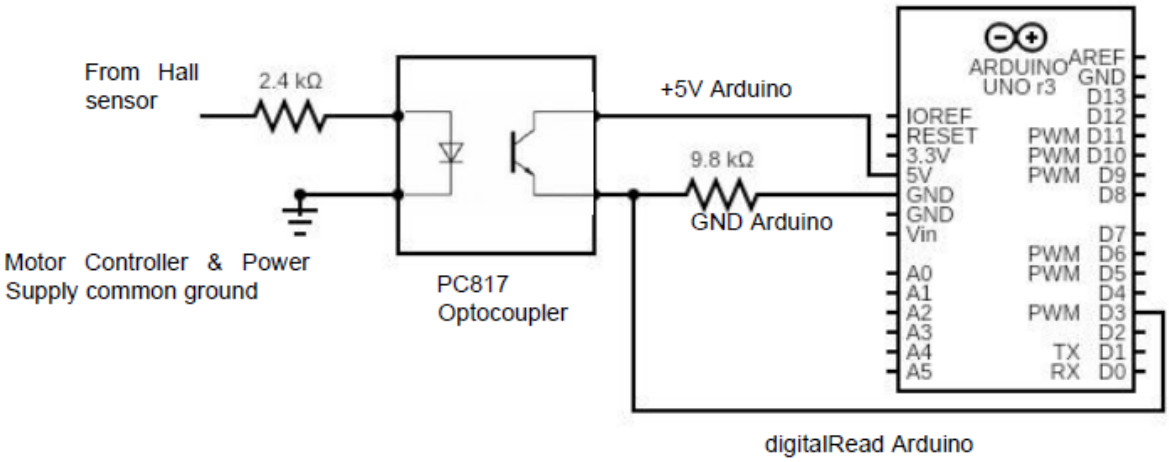


Figure 13 Optocoupler to Arduino Uno connection diagram

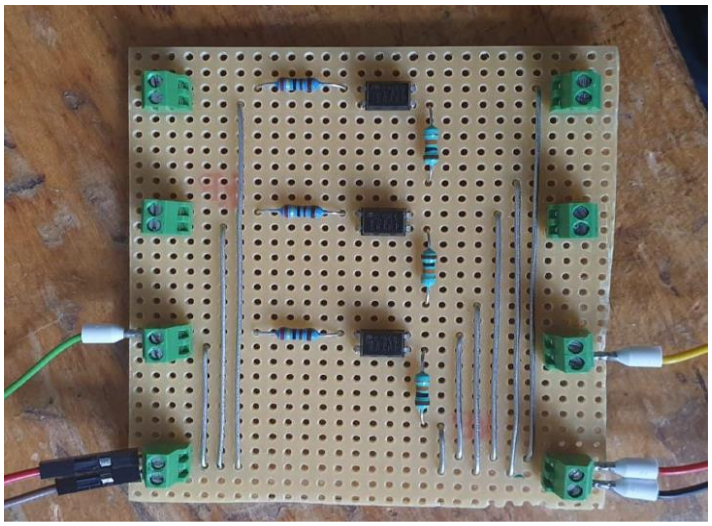


Figure 14 Optocoupler circuit

An optocoupler is used to detect the signal from these hall sensors at it is the least intrusive method of signal detection and does not affect the hall sensor signal to the motor controller in any way.

The opto-coupler grounding on the hall sensor side is grounded along with the motor in order to keep the hall sensors reading un-interfered by the noise on the Arduino side.

1.8 Component selection

1.8.1 Types of sensors

IR sensor:

Here, for the task of Step and object detection to avoid obstacles, IR sensors are used.

The two following models were used:

Sharp Distance Sensor GP2Y0A21YK0F: It a distance measuring sensor unit, composed of an integrated combination of PSD (position sensitive detector), IRED (infrared emitting diode) and signal processing circuit. The range for this sensor is 10 to 80 cm and it is being used in AGV for Step Detection.

Figure 15 GP2Y0A21YK0F IR-Sensor

Sharp Distance Sensor GP2Y0A02YK0F: is a distance measuring sensor unit, composed of an integrated combination of PSD (position sensitive detector), IRED (infrared emitting diode) and signal processing circuit. The range for this sensor is 20 to 150 cm and it is being used in AGV for object detection.

Figure 16 GP2Y0A02YK0F IR-Sensor

Connections:

For both IR sensors being used, the connection is similar:

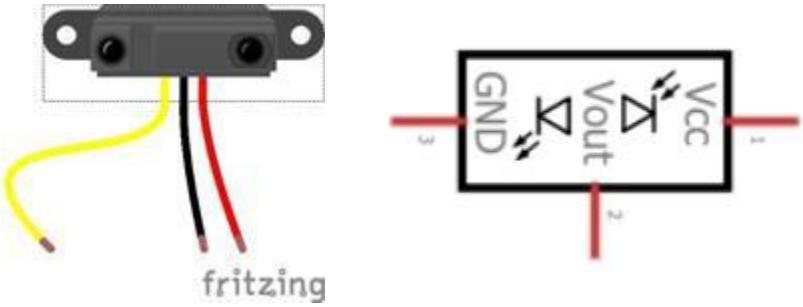


Figure 17 IR-Sensor schematic and connection

1.8.2 Voltage regulator:

The IO PCB board has a TIOL1115 which has an integrated linear voltage regulator (LDO) and can only deliver 20mA. This is not enough since many parts need more than this. For example, one IR sensor needs 30mA itself. For solving the issue, DDR-60G-24 DC/DC converter. This DC-DC convertor is used in the project to allow us to convert 24V to the respective 5,12,24V. This output is a constant output, hence if the battery level goes low from the desired 24V, it will still give us the mentioned output. The modules are easy to use and can be attached on the DIN rail directly. They are also industry certified hence safe to use.

Figure 18 DC-DC Convertor

1.8.3 Limit switch:

It is used for bumper system where due to some reason the IR sensor does not work and AGV is still in motion, and hits onto some obstacle, these limit switches will be activated, and it will help stop the AGV immediately.

A limit switch is an electromechanical device that contains an actuator linked to a series of contacts. When an object meets the actuator, the limit switch triggers the contacts to either form or break an electrical connection (Opener in our case).

Figure 19 Limit Switch

1.8.4 E-Stop Switch XW Series:

The XW series of Emergency Stop switches include a revolutionary new technology called “safe break action”. This innovative concept provides greater levels of human safety and is the first of its kind in the world.

Conventional E-Stop switches are designed with spring pressure on the Normally Closed (NC) contacts, keeping them in the closed position and allowing the machine to operate. Improper installation or excessive force to the stop button in an emergency may break or dislodge a vital part, causing the spring-loaded contact to stay closed. This situation renders the E-Stop incapable of stopping the machine and can lead to catastrophic events, personal injury, and possible loss of life.

This one-of-a-kind “safe break action” design, reverses the energy direction and uses the spring-pressure to assure that the NC contacts will open if the emergency switch is damaged, or the

contact blocks separate due to excessive force. The NC contacts will reliably open, even if they are welded, and stop the machine.

XW series switches offer up to four “safe-break” contacts with a depth behind the panel that is half the size of conventional E-Stop switches. This means that there is an additional contact available, and the switches can be used in Level 4 safety category applications.

The XW E-Stop switches are secured from the rear of the control panel so that the E-Stop cannot be removed from the front. Another unique feature of the XW E-Stop switches is that either a push-turn or push-pull reset method can be used to reset the switches. This eliminates any possible confusion for operators when resetting the switch. The durability and quality of these new E-Stop switches make them extremely reliable. They can withstand the highest stress caused by panic or a reaction to an emergency.

Figure 20 Power Switch

1.8.5 Fuses

A fuse is an electrical safety device that operates to provide overcurrent protection of an electrical circuit. Its essential component is a metal wire or strip that melts when too much current flows through it, thereby interrupting the current. It is a sacrificial device; once a fuse has operated it is an open circuit, and it must be replaced or rewired, depending on type.

The fuses used are general automotive fuses also known as blade type fuses. They have a plastic body and two prongs that fit into sockets. Each fuse has the rated current in amperes printed on the top.

Figure 21 Fuse

1.8.6 Battery:

Part Number	UL7-12
Length	151mm
width	65mm
Height with terminal	99mm
Weight	2.05kg
Normal Voltage	12V
Normal Capacity	7AH
Standard Terminal	F1
Container Material	ABS
Max Discharge Current	105A
Internal Resistance	23mΩ
Design Floating Life at 20°C	5 Years

Table 3 List of battery properties

Figure 22 Battery

1.8.7 Accelerometer:

The InvenSense MPU6050 module is a Micro Electro-Mechanical Systems (MEMS) which consists of a 3-axis Accelerometer and 3-axis Gyroscope inside it. This helps us to measure acceleration, velocity, orientation, displacement and many other motion-related parameter of a system or object.

It is very accurate, as it contains 16-bits analog to digital conversion hardware for each channel. Therefore, it captures the x, y, and z channel at the same time. The sensor uses the I2C-bus to interface with the Arduino.

Figure 23 Accelerometer

1.8.8 Programmable Logic Controller

PLC Modules provided by WAGO Kontakttechnik GmbH were used as the main controller of the AGV. This includes an industrial controller, an analog I/O Module, a digital I/O Module and most importantly, an I/O Master Module.

Figure 24 WAGO PLC Module

750-8206 WAGO Controller

The controller 750-8206(PFC200 CS 2ETH RS CAN DPS) is an automation device that can perform control tasks of a PLC. This controller can be used for applications in mechanical and systems engineering, in the processing industry and in building technology. In the AGV all logic will run through this controller.

750-657 I/O Master:

IO link Technology: IO-Link defines a communication standard (acc. IEC 61131-9) for connecting both current digital inputs/outputs and intelligent IO-Link devices to the control level. IO-Link devices describe sensors and actuators of the field level that have IO-Link functionality.

The IO-Link technology is predominantly used in the industrial sector of manufacturing automation. With IO-Link, configuration, diagnostics, and maintenance from the control are possible to the lowest field level in addition to process data communication.

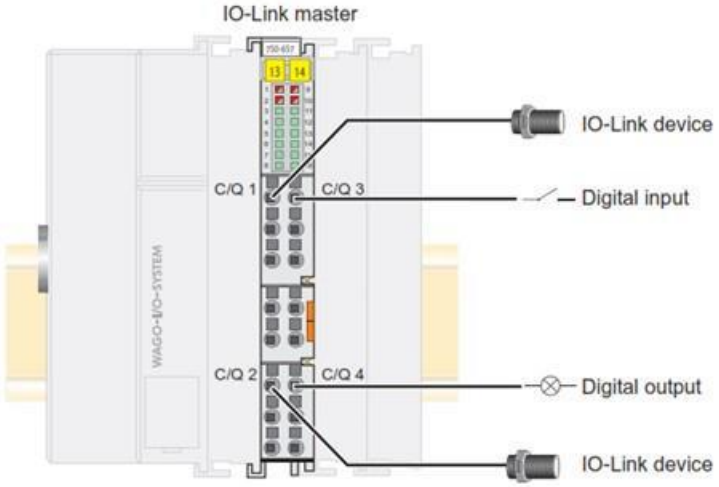


Figure 25 Connection example: IO-Link master with various devices

Device Description:

The IO-Link Master 750-657 module is used to connect intelligent actuators/sensors that the IO-Link communication system uses on the control level.

In addition to the two supply lines L+ and L-, there is another connection (C/Q) used to transfer process, configuration, parameterization, and diagnostic data. The process data width of an IO-Link device can be between one bit and 32 bytes.

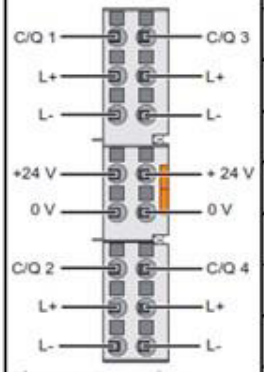


Figure 6: Connections

Termination	Channel	Designation	Function
1	1	C/Q 1	IO-Link Port 1
2	---	L+	Power supply DC 24 V
3	---	L-	Power supply 0 V
4	---	+ 24 V	Field supply 24 V
5	---	0 V	Field supply 0 V
6	2	C/Q 2	IO-Link Port 2
7	---	L+	Power supply DC 24 V
8	---	L-	Power supply 0 V
9	3	C/Q 3	IO-Link Port 3
10	---	L+	Power supply DC 24 V
11	---	L-	Power supply 0 V
12	---	+ 24 V	Field supply 24 V
13	---	0 V	Field supply 0 V
14	4	C/Q 4	IO-Link Port 4
15	---	L+	Power supply DC 24 V
16	---	L-	Power supply 0 V

Figure 26 IO-Link Master 750-657 connections

IO-Link master can handle 24 bytes as followed:

Byte / Offset	0	1	2	3	Offset = 4	Offset = 7	Offset = 9	Offset = 14	21	22	23
Content	Control byte / status byte	Mailbox / Reg.ister comm.	Mailbox / Register comm.	SIO byte	Segment Port 1 Segment Port 1 Segment Port 1	Segment Port 2 Segment Port 2	Segment Port 3 Segment Port 3 Segment Port 3 Segment Port 3 Segment Port 3	Segment Port 4 Segment Port 4 Segment Port 4 Segment Port 4 Segment Port 4 Segment Port 4 Segment Port 4	Unused, 0x00	Unused, 0x00	Unused, 0x00
Segment length					Size = 3	Size = 2	Size = 5	Size = 7			

Figure 27 Internal bus process data, a segment distribution with a 2-byte Mailbox size

750-1405 16-channel digital input Module:

The digital input module 750-1405 (16DI 24VDC 3.0ms) receives control signals from digital field (e.g., of sensors, transmitters, switches, or proximity switches).

The I/O module has 16 input channels, providing a direct connection to 1-wire sensors. The sensors are connected to the Push-in CAGE CLAMP connectors DI 1 ... DI 16.

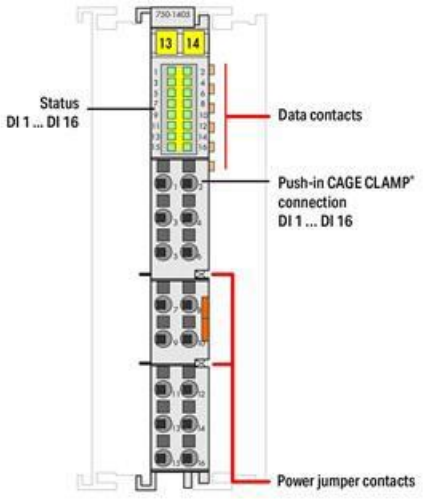


Figure 28 750-1405 16-channel digital input

750-530 8-channel digital output Module:

The 750-530 (8DO 24V DC 0.5A) Digital Output Module transmits binary control signals from the automation device to the connected actuators (e.g., solenoid valves, contactors, transmitters, relays, or other electrical loads).

The module has eight output channels. Eight 1-conductor actuators may be directly connected to the module. The actuators can be connected using connection DO 1 ... DO 8.

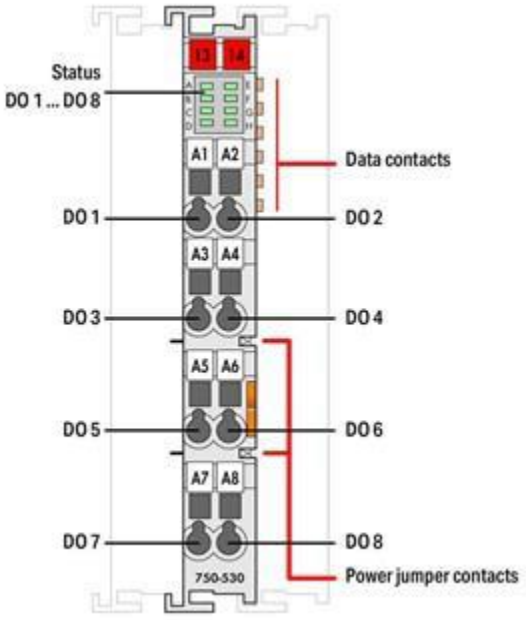


Figure 29 750-530 8-channel digital output

750-600 End Module:

The module 750-600 (End Module) is used to terminate the internal bus of a fieldbus node. This module completes the internal data circuit and ensures correct data flow.

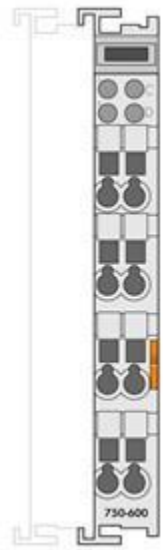


Figure 30 750-600 End Module

WAGO e!COCKPIT Software:

Software being used for programming the PLC is e!COCKPIT. It expedites machine and system startup, while reducing development times for automation projects.

1.9 Communication network

In the following part of the documentation, the interfaces and bus systems required for communication between the different modules are described.

The modules of the robot are the motors or motor controllers, the various sensors for detecting the environment, an Odroid (i.e., a Single Board Computer) on which the Robot Operating System (ROS) runs and is responsible for controlling the Mover along with a Wago PLC.

1.9.1 CAN-Concept

In the following, the concept for the communication between the individual modules based on the CAN bus is presented.

CAN-Bus Topology

The CAN topology is a so-called linear bus, to which the control units are connected via spur lines up to 30 cm long. The bus should be terminated at each of the two line ends with a resistor of the same size as the line surge impedance, which is typically 120 Ohm.

With the help of the CAN bus, a bit rate of at most 1 Mbit/s can be achieved, whereby the bit rate decreases as the length of the line increases. In addition, all ECUs connected to the same CAN bus must communicate at the same bit rate.

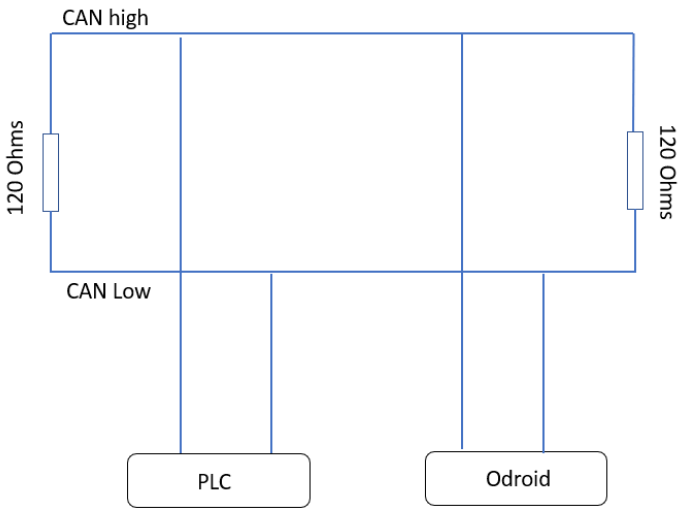


Figure 31 CAN Communication

Physical layer

This section describes the physical layer of CAN buses according to the OSI reference model for network protocols (also known as the ISO/OSI-7-layer model).

A cable for the CAN bus system usually consists of two twisted Lines which are designated CAN High and CAN Low. In addition, there can be one wire each as ground line and power supply line.

The CAN bus can be divided into the so-called high-speed CAN (ISO 11898-2) and the low-speed CAN (ISO 118983), where the low-speed CAN is used for data transfer rates of up to 125 kbit/s. In addition, the low-speed CAN bus may not be used due to lower data transmission rates corresponding to longer lines.

Furthermore, any spur lines may also be used for longer periods of time than 30 cm.

In a CAN bus system, all connected control units have equal rights. To avoid collisions, i.e., simultaneous access by several control units, the so-called CSMA/CR (Carrier Sense Multiple Access/Collision Resolution) method is used.

The priority of a CAN message is defined by the so-called Message Identifier, whereby a logical 0 is interpreted as dominant and a logical 1 as recessive. As a result, the message with the smaller number is transmitted, whereas the ECU which transmitted the message with the higher value switches from transmit to receive. Such a Message Identifier has a length of either 11 bit or 29 bits.

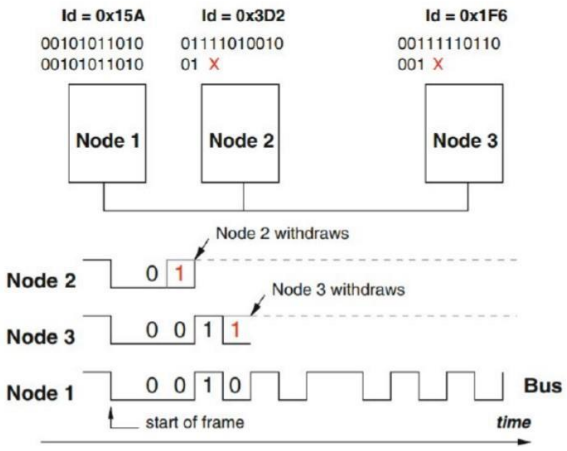


Figure 32 Collision of CAN messages

The figure below shows the voltage levels of both the High-speed CAN and the Low-speed CAN. As can be seen, the voltage difference between the CAN high line (3.5 V) and the CAN low line (1.5 V) on the high-speed CAN is only 2.0 V and corresponds to a logical 0. If there is no voltage difference between the two wires, it is interpreted as logical 1, and the voltage to ground is then 2.5 V in each case.

In the case of a logical 1, the low-speed CAN bus supplies a voltage difference of 5 V, with the CAN high line at 0 V and the CAN low line at 5 V. In the case of a logical 0, however, the difference is only 2.2 V (CAN high signal level = 3.6 V; CAN low signal level = 5 V; level = 1.4 V).

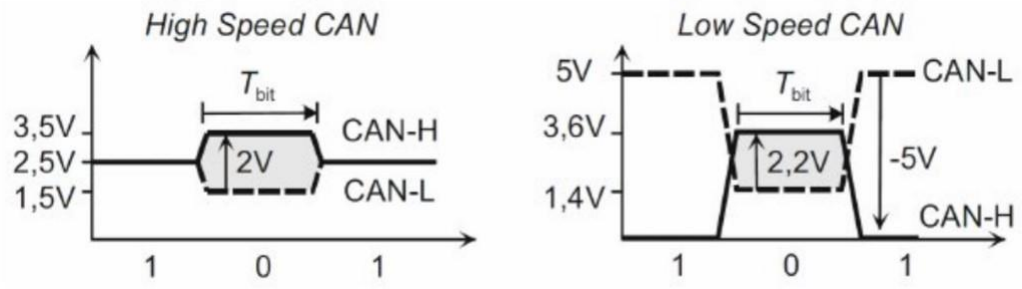


Figure 33 High Speed and Low Speed CAN

Format of CAN messages

The format of a CAN message is shown in the figure below . At the beginning of the message a so-called start bit is set, whereby it is the dominant value, i.e., a zero.

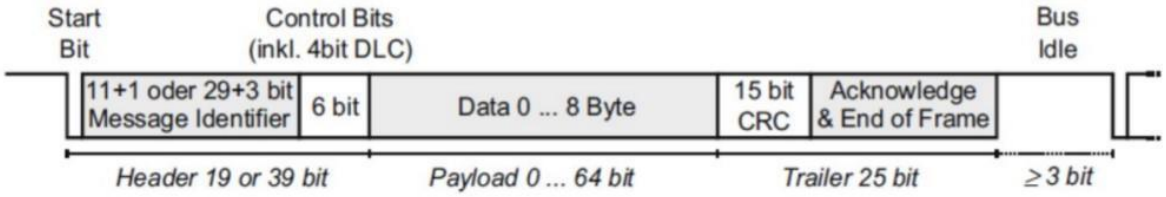


Figure 34 60 Standard CAN Telegram

This is followed by the already mentioned Message Identifier, which is terminated with a length of 29, respectively 11 bit and required for this purpose, to clearly define the message. The identifier is sent to all bus connected control devices, which in turn are then sent to the can recognize from the identifier whether the message is suitable for the respective control unit is relevant or can be ignored.

The so-called arbitration field of a CAN message includes the Message Identifier as one more bit for the so-called Remote Transmission Request (RTR). With the help of the RTR, the control unit to respond to the corresponding CAN message to answer.

In addition, the header of a CAN message also includes a subsequent area with a total length of six bits, which is also referred to as control bits. The control bits include the 4-bit Data Length Code (DLC), which provides information on how much data can be found in the following payload area.

With a single message, a maximum of 8 bytes of user data can be transferred. This area of a message is also called payload.

In addition to the user data, so-called stuffing bits are sent, which serve to re-synchronize the bit clocks of the control units. These are initially synchronized by the start bit, but require resynchronization, which is implemented using signal edges occurring in the messages. The procedure is as described below.

If a message contains the same value five times in succession, i.e., no signal edge, a stuffing bit with the inverse value of the previously sent bits are inserted to create an edge in the message.

The stuffing bits added in this way can enlarge a message by up to 25% in the worst case. The stuffing bits are automatically removed by the recipients of a message.

The payload area is followed by a 15-bit checksum (Cyclic Redundancy Check = CRC), which is determined by the control units. This checksum results from Polynomial Division.

If the polynomial division leaves a remainder that has a different value as zero, the message which must be transmitted most likely has an error.

The CRC bits are followed by the so-called “Acknowledge & End of Frame” area of the CAN message. The ECUs connected to the bus transmit either an acknowledgement of receipt or an “error frame” in this area if, for example, the calculated checksum of the receiver does not match that of the transmitter.

If an error frame is received, the respective message is ignored by all other ECUs, since the data in the CAN network should match the individual nodes. After the error frame has been received, the CAN message is sent by the transmitter to all communication nodes.

Implementation of a CAN bus using the Arduino Microcontroller

For practical implementation of the CAN bus, appropriate CAN controllers & transceivers are required in addition to the cabling. A relatively simple implementation of such a bus system is made possible by using an Arduino microcontroller in combination with a corresponding CAN module.

These CAN modules contain, as shown in the figure below, the CAN controller as well as the transceiver and thus enable the realization of a CAN bus at a relatively low price. Furthermore, appropriate libraries are available for the Arduino, which facilitate the programming of the network.

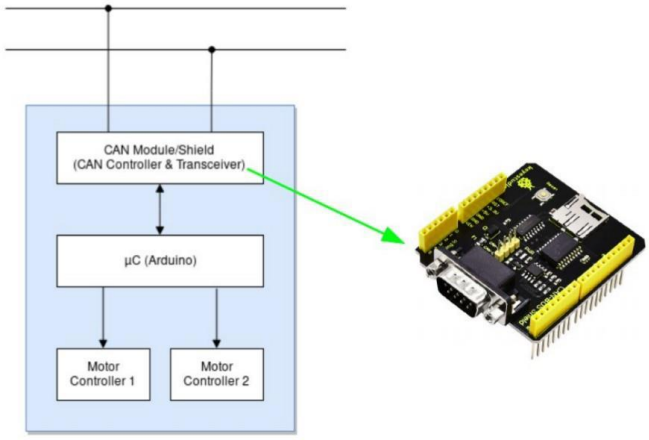


Figure 35 CAN-Bus to Arduino

For the implementation of the CAN bus the MCP2515 CAN-module is used, which interfaces with the Arduino by using the Serial Peripheral Interface (SPI).

Figure 36 MCP 2515 CAN Module

The module makes it possible for the Arduino to send and receive CAN data frames. On the right side there are the connections for the CAN-high and the CAN-low cable which make up the can bus. The board also includes a 120 Ohm terminal resistance (The jumper is named J1 on the module). On the left side are the Pins for the SPI-bus that have to be connected to the Arduino.

Pin	Description
VCC	5V Power input pin
GND	Ground pin
CS	SPI SLAVE select pin (Active low)
SO	SPI master input slave output lead
SI	SPI master output slave input lead
SCLK	SPI Clock pin
INT	MCP2515 interrupt pin

Figure 37 CAN Module Pin Description

Implementing CAN on Odroid

In the following the necessary steps to enable the CAN communication are described. Just like on the Arduino the SPI interface is used for the CAN communication.

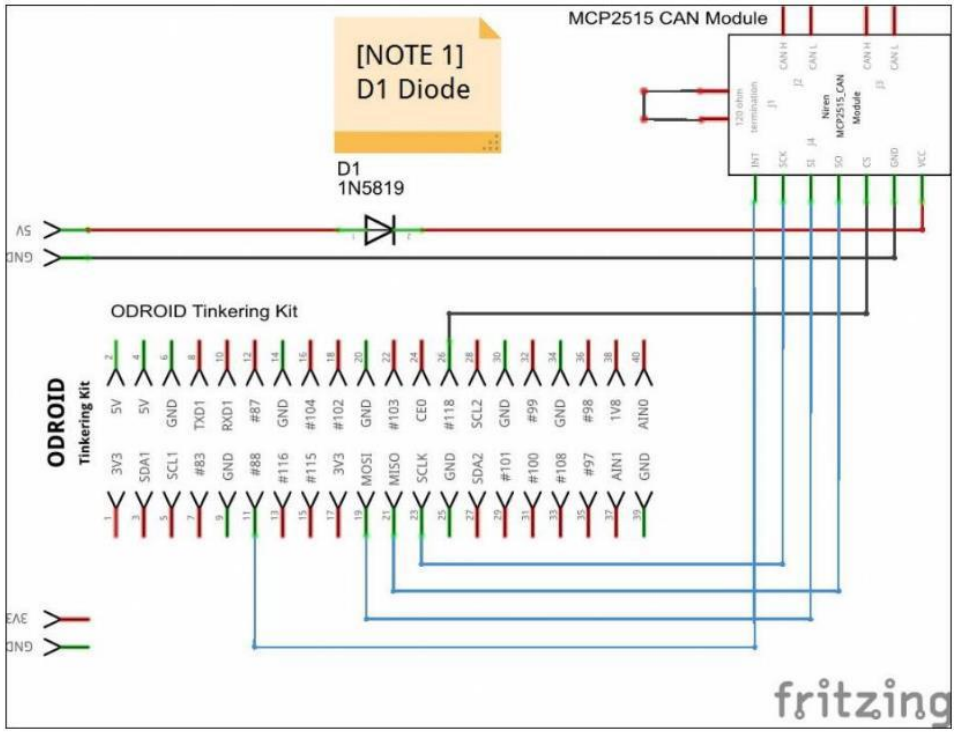


Figure 38 Connection diagram Odroid to MCP2515

The diode on the 5V supply line is not necessary with the Odroid N2+.

S/W installation

Updating kernel is highly recommended

```
sudo su
apt update && apt full-upgrade
```

Add the text “can0”-can(zero)- on the Device Tree Overlay section (line 94) in your file located at /media/boot/config.ini (only accessibly over the root directory).

```
cd /media/boot
gedit config.ini
```

```
; Device Tree Overlay
overlay_resize=16384
overlay_profile=
overlays="spi0 i2c0 i2c1 uart0 can0"

[overlay_custom]
overlays="i2c0 i2c1"

[overlay_hktft32]
overlays="hktft32"

[overlay_hktft35]
overlays="hktft35"
```

.ini Tab Width: 8 Ln 94, Col 36 INS

Now we build the Linux Kernel to make the required changes to it.

Refer the following website.

https://wiki.odroid.com/odroid-n2/software/building_kernel

Native Compile - ODROID-N2/Ubuntu

Note: 8GB eMMC/SD card have not enough space to build kernel source. In order to do native compile, the 5GB of free space is required at least. Furthermore, building the kernel takes around 40 minutes, do not leave the Odroid unattended.

Installing required packages

```
odroid@odroid64:~$ sudo apt update
odroid@odroid64:~$ git clone --depth 1 https://github.com/hardkernel/linux.git -b odroidg12-4.9.y
odroid@odroid64:~$ cd linux
```

Compile & Installation

```
odroid@odroid64:~/linux$ make odroidg12_defconfig
```

Edit the .config file or run "make menuconfig" to add/remove Kernel drivers.

```
odroid@odroid64:~/linux$ make -j4
odroid@odroid64:~/linux$ sudo make modules_install
odroid@odroid64:~/linux$ sudo sync
odroid@odroid64:~/linux$ sudo reboot
```

Changes to the Kernel

Now the necessary changes are made to the Linux Kernel and the changed files are recompiled. First, we set the SPI frequency in the MCP config file to 8MHz since we use a MCP module with an 8 MHz oscillator.

```
cd linux/drivers/net/can/spi
gedit mcp251x.c
```

Now this section of the Kernel needs to be recompiled.

```
cd ~/linux
sudo su
make -j4 modules && make modules_install
```

Now, also change the frequency in the can0.dts file. This file also defines the used pins for the interrupt (gpio-irq) and chip select (reg).

Use the following link for reference.

https://forum.odroid.com/viewtopic.php?t=40570

```
cd linux/arch/arm64/boot/dts/amlogic/overlays/odroidn2
gedit can0.dts
```

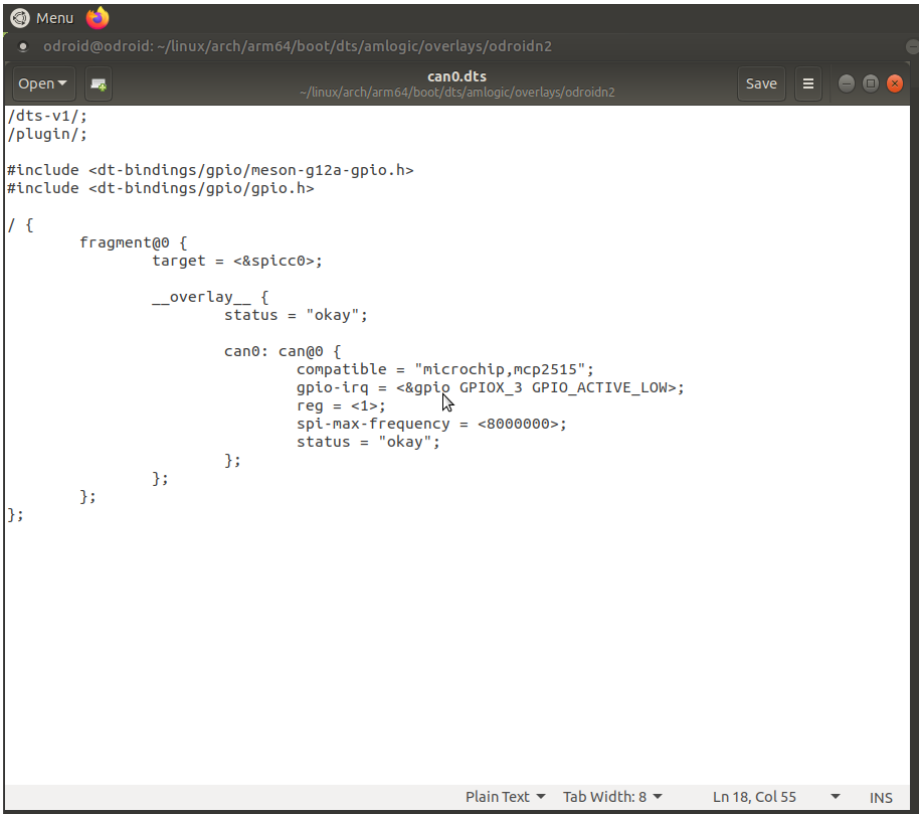


Figure 39 can0.dts

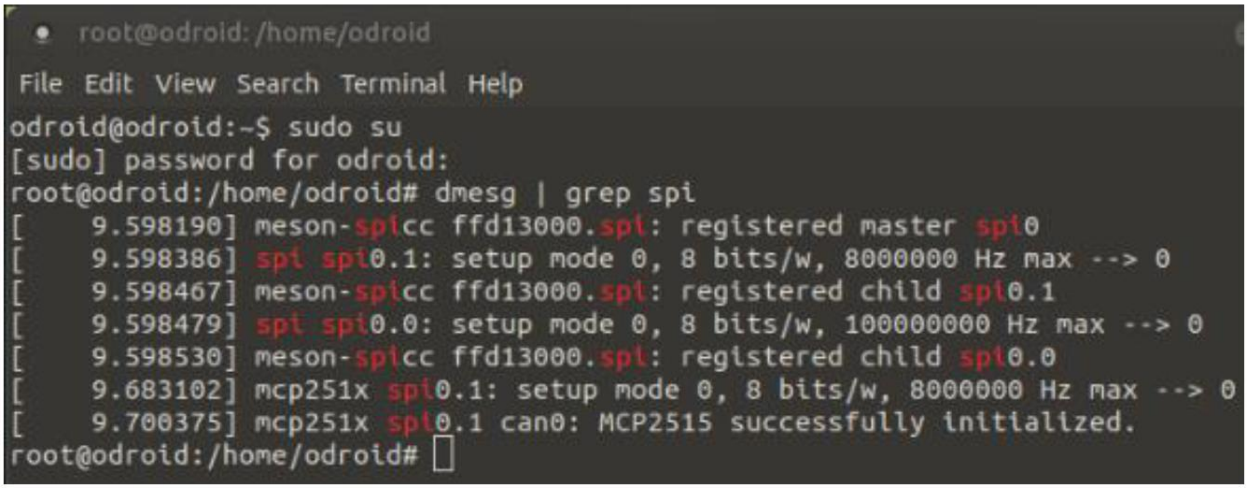
Now compile the device tree overlay file "can0.dtbo" and replace the original file in your memory card with the new one.

```
cd linux
make odroidg12_defconfig
make dtbs

sudo cp -f arch/arm64/boot/dts/amlogic/overlays/odroidn2/can0.dtbo
/media/boot/amlogic/overlays/odroidn2/can0.dtbo
sudo sync
sudo reboot
```

Now, start the Odroid with the MCP connected, otherwise the MCP might not initialize successfully.

```
sudo su
dmesg | grep spi
```




```
lsmod | grep spi
lsmod | grep mcp251x
```

Verifying CAN support configuration

Verify the CAN host driver is registered correctly.

```
ls /sys/class/net/
ifconfig can0
```

```
root@odroid:/home/odroid# ls /sys/class/net/
can0  eth0  lo  wlan0
root@odroid:/home/odroid# ifconfig can0
can0: flags=128<NOARP>  mtu 16
        unspec 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00  txqueuelen 10  (UNSPEC)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@odroid:/home/odroid#
```

Set the bitrate before all operations Example: Set the bitrate of the can0 interface to 125kbps:

```
ip link set can0 type can bitrate 125000 triple-sampling on
ifconfig can0 up
ifconfig
```

```
root@odroid:/home/odroid# ip link set can0 type can bitrate 125000 triple-sampling on
root@odroid:/home/odroid# ifconfig can0 up
root@odroid:/home/odroid# ifconfig
can0: flags=193<UP,RUNNING,NOARP>  mtu 16
        unspec 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00  txqueuelen 10  (UNSPEC)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

eth0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        ether 00:1e:06:42:71:c7  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
        device interrupt 22

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1  (Local Loopback)
        RX packets 604  bytes 51779 (51.7 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 604  bytes 51779 (51.7 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.178.66  netmask 255.255.255.0  broadcast 192.168.178.255
        inet6 2001:4dd5:aec9:0:5dc0:418e:c22e:6fe0  prefixlen 64  scopeid 0x0<global>
        inet6 fe80::d864:9471:75e1:4d43  prefixlen 64  scopeid 0x20<link>
        inet6 2001:4dd5:aec9:0:6c54:43e7:b6ed:9f97  prefixlen 64  scopeid 0x0<global>
        ether d0:37:45:18:54:75  txqueuelen 1000  (Ethernet)
        RX packets 1135  bytes 292102 (292.1 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 933  bytes 149308 (149.3 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@odroid:/home/odroid#
```

Installing SocketCAN utils

CAN-utils package is a collection of CAN drivers and networking tools for Linux. It allows interfacing with CAN bus devices in a similar fashion as other network device.

```
sudo apt install can-utils
```

Loopback test on a single CAN port

```
ifconfig can0 down
ip link set can0 type can bitrate 125000 loopback on
ifconfig can0 up
ip -details link show can0
```

```
root@odroid:/home/odroid# ifconfig can0 down
root@odroid:/home/odroid# ip link set can0 type can bitrate 125000 loopback on
root@odroid:/home/odroid# ifconfig can0 up
root@odroid:/home/odroid#
root@odroid:/home/odroid# ip -details link show can0
3: can0: <NOARP,UP,LOWER_UP,ECHO> mtu 16 qdisc fq_codel state UP mode DEFAULT group default qlen 10
    link/can  promiscuity 0
    can <LOOPBACK,TRIPLE-SAMPLING> state ERROR-ACTIVE restart-ms 0
          bitrate 125000 sample-point 0.875
          tq 500 prop-seg 6 phase-seg1 7 phase-seg2 2 sjw 1
          mcp251x: tseg1 3..16 tseg2 2..8 sjw 1..4 brp 1..64 brp-inc 1
          clock 4000000numtxqueues 1 numrxqueues 1 gso_max_size 65536 gso_max_segs 65535
root@odroid:/home/odroid#
```

The following command shows the received message from the CAN bus.

```
candump can0
```

On a second terminal, the following command sends 3 bytes on the bus (0x11, 0x22, 0x33) with the identifier 500.

```
cansend can0 500#11.22.33
```

```
odroid@odroid: ~
File  Edit  View  Search  Terminal  Help
odroid@odroid:~$ cansend can0 500#11.22.33
odroid@odroid:~$ cansend can0 500#11.22.44
odroid@odroid:~$ cansend can0 500#11.22.55
odroid@odroid:~$
```

```
odroid@odroid:~$ candump can0
  can0  500   [3]  11 22 33
  can0  500   [3]  11 22 33
  can0  500   [3]  11 22 44
  can0  500   [3]  11 22 44
  can0  500   [3]  11 22 55
  can0  500   [3]  11 22 55
```

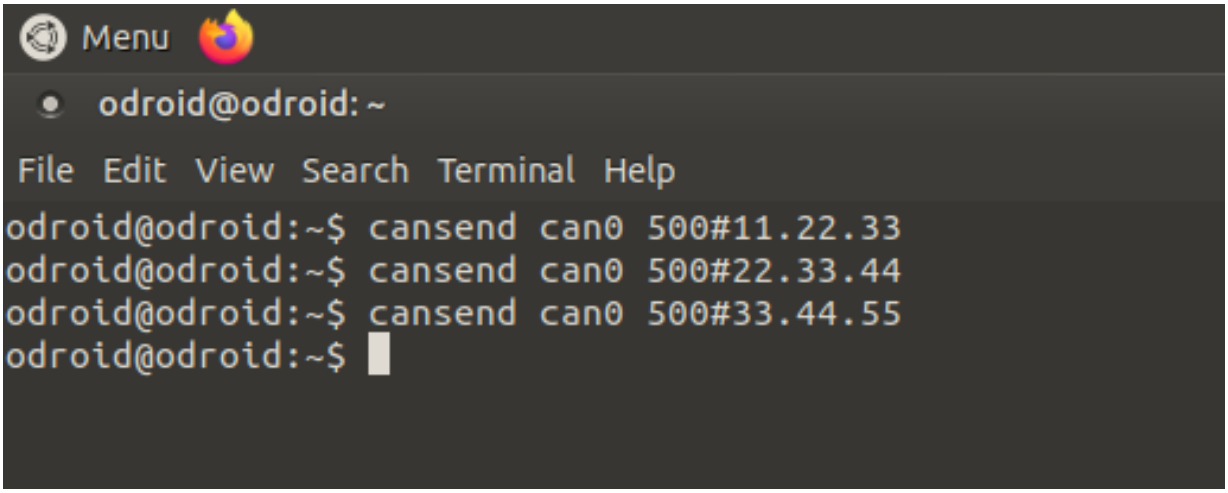
CAN communication between Odroid and Arduino

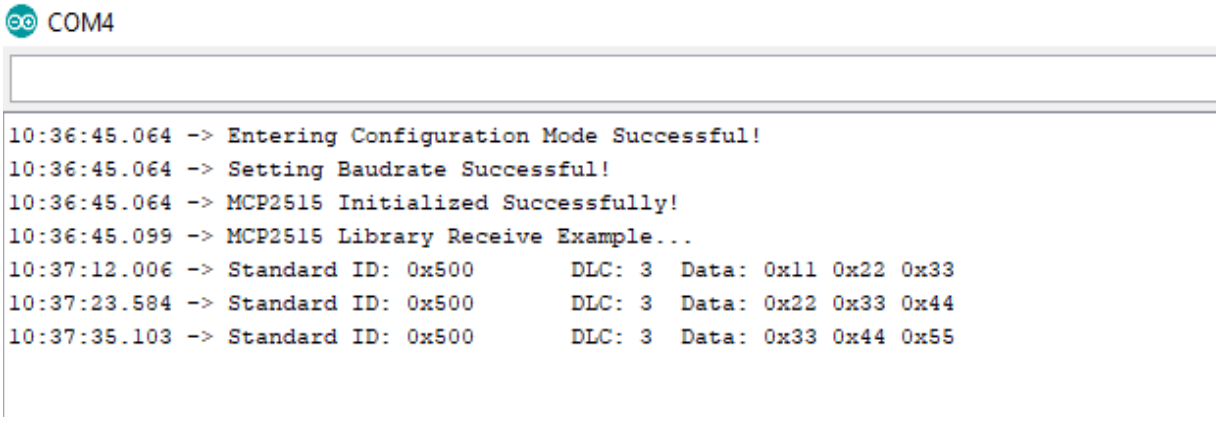
Now a CAN message is sent from the Odroid to an Arduino acting as a receiver.

After reboot do the following,

```
sudo su
dmesg | grep spi
ifconfig can0 down
ip link set can0 type can bitrate 125000 triple-sampling on
ifconfig can0 up


cansend can0 500#11.22.33
```





1.9.2 IO-Link

IO-Link is a short distance, bi-directional, digital, point-to-point, wired (or wireless), industrial communications networking standard (IEC 61131-9) used for connecting digital sensors and actuators to either a type of industrial fieldbus or a type of industrial Ethernet. Its goal is to provide a technology platform that allows for the development and usage of sensors and

actuators that can produce and consume enhanced sets of data, which can then be utilized to optimize industrial automated processes and operations at a lower cost.

An IO-Link system consists of an IO-Link master and one or more IO-Link devices, i.e., Sensors or Actuators. The IO-Link master provides the interface to the higher-level controller (PLC) and controls the communication with the connected IO-Link devices.

An IO-Link master can have one or more IO-Link ports to which only one device can be connected at a time. This can also be a "hub" which, as a concentrator, enables the connection of classic switching sensors and actuators.

An IO-Link device can be an intelligent sensor, actuator, hub or, due to bidirectional communication, also a mechatronic component, e.g., a gripper or a power supply unit with IO-Link connection. Intelligent with regard to IO-Link means that a device has identification data e.g., a type of designation and a serial number or parameter data (e.g., sensitivities, switching delays or characteristic curves) that can be read or written via the IO-Link protocol. This allows parameters to be changed by the PLC during operation, for example. Intelligent also means, however, that it can provide detailed diagnostic information. IO-Link and the data transmitted with it are often used for preventive maintenance and servicing, e.g., it is possible to set an optical sensor in such a way that it reports via IO-Link in good time if it threatens to become dirty. Cleaning no longer comes as a surprise and blocks production; it can now be put on a production break.

The parameters of the sensors and actuators are device- and technology-specific, which is why parameter information in the form of an IODD (IO Device Description) with the description language XML. The IO-Link community provides interfaces to an "IODD Finder", which can be used by engineering or master tools to present the appropriate IODD for a device.

The aim of IO Link was to develop a universal communication interface that transmitted analog and digital signals, and to exchange device parameters with overlying controllers in a system. An IO-Link system consists of a “master” which retrieves all the available information from the devices connected to it (e.g., sensors). In turn, this information can be linked to a gateway (e.g., PLC) connected to a fieldbus interface.

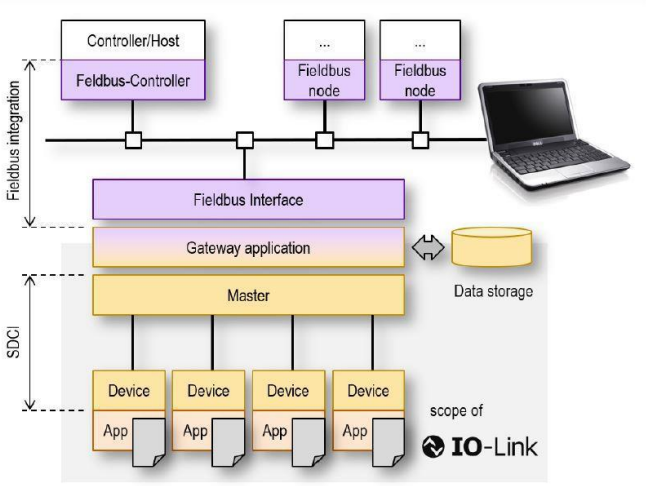


Figure 40 Scope of IO-Link

IO-Link Concept

IO-Link defines a communication standard (acc. IEC 61131-9) for connecting both current digital inputs/outputs and intelligent IO-Link devices to the control level. IO-Link devices describe sensors and actuators of the field level that have IO-Link functionality.

In the scope of the project, IO-Link is used to set up the communication between the PLC and Arduino Nano.

Communication occurs via serial point-to-point links in standard 3-wire technology. Both data and diagnostic information as well as power supply are carried over the serial IO-Link interface simultaneously.

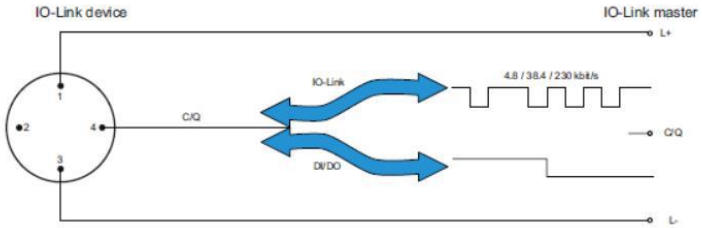


Figure 41 IO Link Communication

To integrate the IO Link devices such as Arduino in the automation systems, the IO-Link devices have to be configured based on standardized description files `IO Device Description` (IODD), which are provided by manufacturer.

The following demonstrates the framework of the IO link system and illustrates the necessary hardware and software components required to build this specific system.

1. IO Link Master connected to the shield through an IO-Link protocol. This Protocol primarily consists of three wire interfaces: - L+, L- and C/Q as shown in the figure.

2. The shield is used as a base plate of Arduino (Nano in our case) which shares a power supply of 5V- 24V as shown.

3. IODD file is used as a description file which is fed to the master through an engineering software or tool.

4. The GUI also provides a header file compatible to the IODD file to the Arduino framework. Preferably the number of bytes and libraries are matched.

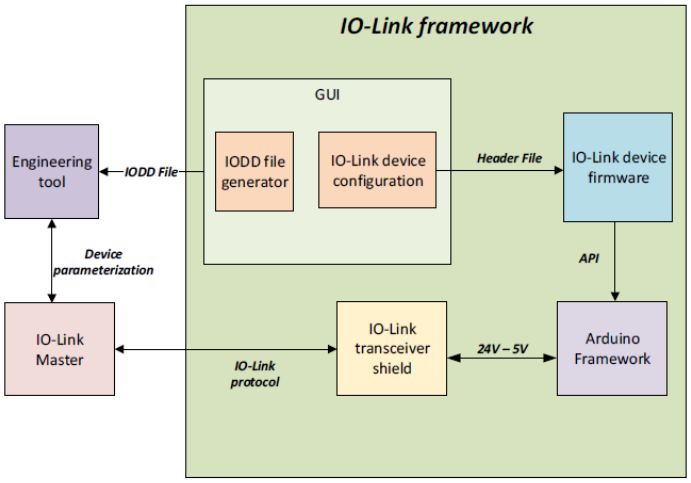


Figure 42 IO-Link Framework

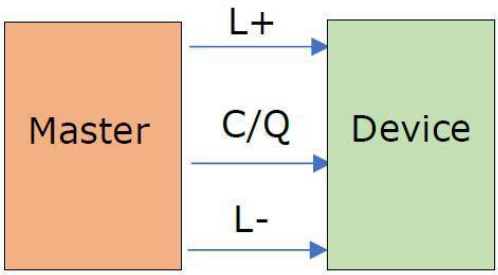


Figure 43 Connection Diagram

Process Image

The size of the process image can be set according to devices attached to the IO-Link ports. Process image sizes of 4, 6, 8, 10, 12, 16, 20 or 24 bytes can be set. The data is exchanged between the master and the device. As mentioned earlier, data sharing works in both ways.

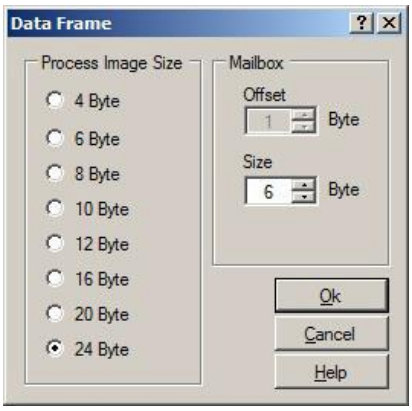


Figure 44 Size of Process Image and Mailbox

Process image						
Pi size	**Input data**			**Output data**		
4 Byte	S0	Status byte		C0	Control byte	
	FC0	Acyclic channel	Register byte 0	FC0	Acyclic channel	Register byte 0
	MB0	Mailbox-Byte	Register byte 1	MB0	Mailbox byte	Register byte 1
	SIO	SIO byte		SIO	SIO byte	
6 Byte	D0	Data byte 0		D0	Data byte 0	
	D1	Data byte 1		D1	Data byte 1	
8 Byte	D2	Data byte 2		D2	Data byte 2	
	D3	Data byte 3		D3	Data byte 3	
10 Byte	D4	Data byte 4		D4	Data byte 4	
	D5	Data byte 5		D5	Data byte 5	
12 Byte	D6	Data byte 6		D6	Data byte 6	
	D7	Data byte 7		D7	Data byte 7	
16 Byte	D8	Data byte 8		D8	Data byte 8	
	D9	Data byte 9		D9	Data byte 9	
	D10	Data byte 10		D10	Data byte 10	
	D11	Data byte 11		D11	Data byte 11	
20 Byte	D12	Data byte 12		D12	Data byte 12	
	D13	Data byte 13		D13	Data byte 13	
	D14	Data byte 14		D14	Data byte 14	
	D15	Data byte 15		D15	Data byte 15	
24 Byte	D16	Data byte 16		D16	Data byte 16	
	D17	Data byte 17		D17	Data byte 17	
	D18	Data byte 18		D18	Data byte 18	
	D19	Data byte 19		D19	Data byte 19	

Figure 45 Process Image

The process image has Status byte, Mailbox Byte and SIO Byte regardless of the configuration as shown in the figure. The size of the mailbox can be varied according to the requirement of the transmission. However, in this project the size has been kept of 2 bytes for mailbox. SIO byte decides whether the port of the IO Link master module is configured as IO-Link, DI (Digital Input) or DO (Digital Output).

IO-Link Communication Setup

To configure the IO-Link master module, the setting tab is selected, where the below mentioned parameters are being set according to the requirement of the system.

Port Configuration

1. The operation mode is selected for each of the port.

2. As mentioned in the section above, IODD file provided by the manufacture contains the information.

3. The information such as Vendor ID, In PD Length etc. is fetch automatically when get data is clicked.

4. The Maximum Cycle time for polling of the connected devices is accordingly set. This is calculation is done by selecting the multiplier and time base as shown in the figure below.

5. The Parameter Server Mode is selected as the last step to configure the port.

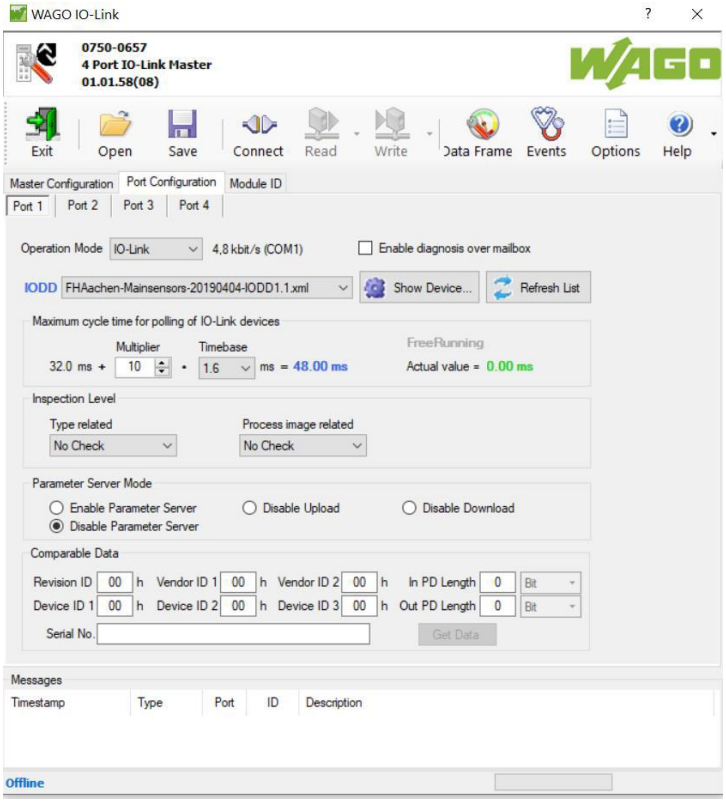


Figure 46 Port Configuration

Master Configuration

The major step in the master configuration setting of the WAGO IO Link Master is the arranging of the upstream and downstream data bus.

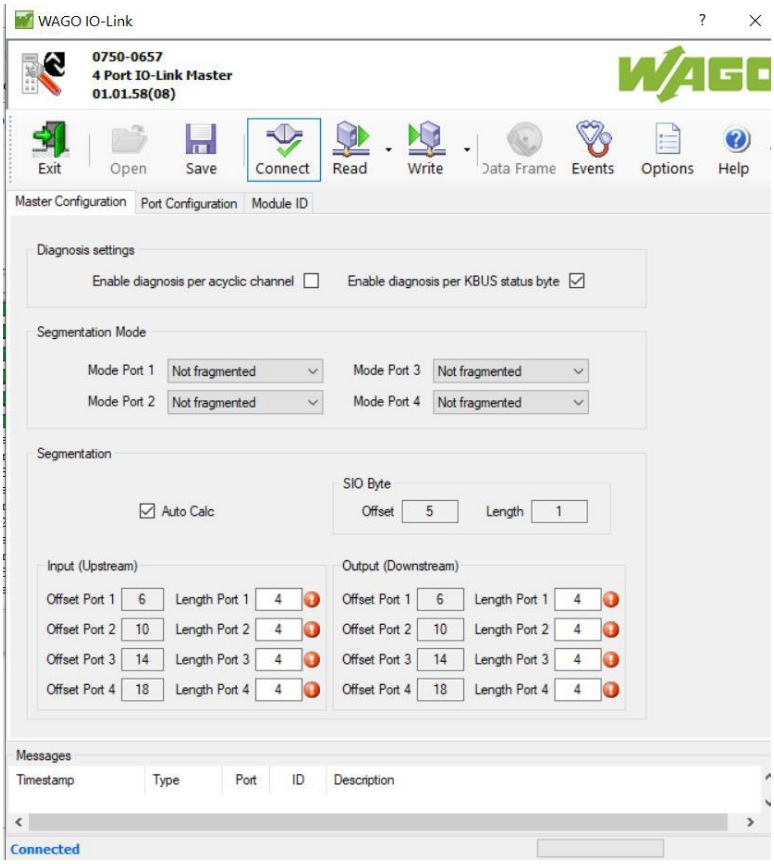


Figure 47 Master Configuration

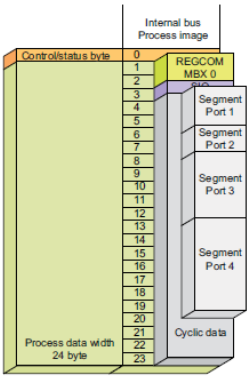


Figure 48 Port width

The process image size of the IO-Link master can be adjusted to the devices (switches, IO-Link compatible devices, etc.) attached to the IO-Link ports. Process image sizes of 4, 6, 8, 10, 12, 16, 20 or 24 bytes can be set. As process data is exchanged, the process image has a fixed structure of a control/status byte, Mailbox bytes and SIO byte regardless of the current configuration. After the SIO byte, another data storage area is available in which cyclical process data can be transferred from IO-Link devices.

1. Control Status Byte

The first byte of the internal bus process image is reserved for the control byte (data direction fieldbus coupler/controller IO-Link master) or status byte (data direction IO-Link master fieldbus coupler/controller). The control byte is used to switch between register communication (see appendix, section "Register Communication" and mailbox communication (see section "Mailbox"). During register communication, the register query response is contained in the status byte followed by 2 bytes of register data. If the mailbox is active, status information of the I/O module and IO-Link ports is displayed cyclically via the status byte.

2. Mailbox

The "Mailbox 2.0" transmission method is hereinafter abbreviated as "Mailbox". If the Mailbox is active (control byte, bit 7 = 0), it is used to transfer configuration, parameterization, and diagnostic data. Byte 1 and byte 2 of the internal bus process image are used by default.

3. SIO byte

"SIO" is an abbreviation for "Standard Input/Output". The SIO byte is used in the IO-Link operating mode in SIO mode (input only), as well as in the "DI" and "DO" operating modes to transfer data to and from digital inputs or outputs to and from the control.

SIO byte							
Bit 7	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
0	0	0	0	SIO3	SIO2	SIO1	SIO0

SIO3	0	Process data of digital port 4: logical '0'
	1	Process data of digital port 4: logical '1'
SIO2	0	Process data of digital port 3: logical '0'
	1	Process data of digital port 3: logical '1'
SIO1	0	Process data of digital port 2: logical '0'
	1	Process data of digital port 2: logical '1'
SIO0	0	Process data of digital port 1: logical '0'
	1	Process data of digital port 1: logical '1'

Figure 49 SIO Byte

4. Cyclic Process Data

The first bytes of the process image are already allocated with the control/status byte, the Mailbox bytes, and the SIO byte. If the selected internal bus data width is more than the set Mailbox size + control/status and SIO byte, another data storage area is available in addition to using the SIO byte in which process data is transferred cyclically from attached IO-Link devices (see the following figure).

Four segments can be configured in this area – one segment for each port available. The segment size is set based on the requirements of the application or attached devices. segmentation provided in the WAGO IO Link tool and the actual device. Hence, we need to select the bytes considering the device and IODD file.

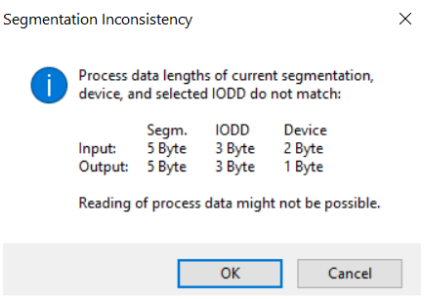


Figure 50 Segmentation inconsistency error

IODD files

Before using the GUI, the user must have done a description of its application. The user must define:

1. Identification Parameters for its device.
2. Type and size of Process data that the device will receive/send.
3. Number and type of custom parameters (optional).
4. Number and type of Events(optional).

Vendor ID	**59**
Device ID	1
IODD File Version	1.0
Vendor Name	ExampleVendor
Product Name	ExampleProduct
Product ID	Example01
Vendor URL	www.example.com
Device name	Example device
Device family	Example Family
Description Text	Example
Operate mode OD Size	8 Bytes
Sample Time sensor	5.0 ms

Figure 51 IODD Example

1.9.3 Three-Layer Communication using IO-Link

For reasons of safety, it must be possible to prove that the communication or control of the Mover runs in real time. For this reason, a PLC with real-time communication must be used, which is why the concept cannot be implemented with pure CAN bus communication. The reason for this is that the Robot Operating System running on the Odroid is not real-time capable, i.e., the Odroid is responsible for the higher-level control (direction, speed, obstacle avoidance) while the PLC is only used to implement the commands coming from the Odroid. In addition, the PLC should trigger an E-stop (interruption of the power supply to the motors) if the sensors detect a collision.

An idea for the future, instead of using the PLC (which consumes a large part of the budget), would be to use a pure CAN bus, since ROS 2 is a successor to the Robot Operating System, where real-time capability is very important. This would make use of an additional PLC to obtain certification for safety critical applications obsolete.

For the reasons mentioned above, communication with several layers must therefore be used. IO-link is used for real-time communication with the PLC. IO-link is used to control the motors and read out the sensors. In case of collision detection by the sensors, the power supply to the motors should be interrupted, this is the first layer of communication.

The second layer is the state machine, which runs on the PLC and, among other things, ensures that the Mover moves in the specified direction at the specified speed (closed loop).

The third layer is the overriding control by the Odroid, which is connected via the CAN bus and can read the speed of the motors.

1.10 PLC AND CODING

1.10.1 Setting up IO-Link Sensor with PLC

Installing IODD File

Normally device manufacturers provide the IODD file as a compressed folder (zip/rar) which contains an XML file (IODD file) and PNG images of the device. Decompress these files and save them in a folder called “IODD data”

Note for sensors from the manufacturer “ifm”

If the IODD folder contains more than one folder, you must select the data from the folder “IODD1.1”

The IO-Link master module requires the IODD to communicate with the IO-Link device you will use. Copy the contents of the folder you created (IODD data) to WAGO´s default folder for saving IODD´s:

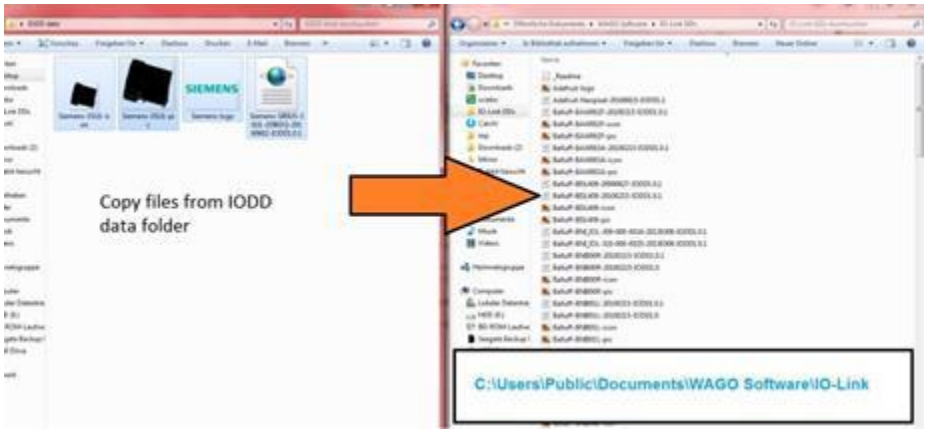


Figure 52 IODD files

The IODD files must be placed in the system in the following folder.

C:\Users\Public\Documents\WAGO Software\IO-Link DDs

Important Note: The PLC must be in STOP Mode to be set up the following instructions once the setup in this section is complete be sure to put the PLC in RUN mode again.

1. Connect your IO-Link device to the WAGO IO-Link master with the cable provided for this lab session. The M12 female threaded connector is connected to the IO-Link shield and the other ending of the cable should be connected to the IO-Link module Port 1 (see datasheet). In the next table the pinout of the cable is shown:

Cable Color	Description
Brown	L+
Blue	L-
Black	C/Q

Figure 53 M12 cable-PLC connection

NOTE:

If more than one conductor must be routed to one connection, these must be

connected in an up-circuit wiring assembly, for example using WAGO feed through terminals.

1. To open the CAGE CLAMP®

insert the actuating tool into the opening

above the connection.

2. Insert the conductor into the corresponding connection opening.

3. To close the CAGE CLAMP® simply remove the tool - the conductor is then

clamped firmly in place.

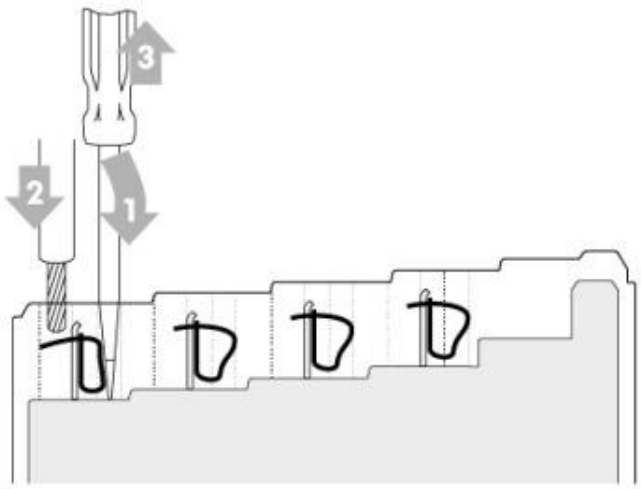


Figure 54 Connecting a Conductor to a CAGE CLAMP

2. Open WAGO´s utility software “WAGO-IO-Check-3” to detect the PLC and setup your IO-Link master. Once you have opened WAGO-IO-Check-3, be sure that the correct IP of the PLC is set through the menu bar (Settings>>Communication).

a. At the top of the opened window selects “Ethernet (TCP/IP).

b. Afterwards type in the static IP of the PLC.

c. Click apply and close the Dialog.

3. After setting up the correct IP in WAGO-IO-CHECK3 software, click the Identify button and wait for the software to detect the PLC and its modules.

4. Right Click on the Module 750-657 and select settings.

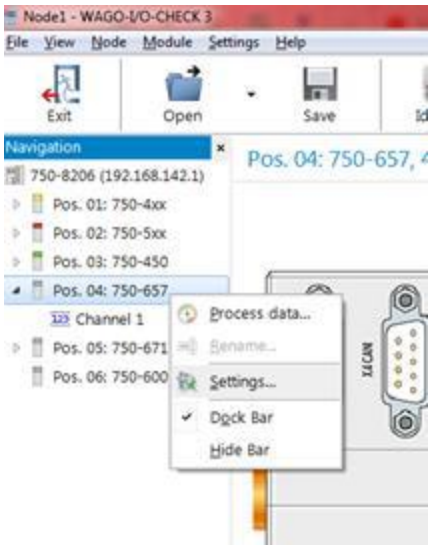


Figure 55 IO Configuration Settings

If the “Connect” button looks as follows, it means that you are connected to the IO-Link master, for the moment.

5. From the table you filled out on step 1 change the “Length Port” option located in the “Master Configuration” tab as follows:

Input (Upstream)

Length Port 1 = Number of bytes of Process data Input (From Device to PLC)

Output (Downstream)

Length Port 1= Number of bytes of Process data Output (From PLC to device)

Figure 56 Port Length and offset in Byte

After you write the data, click the "Auto Calc" option.

Save this configuration to the IO-Link Master by clicking Write>>Save as User settings.

6. Click the Data frame button and select 24 Bytes for Process Image and click "Ok".

7. Click the "Connect" button again to disconnect from the IO-Link master to setup your IO-Link device. It should look like this now

8.Click the tab "Port configuration" and select Port 1.

9. Change the "Operation mode" to IO-Link.

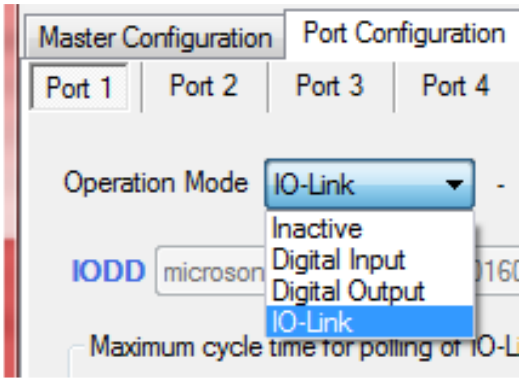


Figure 57 Operation Mode

10. Click the "Refresh List" button, this will look for any new files copied to the directory:

C:\Users\Public\Documents\WAGO Software\IO-Link DDs

11. Select the IODD (xml file) that corresponds to your IO-Link device

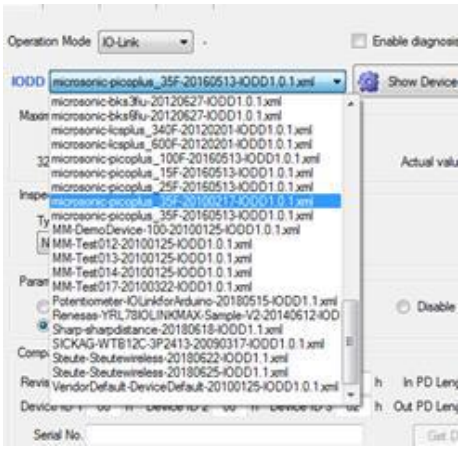


Figure 58 Selection of IODD

12. Set the Inspection Level and Parameter Server Mode options as seen in the next image.

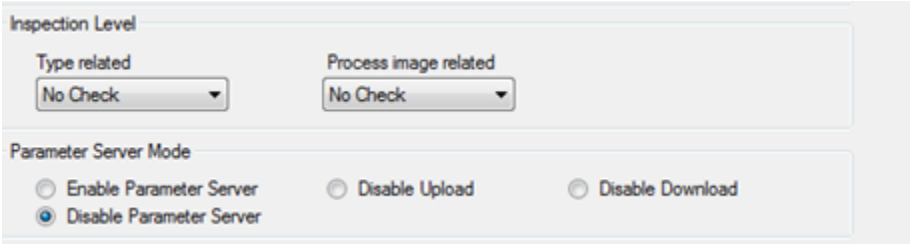


Figure 59 Parameter Server

13. From the table you filled out in Step 1, set the Cycle time.

Note: The cycle time cannot be less than the one indicated in the datasheet of the device.

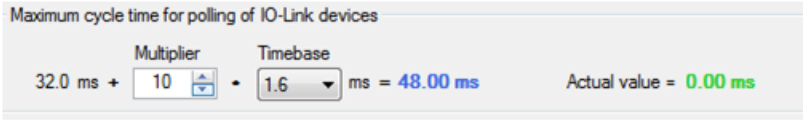


Figure 60 Cycle time

14. Click the "Connect" button again and save this configuration to the IO-Link Master by clicking Write>>Save as User settings.

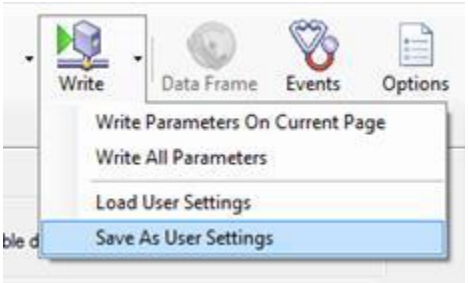


Figure 61 Save as user settings

15. Verify that the IO-Link device is responding by clicking the “Get data” button which will get the Device ID, Vendor Id, and Serial No. of the device.

16. The next step consists of reading/writing on the IO-Link device parameters. To do this click the “Show device” button in the “Port Configuration” tab for “Port 1”. If the next window appears it means that you have set up incorrectly your IO-Link Master, verify that the process data output and input length and IODD has been selected correctly.

18. Once you have the next window open, read all of the device´s information by clicking the next button. A green loading icon should appear in the Menu window, which means the IO-Link master is reading the device´s parameters and information.

19. Modify some of your IO-Link device parameters by using the User manual to know the meaning of its parameters and click the following button to change them.

20. Verify that the parameters that were written, have changed the behavior of the IO-Link device according to the User manual.

21. Check that the device is sending its Process Data by closing the IO-Link settings window and right clicking the module (750-657) and selecting the option “Process Data”.

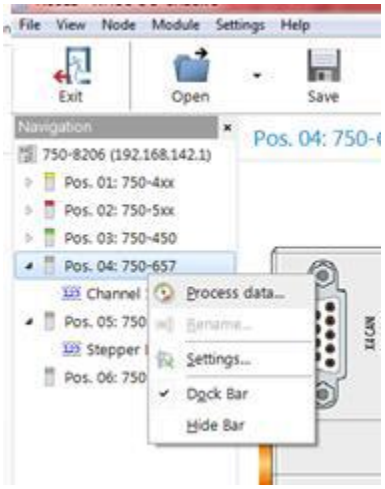


Figure 62 Process Data Check

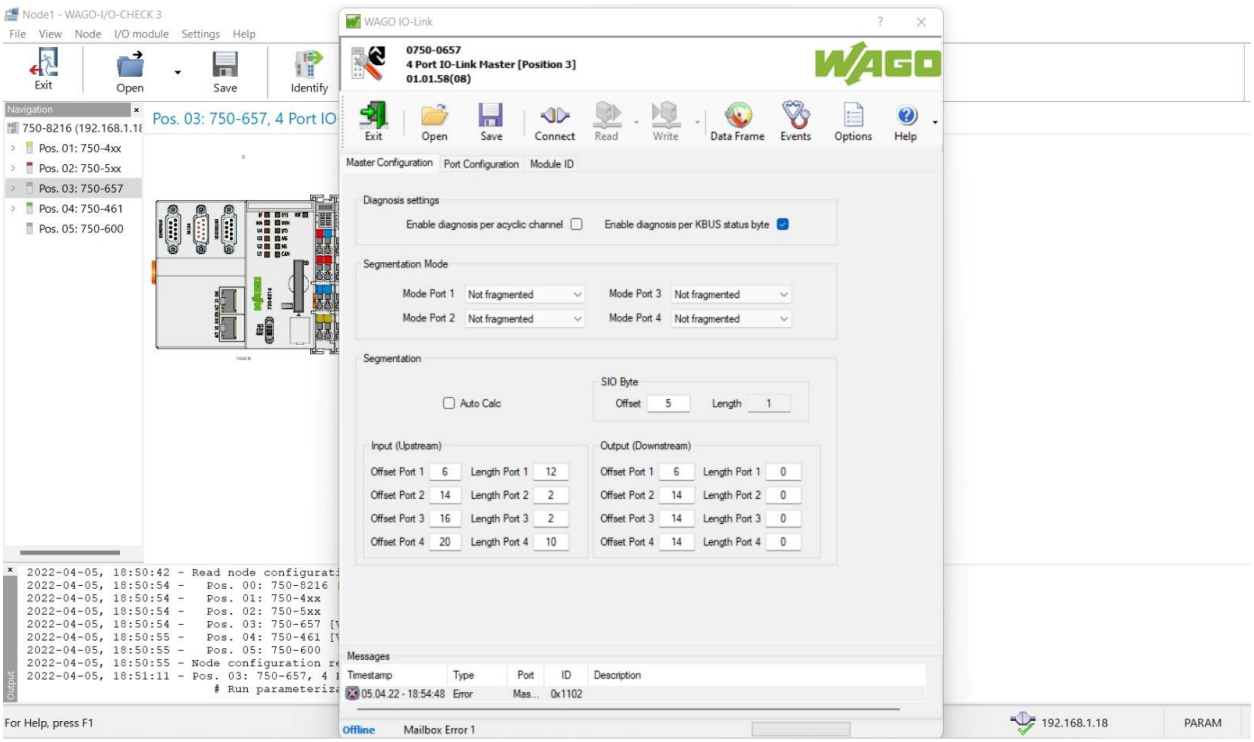


Figure 63 Setting of the output length of each port.

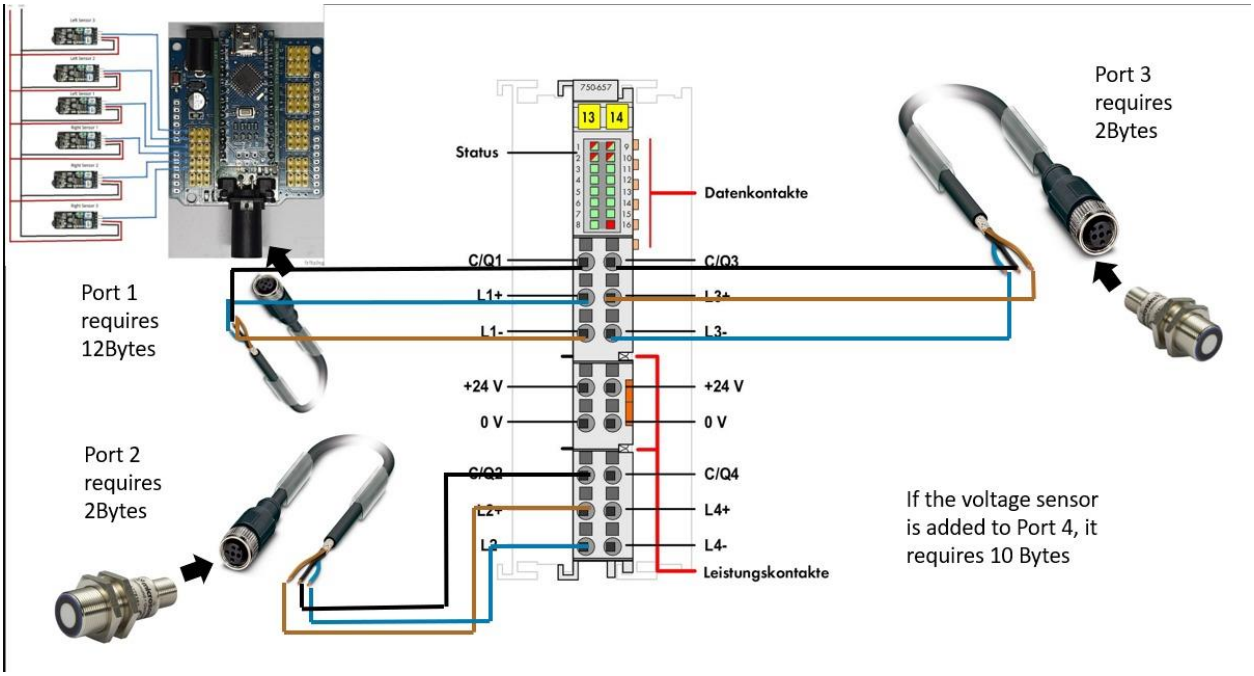


Figure 64 Connections of all the sensors to the PLC

1.10.2 PLC program

Open e!COCKPIT, connect to PLC. After doing that, all the connected modules can be seen in device structure. For example, in our case, we have 3 modules. Now we want to define variables

for the modules which we will use while writing the program and how Arduino controller along with IO link is connected to them.

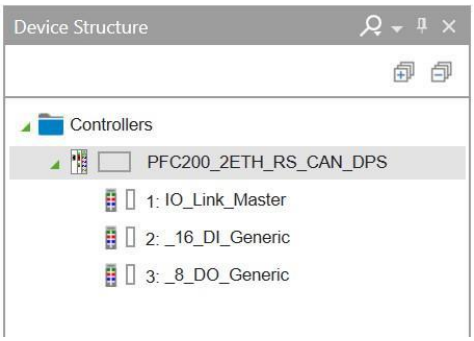


Figure 65 Device on e!COCKPIT

Through the IO read program, we can define how many IO links are connected to the IO Port and with that we can get info of all the devices connected to IO link.

1.10.3 Functions of PLC

IO- Link

All the sensors are connected to Arduino, and we write a program in Arduino IDE defining these variables along with pins. Then we send this data via IO link to the PLC. We need to have libraries for IO link which must be loaded in Arduino Program. These variables are used in PLC program IR step function.

In this System, project for the library “WagoAppIOLink” is delivered. The Motor Controller as well as the Sharp IR distance sensors are connected to an IO-Link master module 750-657.

Required Library:

WagoAppIOLink.compiled_library: Library handling IO_Link master module 750-657.

This program shows the use of general function blocks as well as some visualization templates.

Access of IO-Link process data, e.g., the information from the Sharp IR distance sensor is done by the function block “FbIOL_PortData”. Each port needs an own instance of this function block.

Each IO-Link Sensor connected to one of the ports needs this function block.

FBIOL_PortData (FB)

Interface variables

Scope	Name	Type	Comment
Input	xEnable	BOOL	Enable function block
	I_Port	WagoTypesModule_75x_657.I_Module_75x_657	Access to the module
	bIO_LinkPort	BYTE	Port 1..4
	pTxBuffer	POINTER TO BYTE	Pointer to the area, which should be transmitted, use ADR operator (e.g. ADR(TxData)
	udiTxNBytes	UDINT	Data count to be transmitted, max 32 Byte
	pRxBuffer	POINTER TO BYTE	Pointer to the area, where the received data should be stored, use ADR operator (e.g. ADR(RxData)
	udiRxBufferSize	UDINT	Size of receive Buffer, use SizeOf operator, e.g. SizeOf(RxData)
Inout	xTxTrigger	BOOL	Trigger the transmission of data to the sensor, variable will be reset by function block
	xCommunicationReset	BOOL	reset MBX2 communication in case of fragmented mode
Output	xValid	BOOL	Data from sensor is valid
	xBusy	BOOL	Configuration of the port in progress
	xError	BOOL	Error occured
	udiRxNBytes	UDINT	Number of received bytes
	bRxRefreshCounter	BYTE	Counter for received messages if used with a fragmented port
	oStatus	WagoSysErrorBase.FbResult	ReadingSettings ->first step during configuration of the channel MBX_NotReady-> second step OK ->process values will be delivered InProgress ->wait to reach step MBX_NotReady

Table 4 Interface variables of FBIOL_PortData function block

1.10.4 CAN Communication

As discussed in the earlier chapters of CAN Concept, the CAN telegram consist of an address byte to ensure that the data has been received where its actually needed.

In order to receive the data to the PLC, the 'WagoAppCanLayer2' library is used, where several FBs are there for the same purpose which are listed below.

FbCanL2Open

This function block helps to open the CAN port in Layer2 mode, setting up the transmission mode and baud rate (125 kbps in this case).

Mode is set to mapped mode by setting the variable dwFlags to 0.

Scope	Name	Type	Comment
Input	xEnable	BOOL	opens and configures the port
	I_Port	WagoTypesCan. I_WagoSysCanBase	CAN port, either the internal pfc port or the K-Bus port provided by the module 750-658
	udiBaudrate	UDINT	baudrate in baud valid 100 baud ... 1Mbaud 0 = auto (if supported)
	dwFlags	DWORD	used by module 750-658, details see description
	dwPara	DWORD	reserved for future use
Output	xValid	BOOL	true if port is configured
	xBusy	BOOL	configuration in progress
	xError	BOOL	error occured
	oStatus	WagoSysErrorBase. FbResult	detailed *status* information

Function

This function block opens a CAN port in layer2 mode and configures at least the CAN baudrate

Graphical Illustration

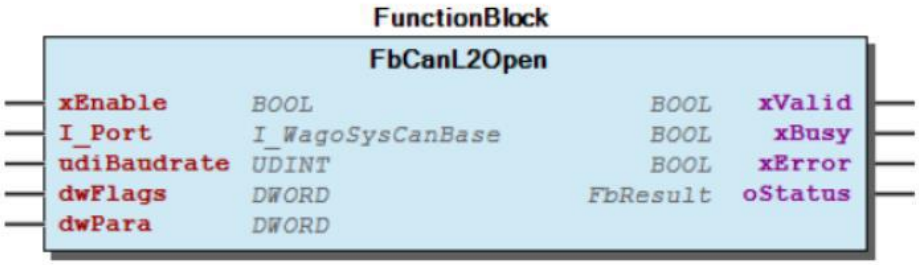


Figure 66 FB to Open the CAN port

```
oOpenInterface           : FbCanL2Open :=(
    xEnable              := TRUE,
    udiBaudrate          := 125000,
    dwFlags              := 0,
    dwPara               := 0 );
sOpenInterfaceErrorInfo  : STRING;
```

Figure 67 Declaration of FB Open Interface

As shown in the figure, the FbCanL2Open is called with the declaration of the input variable needed. Baud rate here is 125 kbps and dwFlags variable sets the mode to mapped mode.

```
oOpenInterface(
    I_Port:= WAGO_CAN_LAYER2_DEVICE);
oOpenInterface.oStatus.ShowResult(sDescription => sOpenInterfaceErrorInfo);
```

Figure 68 Implementation of FB CAN open

FbCanRx11BitFrame

This function block allows to receive CAN messages with a defined CAN ID. This block supports only 11 Bit CAN IDs.

The data will be received in the array of datatype bytes and then is assigned to the respective variables to be processed further. To have a clearer understanding of the code, 'DUT's called structure and union are used.

It can be seen from the description of FB that the `xEnable` enables the receiving function with `xRxTrigger`. The incoming data bytes of CAN telegram is assigned to the array 'xRxBuffer', which could be assigned to the internal variables using structs and union (see code for understanding, for calculations and visualization of Data).

Scope	Name	Type	Comment
Input	xEnable	BOOL	enable function block
	I_Port	WagoTypesCan. I_WagoSysCanBase	Can port, either the internal pfc port or the K-bus port provided by the module 750-658
	xBufferMode	BOOL	False:get latest message, True: use internal buffer ->allows to receive even older messages
Inout	xRxTrigger	BOOL	activate the receive functionality
Output	xValid	BOOL	data received
	xError	BOOL	error occured
	xBusy	BOOL	waiting for Can message with wCanID or waiting for xRxTrigger
	oStatus	WagoSysErrorBase. FbResult	detailed *status* information
	wCounter	WORD	total received messages (11 Bit and 29 Bit)
	wFrames	WORD	messages received in buffer (11 Bit and 29 Bit)
	xRtrFrame	BOOL	RTR frame received
	bRxNBytes	BYTE	amount of byte in message
	aRxBuffer	ARRAY [1..8] OF BYTE	data of CAN message
	wRxId	WORD	Can ID of received message

Table 5 FB for receiving the CAN data

```
oRecv201                : FbCanRx11BitFrame :=(
    xEnable             := FALSE,
    xBufferMode         := FALSE,
    wCanId              := 16#101 );
sRecv201ErrorInfo       : STRING;
//can_data101               : CAN_DATA;
//aRecvData             : ARRAY [1..2] OF BYTE;
bRecvLen201             : BYTE;
xRecvEnable201          : BOOL;
xRecvDataValid201       : BOOL;
xIsRtrFrame201          : BOOL;

wRecvValue201           : WORD; //Received Data Buffer
```

Figure 69 Declaration of FbCanRx11BitFrame

Here, the function block FbCanRx11BitFrame has been declared with the respective CAN IDs. The function block can be called more than once as per the requirement of the system. In this project, the function block is called 3 times with different CAN IDs.

```
oRecv201(
    xEnable     := oOpenInterface.xValid AND NOT oOpenInterface.xError,
    I_Port      := WAGO_CAN_LAYER2_DEVICE,
    xRxTrigger  := xRecvEnable201,
    xValid      => xRecvDataValid201,
    xRtrFrame   => xIsRtrFrame201,
    bRxNBytes   => bRecvLen201,
    aRxBuffer   => GVL.motor_right.can.received_data );
oRecv201.oStatus.ShowResult(sDescription => sRecv201ErrorInfo);
```

Figure 70 Implementation of FbCanRx11BitFrame

FbCanTx11BitFrame

Scope	Name	Type	Comment
Input	xEnable	BOOL	enable function block
	I_Port	WagoTypesCan. I_WagoSysCanBase	CAN port, either the internal pfc port or the K-Bus port provided by the module 750-658
	wCanId	WORD	CAN ID
	xRtrFrame	BOOL	send RTR frame
	aTxBuffer	ARRAY [1..8] OF BYTE	data for CAN message
	bTxNBytes	BYTE	amount of byte in message
Inout	xTxTrigger	BOOL	activate the transmit functionality
Output	xValid	BOOL	data transmitted
	xError	BOOL	error occured
	xBusy	BOOL	transmission in progress
	oStatus	WagoSysErrorBase. FbResult	detailed *status* information

Table 6 FB for Transmitting the CAN data

This function block allows to transmit the CAN messages with a defined CAN ID. This block supports only 11 Bit CAN IDs. The parameters needed for the communication purpose are somewhat similar. However, be aware of whether the variables are defined as input variables or output variables.

```
tx110                   : FbCanTx11BitFrame :=(
    xEnable             := FALSE,
    wCanId              := 16#110 );
    xRtrFrame110        : BOOL;
    btxlength110        : BYTE := 8;
    btxenable110        : BOOL;
    xTxDataValid101     : BOOL;
    wTxValue110         : WORD;
```

Figure 71 Declaration of FbCanTx11BitFrame

Here, the function block FbCanTx11BitFrame has been declared with the respective can IDs. The function block can be called more than once as per the requirement of the system.

```
tx110(
    xEnable    := oOpenInterface.xValid AND NOT oOpenInterface.xError,
    I_Port     := WAGO_CAN_LAYER2_DEVICE,
    aTxBuffer   := GVL.send_can.can_out.sent_data,
    xRtrFrame   := xRtrFrame110,
    xTxTrigger := btxenable110,
    xValid      => xTxDataValid101,
    bTxNBytes   := btxlength110
     );
```

Figure 72 Implementation of FbCanTx11BitFrame

1.11 STATE MACHINE

1.11.1 Introduction

A state machine is a system comprised of states, transitions, and actions. It is used to simplify the real, continuous world description of a system into a finite set of actions and states that vary based on defined conditions.

A state is a set of parameters for the entire system that can be attained and recreated deterministically. This indicates that every occurrence of a state throughout the system's runtime causes the same set of actions to be performed.

A transition switches states deterministically, which means that each state has its own set of circumstances that connect it to other states. Deterministic means that all transitions from a state have non-overlapping sets of parameters that determine the following state.

An action describes what the system does in its state. We differentiate four different actions depending on the current state of the system. These actions are entry action, ongoing action, final action and transition action. The entry action of a given state happens once when reaching it, if the system stays in that state, this action will not trigger again. The ongoing action is what the system does every loop while in a state. The final action is what happens right before the current state transitions to another state, whereas the transition action is what happens during this transition. Since the transition is a very fast process, in the present case transition actions will generally not be needed.

One of the major advantages of using a state machine is the required prerequisite of having a deterministic runtime of each loop. This leads to a clocked behavior that allows reaction times to be predictable. In turn, this makes it straightforward to simulate the real system behavior, allowing setups like software in the loop or hardware in the loop.

Software or hardware in the loop means that during runtime of the system, a simulation is used to determine the outcome of the current train of actions, where hardware in the loop indicates that the system itself runs the simulation whereas software in the loop indicates that the simulation is run on a separate system.

1.11.2 Design

The state machine can be defined in different levels. The lowest level that is closest to the hardware and sensor system will be called Machine Status Level here. Above that, the second level will be referred to as Security Level. The topmost state machine level is the Drive Level. Finally, all state machine levels will be implemented into the same final state machine that will use the Drive Level and implement all other state machines into it through the use of semaphores.

1.11.3 Vision

The state machine relies on the current design of the system. Every additional sensor will be able to make the system safer by improving the environmental awareness of the robot. A potential LIDAR for example would make SLAM (Simultaneous Localization and Mapping) applications

easy, which would potentially lead to a perfect obstacle avoidance, mitigating some of the errors the current loadout might encounter.

Depending on the use case, the robot could also benefit from fixed drive plans, especially if the work it is expected to complete is always the same. For this, an additional “Teach-in” state that relies on manual mode to tutor a movement into the robot could be implemented. This would enable a predictable, consistent movement of the system.

For industrial application, a certain fail-safety in the programming itself is also required. In order to ensure secure drive plans and tamper-proof programming, security systems like Bell-LaPadula could be implemented. This will not be needed in closed-off environments though.

1.12 ROS

ROS has a plethora of applications but in our case, we are using one of its main functions. For Autonomous navigation of a certain robot, there are three building blocks, i.e., Motion Planning, Localization and Perception. ROS has a very convenient way of implementing the aforementioned steps because of its open-source nature and heaps of packages already available at our disposal. In this project we have used ROS-Kinetic but as a suggestion, I would advise the use of ROS-Melodic and Ubuntu 18.04.

At a higher level, ROS should be implemented to achieve free navigation. For real time control of AGV, the motion control system is based on the PLC technology. However, the provided system is not computationally capable of autonomous navigation tasks including perception, localization, cognition, and motion control. A control system combining the WAGO PLC motion control system and an autonomous navigation system is desired. ROS, on which leading research in the robotics field is working and whose active community offers free open-source code, is chosen as the intelligent robot framework responsible for autonomous navigation tasks. The combination of WAGO PLC control system and the ROS robot framework contributes to fulfill our goal of autonomous driving with a real time control system.

The controller and the navigation-PC includes a high-level intelligent navigation system, which deals with navigation tasks and sends motion command to the WAGO. A laptop PC, in which Linux Ubuntu OS 18.04 is installed, runs ROS Melodic. Within the ROS structured communication layer, data is exchanged between different ROS nodes, which can publish over a topic to send different types of messages.

Use this link for further information: http://wiki.ros.org/Documentation

Note: To get a basic idea of what ROS is and how it is used in the field of mobile robotics, follow the turtlebot3 tutorials.

1.12.1 Components

Odroid N2+

ODROID-N2+ is the latest of the ODROID series which are single board computers. They offer open-source support, and the board can run various flavors of Linux, including the latest versions of Ubuntu 20.04 as well as Android distros. By implementing the eMMC 5.0, USB 3.0 and Gigabit Ethernet interfaces, the ODROID-N2+ boasts amazing data transfer speeds, a feature that is increasingly required to support advanced processing power on ARM devices.

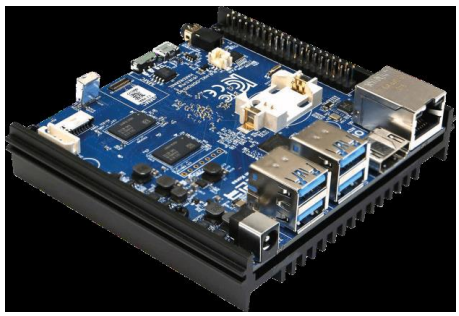


Figure 73 ODROID N2+

	Odroid XU4	Raspberry Pi4B	Odroid C2	Odroid N2+
Processor	Samsung Exynos5 Octa ARM Cortex™-A15 Quad 2Ghz & Cortex™-A7 Quad 1.3GHz	Quad core Cortex-A72	Cortex A53 1.5 GHz quad core	Quad-core Cortex-A73(up to 2.4Ghz) and Dual-core Cortex-A53 (up to 2Ghz)
RAM	2GB LPDDR3 RAM at 933MHz	2/4/8GB	2GB	DDR4 4GB or 2GB at 2133MHz
Interfaces	2x USB 3.0 1x USB 2.0 Ethernet 1x HDMI eMMC5.0 Micro-SD card	2x USB 3.0 2x USB 2.0 Ethernet 2x micro-HDMI Micro-SD card	4x USB 2.0 Ethernet 1x HDMI eMMC5.0 Micro-SD card	4x USB 3.0 1x USB 2.0 Ethernet 1x HDMI eMMC5.0 Micro-SD card
Price	69,95€	37/56/75€	55€	92,28/112,79€

Table 7 Comparison of single board computers

On comparison, it was seen that the higher cost of N2+ is justified by the performance gains over similar single board computers such as the XU4 or the raspberry pi 4. The N2+ contains a quad core 2.4 GHz processor and 4GB of RAM.

IFM 3D Camera

To be able to implement the proper ROS packages for full autonomy, self-driving, and SLAM (Simultaneous Localization and Mapping), the AGV must be able to "see" its surroundings.

The group was provided an IFM O3X101 3D camera to test, with the option of implementing it to the application of the AGV. The O3X101, is an optical camera which measures the distance between the camera and the nearest surface, point by point using the time-of-flight principle. It illuminates the scene with an infrared light source and calculates the distance by means of the light reflected from the surface. At 554€, this IFM 3D camera gives the project more room to work within the budget as well as providing a viable option for implementing a fully autonomous model. The model O3X101 has an operating distance range of 50 to 3000 mm, and a field of view size of 3.4 by 2.5 meters (at a measuring range of 3 meters).

Specifications: Operation distance: 50...3000mm

Angle of aperture (degrees): 60 x 45 (horizontal x vertical)

Max reading rate: 20Hz

Figure 74 IFM Camera - 3D

Figure 75 IFM :PointCloud to Laserscan

Joystick

One would wonder, what is the use of a joystick in autonomous navigation. Does not using joystick defy the purpose of autonomous navigation? Well, the answer is yes and no. If we know the environment or the world in which the robot is going to drive, then it is always a good practice to map the area using tele-operation i.e., using a Joystick, even though it defies the purpose of autonomous navigation. On the other end of the spectrum, if the world is unknown and it is of utmost importance to map the environment autonomously then frontier exploration is performed.

Figure 76 Joystick

1.12.2 Building an AGV with ROS

Writing a URDF

To understand the basics of URDF, use this link http://wiki.ros.org/urdf/Tutorials.

URDF stands for Unified Robot Description Format. When STL files or DAE files are available then URDF is written to visualize the robot in Gazebo and RVIZ. It helps us to understand the transformations better by using robot_state_publisher and joint_state_publisher. tf is a package that lets the user keep track of multiple coordinate frames over time. tf maintains the relationship between coordinate frames in a tree structure buffered in time, and lets the user transform points, vectors, etc. between any two coordinate frames at any desired point in time.

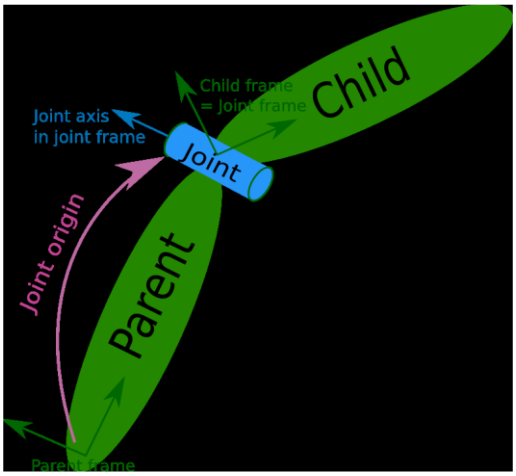


Figure 77 URDF

If you have SOLIDWORKS then you can also use the SOLIDWORKS -URDF plug in for ROS which lets SOLIDWORKS create the URDF. As our university does not provide license for this CAD software, the URDF was written by ourselves. Here is a snippet of the URDF. The entire URDF can be found in the agv_description package. It is documented too. The values of the joints are determined just with trial-and-error method.

```
<link name="base_footprint"/>
<joint name="base_joint" type="fixed">
        <parent link="base_footprint"/>
        <child link="base_link"/>
        <origin xyz="0 0 0.076" rpy="0 0 0"/>
</joint>

<link name="base_link">
        <visual>
                <geometry>
                <mesh filename="package://agv_description/meshes/Autonomous_Vehicle/
→V3_body_Assembly.stl" scale="0.01 0.01 0.01"/>
                </geometry>
                <material name="blue"/>
                <origin xyz="0 0 0" rpy="0 0 0" />
        </visual>

        <collision>
                <geometry>
                <mesh filename="package://agv_description/meshes/Autonomous_Vehicle/
→V3_body_Assembly.stl" scale="0.01 0.01 0.01"/>
                </geometry>
                <origin xyz="0 0 0" rpy="0 0 0" />
        </collision>
        <inertial>
                <origin xyz="0.25 -0.25 0.15" rpy="0 0 0"/>
                <mass value="18.316"/>
                <inertia ixx="0.519" ixy="0.0" ixz="0.0" iyy="0.519" iyz="0.0"
                                izz="0.763"/>
        </inertial>

        </link>
```

Creating WORLD in Gazebo

World is the environment in which the robot is going to move. Hence, in order to simulate it a real world, it needs to be designed inside Gazebo so that we can obtain a map by teleoperation. The world consists of various sdf files and a config file. In our project the name of the world is **rondell** world. It is not necessary to explain how the world was written because we will not change it in this project. If you want to make it fancier, then different objects can have different Gazebo/Colors, but it doesn't make a difference with the simulation and hence we kept it simple.

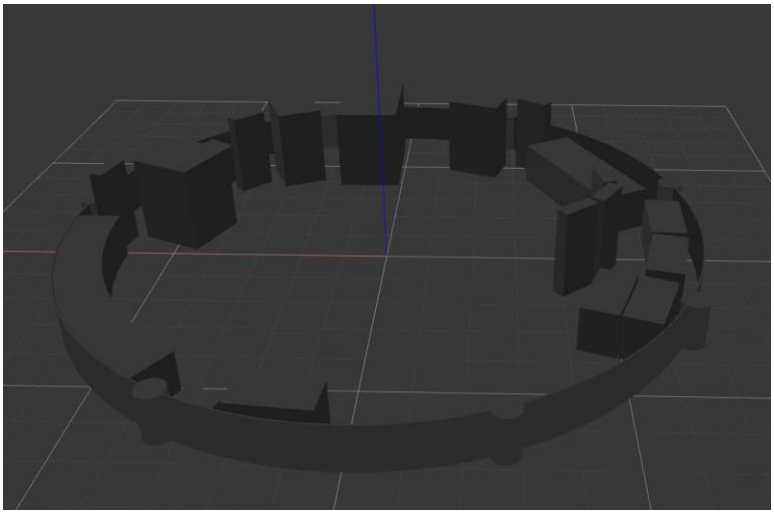


Figure 78 Rondell World

TF Frames

As explained previously, it is very tedious to write down the transform for each link to its parent and that's why we use robot_state_publisher and joint_state_publisher to do this for us.

In the launch file, you can see the usage of this.

```
<node name="joint_state_publisher" pkg="joint_state_publisher" type="joint_state_
,→publisher">

        <param name="use_gui" value="false"/>

</node>

<node name="robot_state_publisher" pkg="robot_state_publisher" type="robot_state_
,→publisher">

        <param name="publish_frequency" type="double" value="50.0" /> </node>
```

When this is used, we get the following transformation:

To launch the file:

roslaunch agv_description test_with_bldc.launch

In order to visualize the tf_tree use:

rosrun rqt_tf_tree rqt_tf_tree

LIDAR simulation and Camera Image

In order to simulate LIDAR and CAMERA we have used their respective Gazebo plug-in's which can be seen in the URDF file.

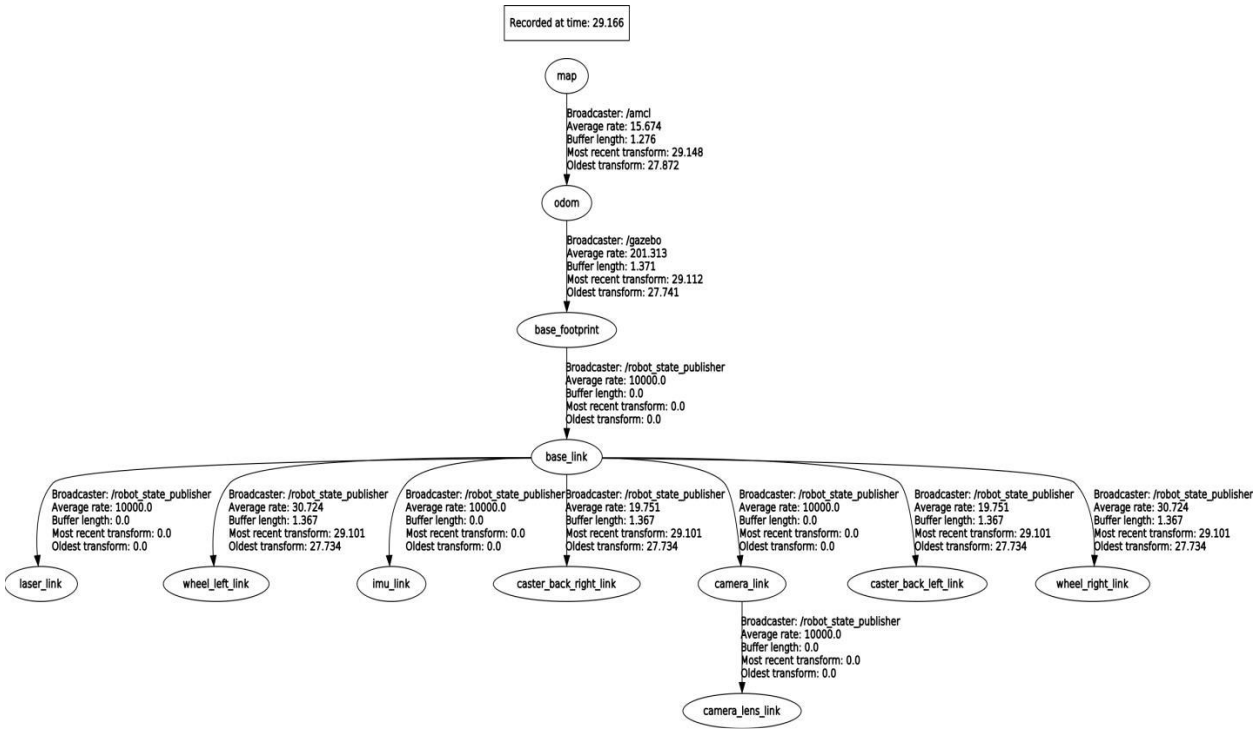


Figure 79 Gazebo plug-in's used

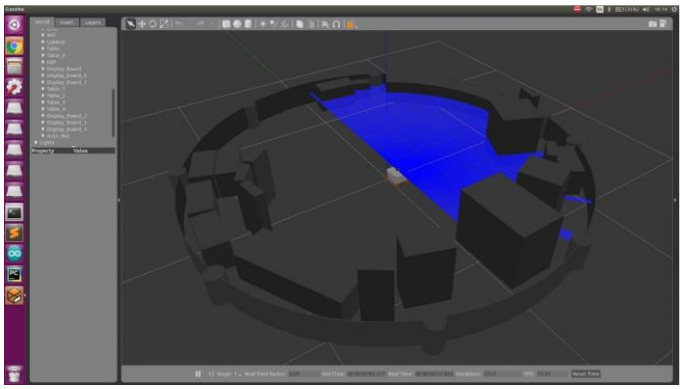


Figure 80 LIDAR seen on Gazebo world

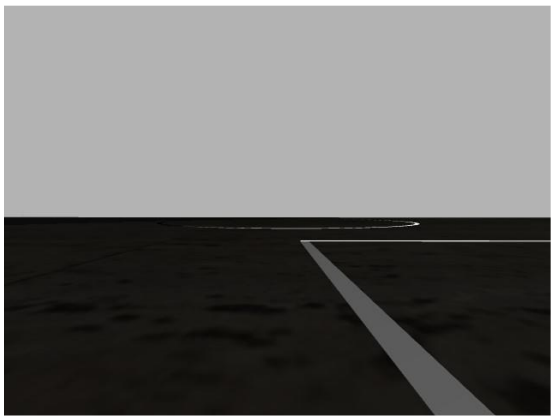


Figure 81 messages published under the topic /scan

This image can then be used to detect the AR Tags and so on. Depending on the strategy of the motion planning in the arena, camera can be utilized to detect anything.

MAPS

Maps are created by operating the robot either in the real world itself (more accurate) or in the simulation environment using the respective/necessary plug ins.

Here we have used two different approaches. One with odometry and the other without odometry.i.e., gmapping and hector_slam. Check the robot_navigation and robot_localization package.

gmapping – this package contains a ROS wrapper for OpenSlam's Gmapping. The gmapping package provides laser-based SLAM (Simultaneous Localization and Mapping), as a ROS node called slam_gmapping. Using slam_gmapping, you can create a 2-D occupancy grid map (like a building floorplan) from laser and pose data collected by a mobile robot. Use the following link for reference: http://wiki.ros.org/gmapping

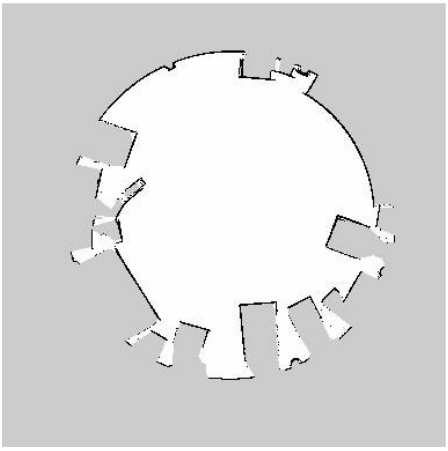


Figure 82 Gmapping

hector_slam – hector mapping is a SLAM approach that can be used without odometry as well as on platforms that exhibit roll/pitch motion (of the sensor, the platform or both). It leverages the high update rate of modern LIDAR systems like the Hokuyo UTM-30LX and provides 2D pose estimates at scan rate of the sensors (40Hz for the UTM-30LX). While the system does not provide explicit loop closing ability, it is sufficiently accurate for many real-world scenarios. The system has successfully been used on Unmanned Ground Robots, Unmanned Surface Vehicles, Handheld Mapping Devices, and logged data from quadrotor UAVs. Use the following link for reference: http://wiki.ros.org/hector_mapping

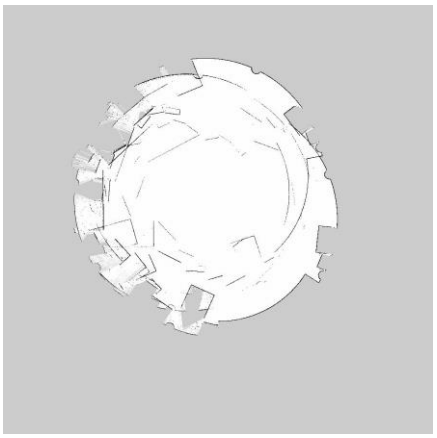


Figure 83 Hector_slam

It is evident from this that gmapping creates better map than hector slam. Hence, if encoder data is available along with IMU, fusing it together using robot_localization package would give the odometry data and produce more accurate map.

Steps to use the workspace

There are different types of packages available in the workspace.

-agv_description has worlds, urdf and launch files. There are 3 different types of robot versions in the package. Launch file used for this simulation is **test_with_bldc.launch**.

-**agv_gazebo** contains different worlds which can be used in the launch files.

-**agv_localization** has maps folder along with AR tag follower and line follower script which can be used.

-**agv_navigation** contains the main navigation launch files along with conifg files, move_base and amcl. Use the following links for reference:

http://wiki.ros.org/move_base

http://wiki.ros.org/amcl

Launch the file: complete_setup.launch

1.13 VISION PIPELINE

1.13.1 Introduction

For the AGV to be able to “see”, a vision pipeline is necessary. This is the series of steps that are required to convert raw image data to useable data that can be used for navigation. For the initial implementation as a line follower robot, the vision sensor needs only to detect obstacles within a specified range in order to stop motion when needed. However, this can be greatly expanded as the rover progresses to higher levels of autonomy. The requirements for a ROS robot are Perception, Localization and Navigation. The 3D sensor can help not only with vision, but also localization. The sensor we are using here is the O3x101 from ifm, a 3D Time of Flight camera, and the steps for setting up and using this sensor to get useful information is detailed below.

1.13.2 Preparation of Operating system

We are currently working with Ubuntu 18.04 for ROS Melodic on the Odroid N2+. The latest version as of writing this is available here:

https://wiki.odroid.com/odroid-n2/os_images/ubuntu/20200224

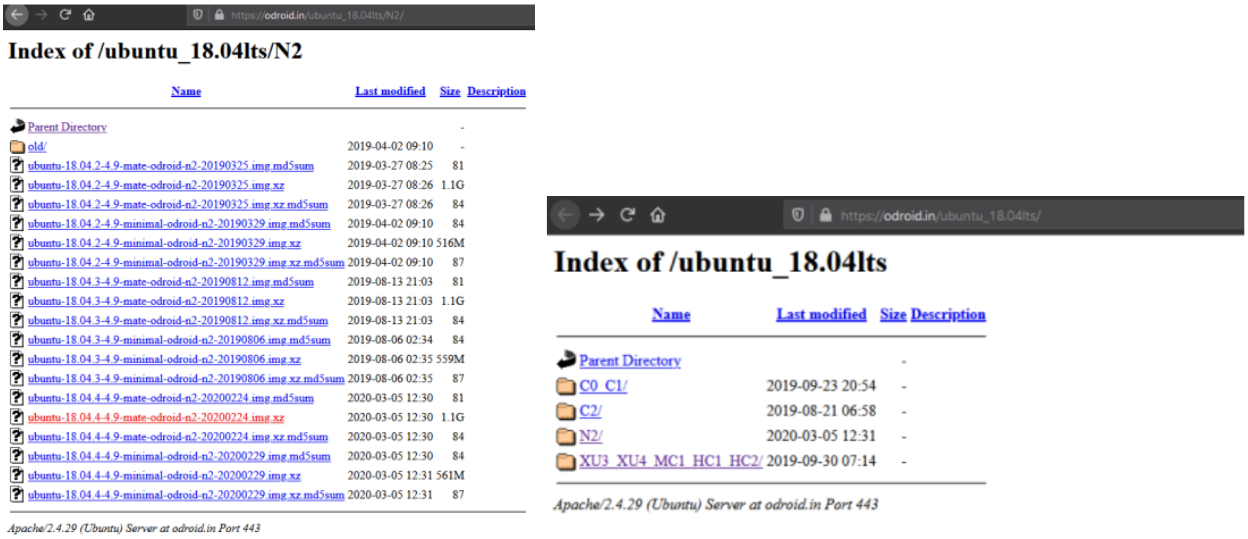


Figure 84 Downloading the ubuntu image

The required operating system image is downloaded onto a computer and flashed to the eMMC card using a compatible software such as balenaEtcher. Use the following link for reference: https://www.balena.io/etcher/

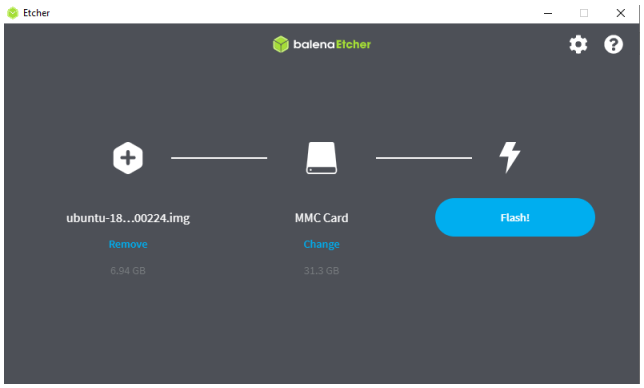


Figure 85 balenaEtcher

Connect the eMMC card and open balenaEtcher

Flash from file

Connect the eMMC to the Odroid

Boot up

The default root password is **"odroid".**

Warning: The Odroid is extremely sensitive to electrostatic discharge and the circuit board should never be handled with bare hands. The case should be left on at all times, except when changing connections (Odroid should be powered off) and suitable precautions such as an ESD mat should be taken.

1.13.3 Installation of ROS Melodic

The main steps for installation of ROS Melodic can be found here:

http://wiki.ros.org/melodic/Installation/Ubuntu

The following steps are valid for installation on Ubuntu Mate 18.04 for the Odroid as well as Ubuntu 18.04 for PCs. It is recommended that the guide above be followed as it is likely to be updated.

- **Setup sources list:**

```
$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
```

- **Setup Keys:**

```
$ sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80' --recv-key C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654
```

- **Installation:**

```
$ sudo apt update
```

```
$ sudo apt upgrade
```

```
$ sudo apt autoremove
```

• **Full ROS installation:**

If an error saying " Unable to acquire dpkg frontend lock…" is encountered while trying to install packages at any point, restarting the Odroid usually fixes it.

```
$ sudo apt install ros-melodic-desktop-full
```

• **Setup sources:**

```
$ echo "source /opt/ros/melodic/setup.bash" >> ~/.bashrc
```

```
$ source ~/.bashrc
```

Additional steps:

Additional packages required:

```
$ sudo apt install git vim dkms htop screen openssh-server python-pip python3-pip
```

1. Additional ROS and Python packages required:

```
$ sudo apt install python-catkin-tools ros-melodic-jsk-rviz-plugins ros-melodic-usb-cam ros-melodic-cv-camera ros-melodic-pid libgstreamer-plugins-base1.0-dev
```

2. Install Visual studio: (optional) (this will not run on the Odroid but can be used on a PC to write code)

```
$ snap install code –classic
```

1.13.4 Installation of ifm3D drivers

These drivers help the camera interface with the computer. First the driver must be installed followed by the ifm3d wrapper to adapt it for ROS.

Preparing the system:

https://github.com/ifm/ifm3d-ros/blob/master/doc/melodic.md

Like the previous section, it is recommended that the original guide be followed. However, this version includes notes that help with issues that may be run into along the way.

1. Update the Baseline Packages of your Ubuntu 18.04 Install

```
$ sudo apt-get update
```

```
$ sudo apt-get -u upgrade
```

2. Install additional dependencies.

```
$ sudo apt-get install libxmlrpc-c++8-dev
```

```
$ sudo apt-get install libgoogle-glog-dev
```

3. Install the drivers:

```
$ mkdir ~/dev
$ cd ~/dev
$ git clone https://github.com/lovepark/ifm3d.git
$ cd ifm3d
$ mkdir build
$ cd build
$ cmake -DCMAKE_INSTALL_PREFIX=/usr ..
$ make
```

Once this last command has finished executing, the 3D camera must be connected. The camera is powered on and plugged into the ethernet port on the Odroid. Once this is done, the connection must be configured so that the Odroid (or PC) can access the packets sent by it.

To configure the network:

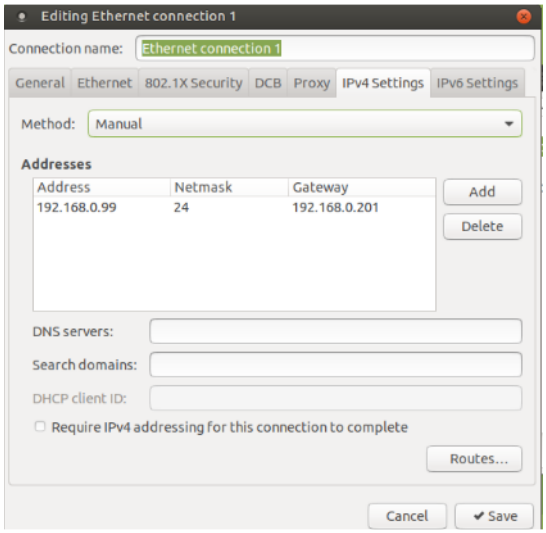


Figure 86 Ethernet configuration settings

1. Open Network Connections and select the ethernet connection, usually it is the Ethernet connection 1,

2. Navigate to the IPv4 settings tab,

3. Set the method to manual,

4. Click Add and give the following details.

Address: 192.168.0.99 (99 here can be replaced if there is no conflict)

Netmask: 255.255.255.0 (This may change after entering, but that is not a problem)

Gateway: 192.168.0.201 (From the camera datasheet)

After saving these settings, ensure that the network connection is enabled (else click Ethernet connection 1 in the internet connections menu at the top right corner to connect it).

To test the connection, run the following command to ping the given IP address, which should show no losses.

```
$ ping 192.168.0.99
```

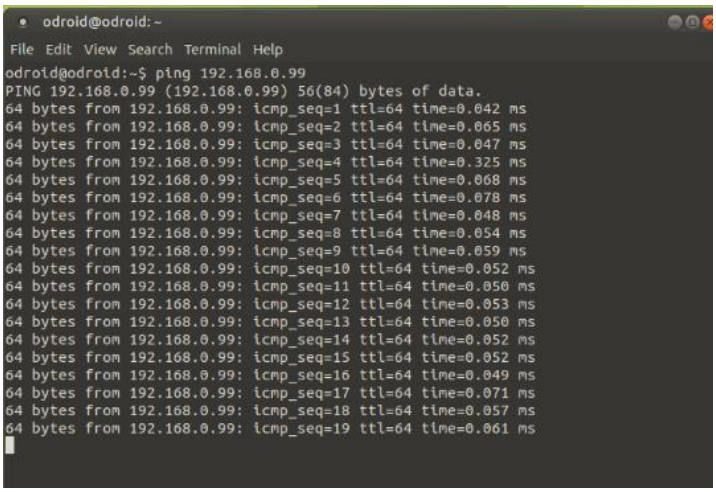


Figure 87 Pinging the sensor

The next command runs a series of tests on the camera, including a reboot test. If the network is not configured to automatically reconnect when the connection is available, the camera will not be detected after the reboot, and this will cause `$ make check` to fail. Therefore, in the first tab of the settings for Ethernet connection 1, this should be checked.

```
$ make check
```

```
$ make package
```

```
$ make repackage
```

The following commands install packages that are regularly updated. Therefore, it is often the case that the versions available are newer ones, leading to the commands given below not working. Running `$ ls` in the current working directory will display the available packages from which the correct version can be determined. Alternatively pressing the Tab key to autocomplete after "ifm3d_" will also work. Please note that "arm64" is the relevant to the Odroid and similar devices and that with most PCs, the package will be replaced by amd64.

```
$ sudo dpkg -i ifm3d_0.20.0_arm64-camera.deb
```

```
$ sudo dpkg -i ifm3d_0.20.0_arm64-framegrabber.deb
```

```
$ sudo dpkg -i ifm3d_0.20.0_arm64-image.deb
```

```
$ sudo dpkg -i ifm3d_0.20.0_arm64-swupdater.deb
```

```
$ sudo dpkg -i ifm3d_0.20.0_arm64-tools.deb
```

This concludes the installation of the drivers that help interface the camera with the computer.

1.13.5 Installation of the ifm 3D-ROS wrapper

The guide for this can be found at:
https://github.com/ifm/ifm3d-ros/blob/master/doc/building.md
For the following steps, the camera must be disconnected.

• Installation of gedit

```
$ sudo apt-get install gedit
```

• Open the .bashrc file

```
$ gedit ~/.bashrc
```

• Add the following lines at the end of the file

Make sure to refer to the right ROS Version in the first two lines. On the website the following lines refer to Kinetic, whereas we work with Melodic.

```
if [ -f /opt/ros/melodic/setup.bash ]; then

source /opt/ros/melodic/setup.bash

fi

cd ${HOME}

export LPR_ROS=${HOME}/ros

if [ -d ${LPR_ROS} ]; then

for i in $(ls ${LPR_ROS}); do

if [ -d ${LPR_ROS}/${i} ]; then

if [ -f ${LPR_ROS}/${i}/setup.bash ]; then

source ${LPR_ROS}/${i}/setup.bash --extend

fi

fi

done

fi
```

Figure 88 Editing the .bashrc file

Download the wrapper filed from GitHub

```
$ cd ~/dev
$ git clone https://github.com/lovepark/ifm3d-ros.git
```

At this point the camera must be reconnected.

```
$ cd ..
$ mkdir ~/catkin
$ cd ~/catkin
$ mkdir ifm3d
$ cd ifm3d
$ mkdir src
$ cd src
$ catkin_init_workspace
$ ln -s ~/dev/ifm3d-ros ifm3d
```

- Build the code

```
$ cd ~/catkin/ifm3d
$ catkin_make -DCATKIN_ENABLE_TESTING=ON
$ catkin_make run_tests
$ catkin_make -DCMAKE_INSTALL_PREFIX=${LPR_ROS}/ifm3d install
```

Package Installation should now be complete.

1.13.6 Testing

• In two different new terminals, run the following commands

$ roslaunch ifm3d camera.launch

$ roslaunch ifm3d rviz.launch

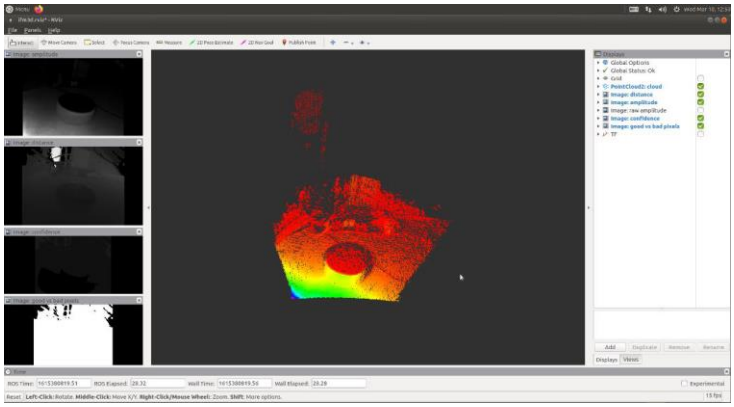


Figure 89 RViz opens to display the point cloud data

RViz should open and directly display a pointcloud image from the camera. This should show that the setup has been performed correctly.

1.13.7 Creating the first ROS node

A good guide to writing a first ROS node to work with the Point Cloud Library can be found at http://wiki.ros.org/pcl/Tutorials.

The tutorial deals with the voxel grid filtering and SAC segmentation which are commonly used processes in 3D vision pipelines. The steps that were followed to create the vision pipeline closely follow this guide and the PCL tutorials at:

https://pcl.readthedocs.io/projects/tutorials/en/latest/

• Create the package (here named “agv_ws”)

```
$ mkdir ~/agv_ws
$ cd ~/agv_ws/
$ mkdir src
$ cd src/
$ catkin_create_pkg agv_vision pcl_conversions pcl_ros roscpp sensor_msgs
```

• Create the empty node file here

```
$ cd agv_vision/src/
```

$ touch obstacle_detection.cpp

"**obstacle_detection.cpp**" is the node where the code can now be written.

• Edit the "package.xml" file (in the package agv_vision folder) and add the following lines

<build_depend>lick-all-dev</build_depend>

<exec_depend>libpcl-all</exec_depend>

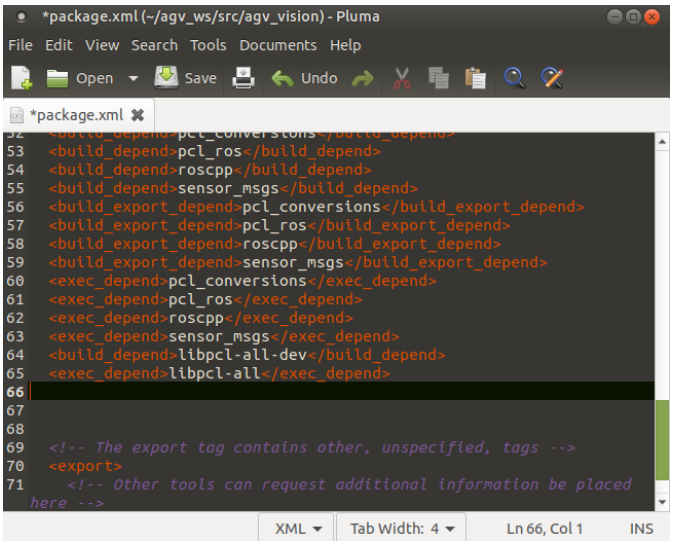


Figure 90 Editing package.xml

• Edit the "CMakeLists.txt" file (in the package agv_vision folder) and add the following lines at the end

add_executable(obstacle_detection src/obstacle_detection.cpp)

target_link_libraries(obstacle_detection ${catkin_LIBRARIES})

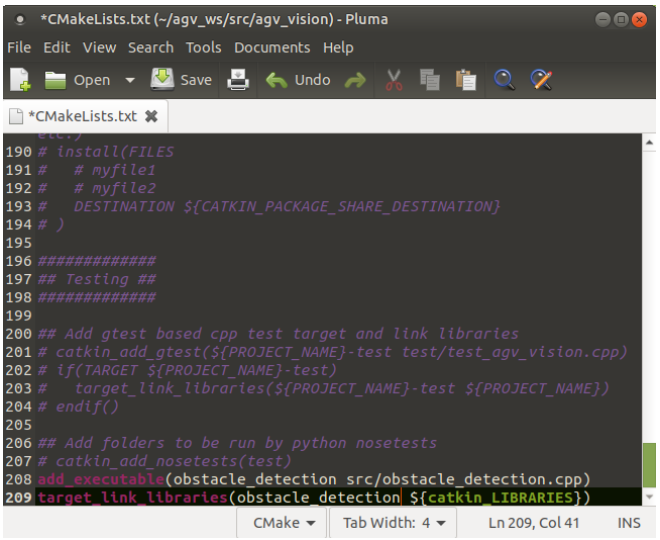


Figure 91 Editing CMakeLists.txt

Every time the code here is modified, the workspace must be recompiled to update these changes. The following steps show how to compile with $ catkin_make.

• Navigate to the workspace directory

```
$ cd ~/agv_ws/
```

• Compile

```
$ catkin_make
```

• Source the workspace

```
$ source devel/setup.bash
```

The ".bashrc" file contains commands that are run every time a new terminal is opened. Therefore, the following adds the command to source the workspace to ".bashrc" preventing the need to do so every time a terminal is opened. However, it is necessary to source, every time the code is recompiled.

• To add the source command to .bashrc

```
$ echo "source ~/agv_ws/devel/setup.bash" >> ~/.bashrc
```

• The ros node that was created can be run with

```
$ rosrun agv_vision obstacle_detection
```

1.13.8 Steps involved in 3D image processing

The raw input from the 3D camera provides us with a group of points known as the pointcloud which only represents how far the light from the laser has travelled before it was reflected to the sensor. By itself, this data does not tell the computer much about the obstacles present in front of the camera. Therefore, a sequence of processing steps must be performed to get useful information out of the raw camera data. Detailed below are some of the most-used functions from the point cloud library, many of which are actively used for obstacle detection in this project. These functions are explained in detail in the tutorials provided by the Point Cloud Library.

https://pcl.readthedocs.io/projects/tutorials/en/latest/

Each of the functions are performed on the same raw image as seen below. The test bench consists of three objects placed on a flat tabletop surface.

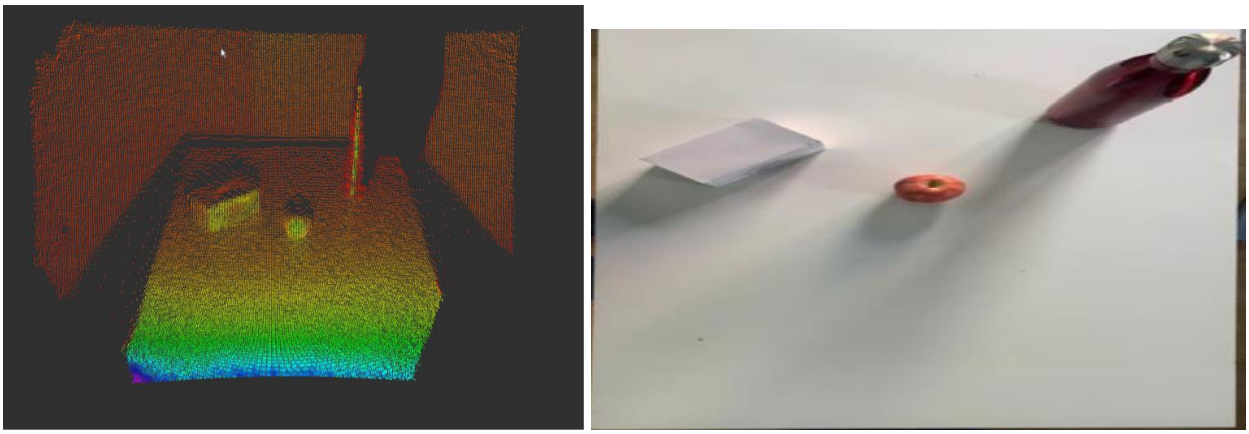


Figure 92 Raw input and test setup

1.13.9 Outlier Filtering using Statistical Outlier Removal filter

The raw point clouds that are generated by the 3D cameras often contain stray points which can affect the results of further calculations that are performed. This could particularly affect processes like the estimation of surface normal or the calculation of centroids. Some of these irregularities can be solved by performing the statistical analysis on each point's neighborhood and trimming those which do not meet a certain criterion. Statistical Outlier Removal is based on the computation of the distributions of the distances of all the points in the data set to those in its neighborhood. It is assumed that the resulting distribution is Gaussian with a mean and a standard deviation. Those points where the mean distances do not fall within the defined interval are determined to be outliers and removed from the dataset.

```
// Outlier filtering (removes stray points)
pcl::StatisticalOutlierRemoval<pcl::PointXYZ> sori;
sori.setInputCloud (cloud);
sori.setMeanK (50);
sori.setStddevMulThresh (3);
sori.filter (*cloud_outlier);
```

Code Explanation

First an object of type StatisticalOutlierRemoval named "sori" is created. The input cloud is stored in variable "cloud". "setMeanK" is used to set the number of points that are used to create the distribution. "setStddevMulThresh" is sets the threshold standard deviation above which the points are removed. "sori.filter" is used to run the filtering and the filtered cloud is stored in the variable "cloud_outlier".

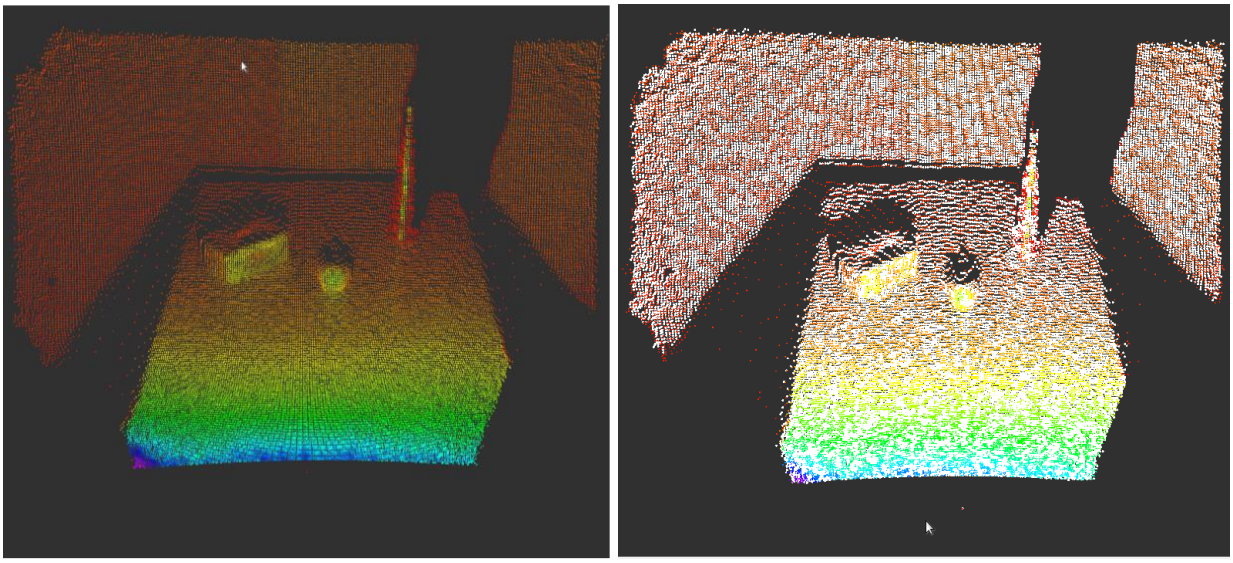


Figure 93 Outlier Filtering

The raw data on the left show a lot of stray points between the table and the walls. The white points on the right show the filtered output cloud overlayed on the input cloud. The threshold value needs to be tuned to the specific application as a value that is too low can end up in unintended removal of data and the too high value will not be effective.

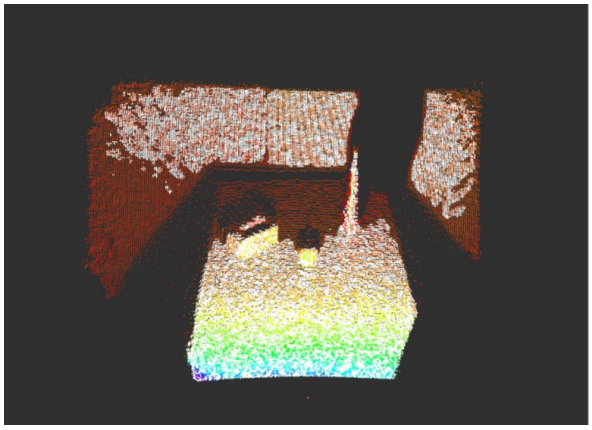


Figure 94 Too low threshold value

1.13.10 Passthrough filtering to crop the cloud

Passthrough filtering is used to crop the point cloud by removing points that lie outside the defined bounds. This helps us to focus the processing on only the region that is required.

Code Explanation:

```
//Passthrough filtering - x (set maximum and minimum bounds)
pcl::PassThrough<pcl::PointXYZ> passx;
passx.setInputCloud (cloud);
passx.setFilterFieldName ("x");
passx.setFilterLimits (0.5, 1.2);
passx.filter (*cloud_passfilteredx);

//Passthrough filtering - y (set maximum and minimum bounds)
pcl::PassThrough<pcl::PointXYZ> passy;
passy.setInputCloud (cloud_passfilteredx);
passy.setFilterFieldName ("y");
```

An object “pass” of type Passthrough (separate objects are used for each direction) is created and the input cloud is set. The fieldname is used to set the direction in 3D cropping takes place. The X-filter field determines the range in the forward direction whereas the Y-filter field crops the image horizontally. “pass.filter” performs the filtering and the output is stored in the cloud ”cloud_passfiltered”.

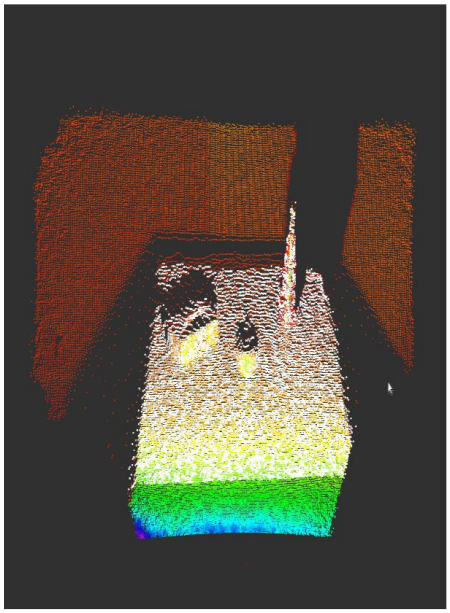


Figure 95 Passthrough filtering

As seen above the cropping reduces the area of consideration to only that of the table.

1.13.11 Voxel Grid filtering

Voxel Grid filtering is used to down sample the point cloud if it is too dense to reduce the processing load required. The voxelated grid approach works by dividing the input cloud into a grid of 3D boxes named voxels. The points in each voxel are approximated by considering the centroid of all the points in the vowel.

Code Explanation

```
// Voxelgrid filtering (reduces number of points)
pcl::VoxelGrid<pcl::PointXYZ> vox;
vox.setInputCloud (cloud_passfilteredy);
vox.setLeafSize (0.01, 0.01, 0.01);
vox.filter (*cloud  filtered);
```

The object named “vox” is created and the input cloud is set similar to the previous steps. The “setLeafSize” function allows you to define the volume of the individual voxels that are used to compute the downsampled cloud. The “filter” function performs the filtering and stores the output in a cloud named “cloud_filtered”.

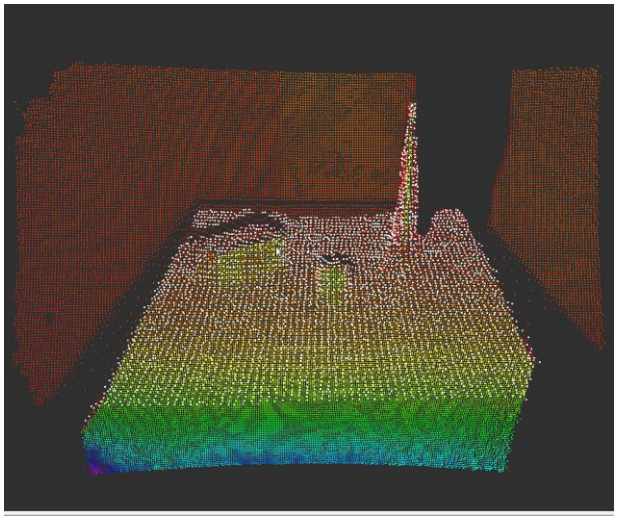


Figure 96 Voxel grid filtering

As seen in the image above, the filtered cloud contains fewer points than the input cloud but does not lose the structure of the surface which is being scanned.

1.13.12 Planar Segmentation

In many applications like the rover, it is assumed that the ground is regular and continuous, and that only the obstacles are required to be detected. In these cases, the next step in processing would be to remove the ground plane from the data. The ground would be the plane with the most points lying on it. To find this a fitting algorithm such as RANSAC is used. RANSAC stands for RANdom SAmple Consensus, and it is an iterative method that randomly samples the data set. A distance threshold is set and for each plane around a point, the other points are determined to be inliers (points lying within the threshold distance) or outliers. The plane with the most inliers is determined to be the ground plane. An extraction step can then be used to remove all the points in this plane.

```
//Segmentation step
pcl::ModelCoefficients::Ptr coefficients(new pcl::ModelCoefficients);
pcl::PointIndices::Ptr inliers(new pcl::PointIndices);
// Create the segmentation object
pcl::SACSegmentation<pcl::PointXYZ> seg;
seg.setModelType (pcl::SACMODEL_PLANE);
seg.setMethodType (pcl::SAC_RANSAC);
seg.setDistanceThreshold (0.04);
seg.setInputCloud (cloud_passfilteredy);
seg.segment (*inliers, *coefficients);

//Extraction step

pcl::ExtractIndices<pcl::PointXYZ> extract;
extract.setInputCloud (cloud_passfilteredy);
extract.setIndices (inliers);
extract.setNegative (true);
extract.filter(*cloud_segmented);
```

Code explanation:

An object “seg” of class SACsegmentation is created. RANSAC is then selected as the method. “setDistanceThreshold” sets the threshold value that determines whether the points are inliers or outliers. The input cloud is set and then the segment function is called. The indices of the inliers are saved and passed to the extraction step. Here the same cloud as the earlier step is passed as input and the indices of the points in the ground plane are passed to the object as well. The extraction object is then used to filter out the given indices.

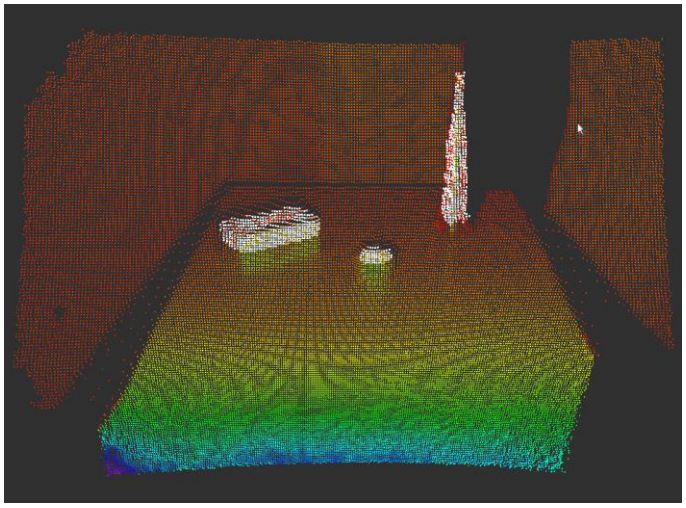


Figure 97 Planar Segmentation

In the dataset, the table would become the ground plane as it is the plane with the most points. The remaining points are therefore only the ones representing the obstacles.

1.13.13 Euclidean Clustering

Clustering is a method of grouping points that are lying close together so that we can separate the different objects out of a 3D scan and later estimate properties such as the position of these objects in 3D space. To optimize the process a search algorithm such as a k-d Tree is used. This helps the program know which points are close to each other. For the actual algorithm of Euclidean clustering a point that has not been previously selected is taken and its neighbors are determined using the k-d Tree. Each of these points are then checked for a predefined proximity (i.e., if they are less than a certain distance) from the point and if they are, the point is added to the cluster. The proximity check is then recursively called on the neighboring points of this point as well. A minimum cluster size is defined to avoid the clustering of stray points.

Code explanation

The KD Tree object is created, and the input cloud is provided. ClusterTolerance, MaxClusterSize and MinClusterSize are defined. The input cloud and search method are provided to the extraction object as well and then the extract function is called. The indices are then stored as "cluster indices''. The number of clusters extracted can be seen in the terminal window while the node is being run.

```
// Creating the KdTree object for the search method of the extraction
pcl::search::KdTree<pcl::PointXYZ>::Ptr tree (new pcl::search::KdTree<pcl::PointXYZ>);
tree->setInputCloud (cloud_segmented);

std::vector<pcl::PointIndices> cluster_indices;
pcl::EuclideanClusterExtraction<pcl::PointXYZ> ec;
ec.setClusterTolerance (0.02); // 2cm
ec.setMinClusterSize (50);
ec.setMaxClusterSize (25000);
ec.setSearchMethod (tree);
ec.setInputCloud (cloud_segmented);
ec.extract (cluster_indices);
ROS_INFO_STREAM(cluster_indices.size()<<" clusters extracted ");
```

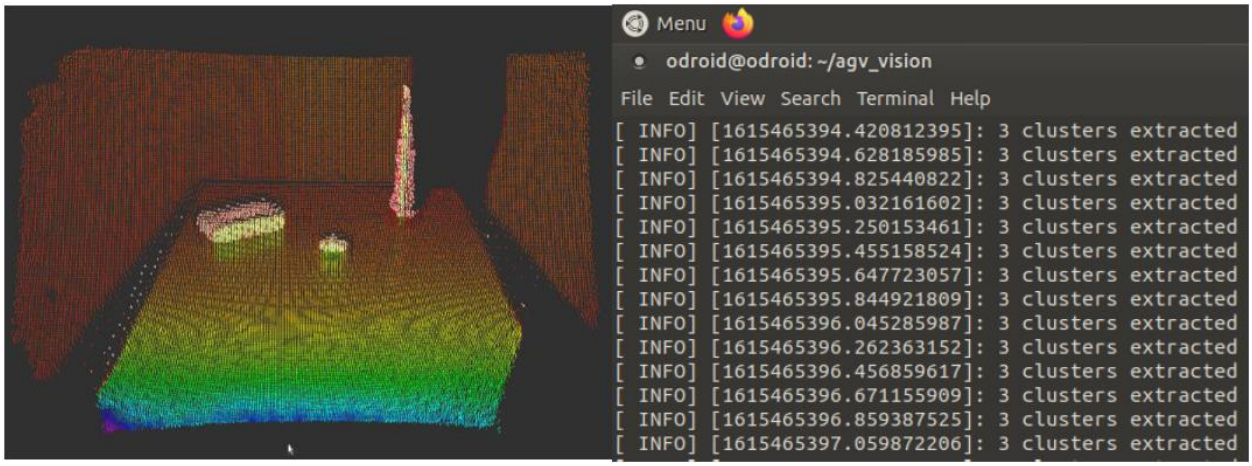


Figure 98 Cluster Extraction with three objects

1.13.14 Obstacle detection

For a line follower mechanism, the most important requirement of a vision sensor is to be able to tell if an obstacle is present or not, for the robot to stop. This can be done in the clustering step itself. A Boolean variable is used which is set to false at the beginning of the processing step and then changed to true if the size of the cluster array is greater than zero. This Boolean variable is published on a different topic from the one of the output clouds. A subscriber to this separate topic can be used to send messages regarding the presence of obstacles to the controlling computer.

```
if(cluster_indices.size()>0)
{
objectDetected.data=true;
}
```

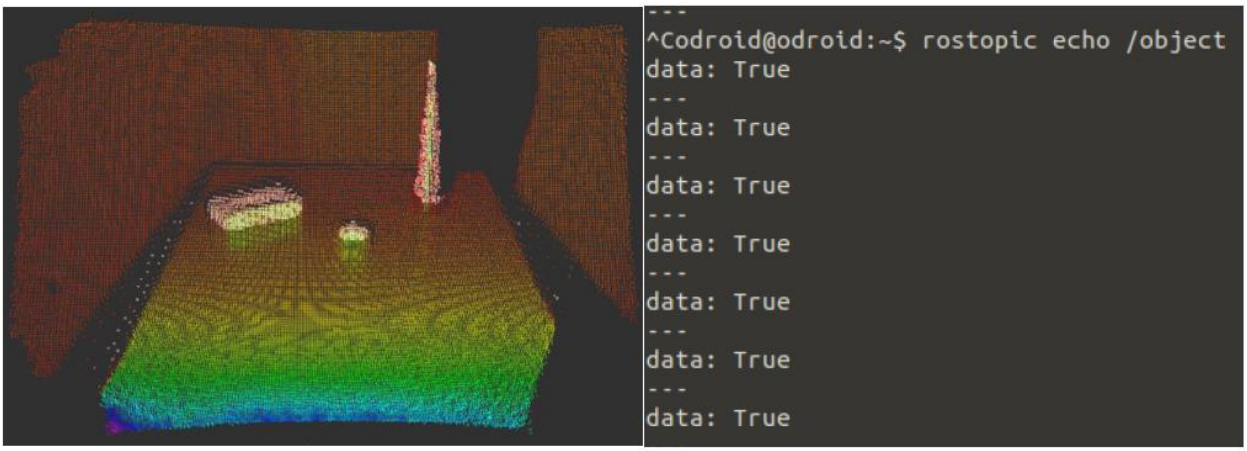


Figure 99 Object detection with three objects

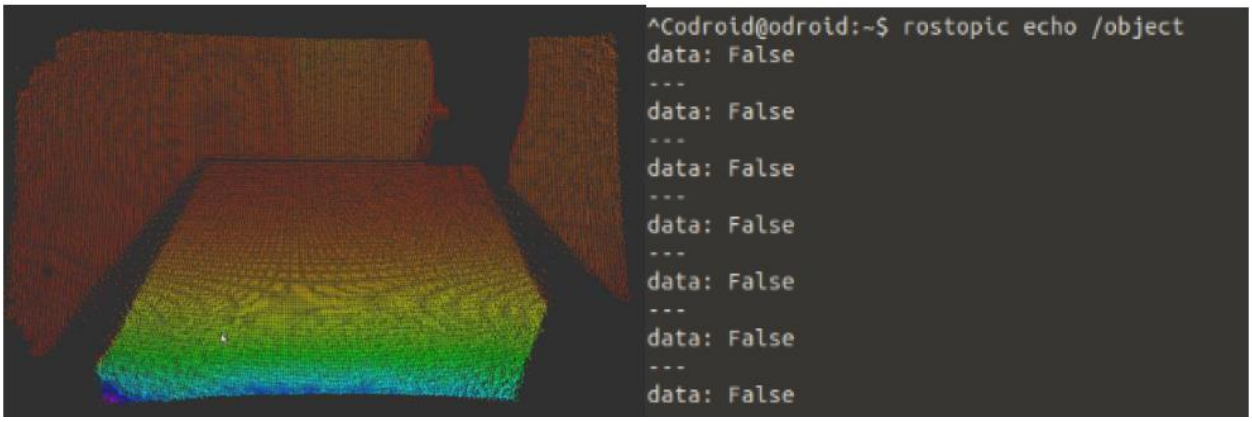


Figure 100 Object detection with no objects

1.13.15 Sensor and ROS messages

There are two types of messages that are currently used in this project for handling the point cloud data. The cloud published by the camera node utilizes the format <sensor_msgs::PointCloud2> which is a type of point cloud message that is compatible with ROS. However, in order to handle the data and perform the different operations of the point cloud library, this data must be converted to type pcl::PointCloud<pcl::PointXYZ> . The function pcl::fromROSMsg(*input_cloud, *output_cloud) can be used to convert the cloud from the ROS format to PCL. After the processing, pcl::toROSMsg(*input_cloud, *output_cloud) is used to convert the message back to PointCloud2 as a ROS message so that it can be published.

1.13.16 CAN Node

When any clusters are detected, the statuses are published over a topic "/object". A subscriber to this topic is used to check the published data and depending on the status of the Boolean, publish the right message over the CAN network.

The new node is added to the src folder of the agv_vision package.

Note: when a new node is added, the file CMakeLists.txt must be edited as well and the workspace must be recompiled with catkin_make.

1.13.17 Launch File

ROS allows the use of launch files to launch multiple nodes at the same time. To launch the camera node, the obstacle detection node, the CAN node as well as the ROS MASTER with one command, we can use a launch file.

Launch files are a perfect tool for managing the processes of a more complex ROS application. On top, a launch file can include other launch files, which makes it even easier to structure the complex starting process of a robot system. A launch file should be placed in a folder named launch inside the ROS package.

$ cd ~/agv_ws/src/agv_vision/

$ mkdir launch

$ cd launch/

$ touch vision.launch

```
<launch>
<node name="obstacle_detection" pkg="agv_vision" type="obstacle_detection" />
<include file="$(find ifm3d)/launch/camera.launch" />
<node name="CAN_node" pkg="agv_vision" type="CAN_node" />
</launch>
```

The "name" determines the instance of the node of a certain "type". Therefore, the name must be unique. The launch file can also include nodes from other packages or other launch files.

The launch file can be run (after compiling and sourcing) with:

$ roslaunch agv_vision vision.launch

1.13.18 Running files on startup

In order to not need a monitor and keyboard connected to the rover, it is essential that the required launch files are run automatically when the device is booted up. This can be done using the "robot_upstart" package. A detailed tutorial on using this package can be found here:

https://roboticsbackend.com/make-ros-launch-start-on-boot-with-robot_upstart/

The package can be installed with:

$ sudo apt-get install ros-melodic-robot-upstart

Then the required launch file can be set up with:

$ rosrun robot_upstart install ~/agv_ws/src/agv_vision/launch/vision.launch –job my_robot_vision --symlink

$ sudo systemctl daemon-reload

If the process needs to be deleted:

$ sudo systemctl disable my_robot_vision.service

Ubuntu can also be set up to bypass the login screen.

1.13.19 ROS over a network

Another useful tool to control the rover without physical input mechanisms such as a keyboard connected to the Odroid is using ROS over the Wi-Fi network. Two or more computers can be networked in such a way that topics published on one system are also visible on other systems. A comprehensive tutorial for doing this can be found here:

https://risc.readthedocs.io/2-ros-network-wifi-ethernet.html

As the processing gets more complicated, it might be worth experimenting with moving parts of the processing steps away from the Odroid to a connected computer with higher processing power and weighing the tradeoff between processing speed and communication speed. This would also be a method of remotely sending instructions to the rover.

1.13.20 Backing up and restoring Odroid image

At any time, a complete image of the Odroid can be saved and restored as required. This might be extremely useful while making system modifications like setting up for CAN. If something goes wrong with the software, it is possible to restore to an earlier stage.

In order to back up on an Ubuntu system:

• In a terminal enter $ lsblk to display all connected drives

• Note down the name of the connected eMMC (the disk itself and not its sub partitions)

• $ sudo dd if=dev/(name of the drive without parentheses) of=(preferredlocation/preferred file name.img) bs=1M status=progress

'If' here is input file and 'of' is output file. In order to restore from the image, interchange the file paths for the two so that the .img file becomes input and the disk becomes output.

1.14 Mechanical components ordered

Sr. No.	Name/Model Number	Qty	Description(mm)	Company Supplier	Quotation	URL
1	Square Profile-6	6	500	ITEM GmbH	NO	https://product.item24.de/produkte/produktkatalog/produktdetails/products/konstruktionsprofile-6-1001042790/profil-6-30x30-natur-41901/
2	Compact Enclosure	1	Art. Nr.1350.500 500 x 500 x 300	Rittal International	YES	Rittal_Angebot_12090692.PDF
3	Wandbefestigung Halter	2 VE (8 Stücke)	KS 1483.010	Rittal International	YES	Rittal_Angebot_12091914Halter.PDF
4	e-Kette Serie 17	3	24 Glieder pro Stück(17.1.063.0)	IGUS GmbH	YES	ITEM_Angebot-Profile6_and_acrylic_glas
5	T Slot Nuts	15	For Profile 6	ITEM GmbH	NO	
6	Abdeck und einfass profile-6	6	500 length	ITEM GmbH	NO	

Table 8 Order list

1.15 3D printing

A great majority of the parts that was designed and used are made with a 3D printing process.

There are three basic stages for 3D printing.

1. Modelling:

This carried out in a 3D modeling software, in our case Autodesk Inventor was. This software enables you to create, open, edit, save, and export 3D printed files.

2. 3D file export:

Once the model is created, it then needs to be exported as either STL, OBJ or 3MF file. These are file formats that are recognized by 3D slicer software. They differ from the file formats that are native to the 3D modeling software as they just hold the final geometry and not the individual primitives and editable content. You can still modify the 3D model's size, but not the geometry.

3. Slicing the file export:

The STL or OBJ file can then be imported into the Cura software where it is sliced and output as a G-Code. This G-Code is just a text document (in essence) with a list of commands for the 3D printer to read and follow such as hot-end temperature, move to the left this much, right that much etc. The slicer software in the Lab is a software from MakerBot which is optimized for the MakerBot printed in the Lab.

Build time:

To understand the build time better, let us consider two cylinders, one vertical and the other horizontal. The cylinder lying down will have a faster printing time than that of the standing cylinder due to the fact it will have total layers.

Part Strength:

It is very important when modeling a 3D printed part to understand the forces that will be applied on that part. In the figure below we can see this effect,

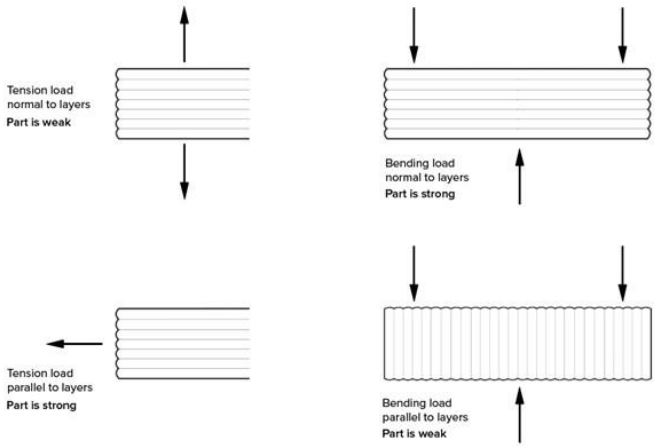


Figure 101 Forces on the printed parts

1.16 Mechanical modelling and design

The bot's mechanical framework should be cost-effective and capable of supporting all of the components, such as the LIDAR, ROS camera, RGB light tower for bot-status, and so on. The cost of a custom-made body, such as custom material and proportions, is significantly greater than the cost of a job shop manufacture. So, in order to save money, we've opted to use a store-bought box, such as a tiny enclosure. We discovered a stainless-steel body and mounting plate compact enclosure from Rittal GmbH with dimensions of 500 x 500 x 300.

Figure 102 Bot casing

Wall Mounting Brackets:

To support the load supporting pallet on to the AGV's body, wall mounting brackets from Rittal GmbH are to be used. 6 or 8 brackets are to be placed on all the four sides on top of the body to support the pallet.

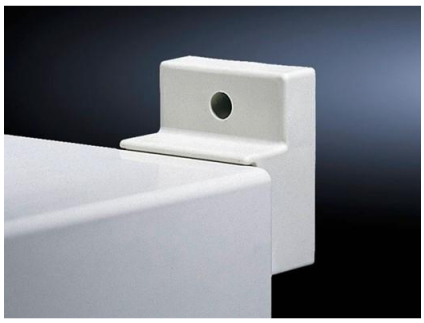


Figure 103 Wall Mounting Brackets

Details of Wall Mounting Brackets:

Model No	KS 1483.010
Provider	Rittal GmbH & Co. KG
Material	Glass fiber reinforced polyamide

Mechanical Design has been updated few times due to suggestions, modifications and material properties and availability in market.

The mechanical concept of the AGV planned is that it should always be accessible even when the AGV is loaded with the pallet. To accommodate this facility, the idea of racks is included in the design.

Racks, which are basically any metal or non-metallic plate supporting all the parts and placed vertically in the bot provided that, racks can slide. i.e., racks have one degree of freedom. If suppose the AGV's items needs to be accessed, then one need not unload the pallet instead can access the items which are placed on the racks, which can slide over guided rails.
The final version of the bot includes a steel enclosure form Rittal B.V with dimensions 500 x 500 x 350 mm with two racks firmly places facing each other. The racks are firmly placed because in the low-cost version to save money and not to compromise with the body integrity and to minimize the mechanical operations on to the block, this idea and concept is finalized.

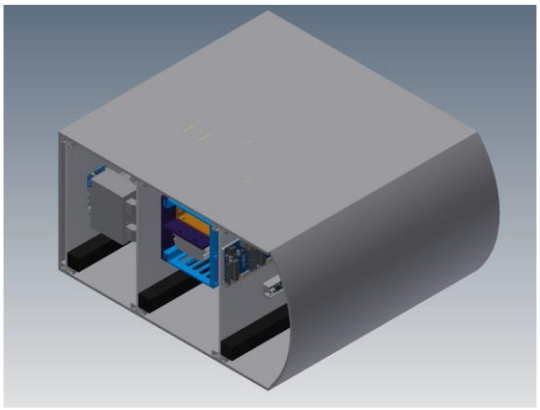


Figure 104 AGV having 3 racks supported on ITEM

Figure 105 Left Rack containing Motor controllers, IO links for sensors and Odroid

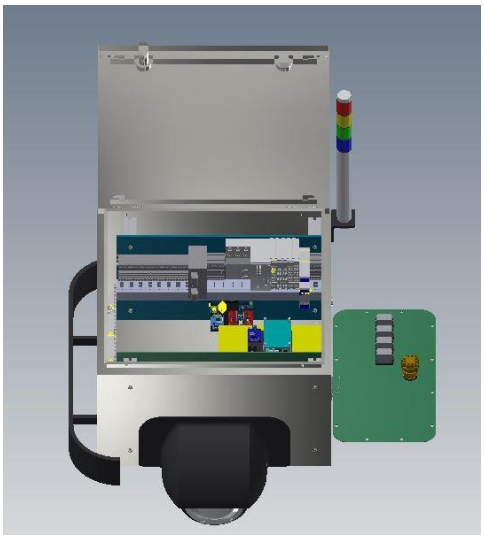


Figure 106 Right rack containing PLC and components, Current and voltage sensor circuit.

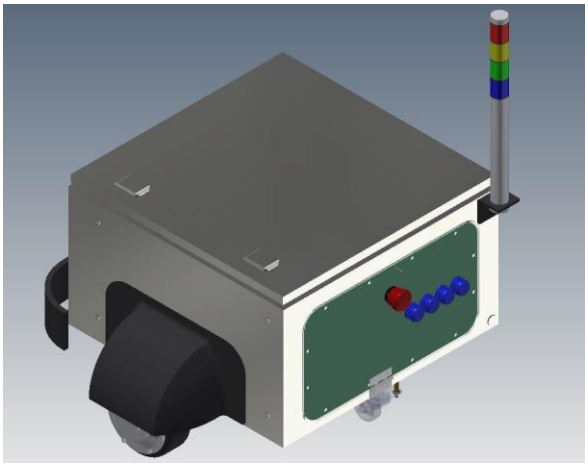


Figure 107 Back view of the AGV with door containing Relay switches and emergency switch

The wiring should be done carefully, and the parts are supported onto the racks using 3D printed parts. The door on the back side of the AGV, can be cut from the acrylic glass from ITEM GmbH and necessary arrangements can be made to place components such as kill switch, 3-chanel relay switch, fuse etc.

CHAPTER TWO – SYSTEM DEVELOPMENT

2.1 Versioning of the Autonomous Mover

In the previous group work, the task was to design two versions of the Autonomous Mover, a low-cost version and a version that meets industrial requirements. Based on this versioning, a selection of different components took place. In the present project work, reference is made only to the" model factory version ", which is intended to meet the requirements of industrial standards.

The temporary goal here is to develop an automated guided vehicle which will follow the line in front of it and avoid obstacles along its path to detect stations accordingly.

2.1.1 Component selection

The requirement for real-time capability and the need to comply with industrial safety standards such as IEC 61058 and DIN EN 60204 while meeting the requirement specification requires careful selection of system components. For this reason, extensive component screening, comparison, and selection has already taken place. Here is a brief list of the components and their intended function that were handed over to the project team in March 2021.

Components used	Function in the concept design
WAGO PCF 200	Main Processing Unit; runs all logic in a finite state machine model
Wago CAN module	Communication bus to interface the Odroid N2+
Wago IO-Link module	Communication bus to interface actuators and analog and digital sensors
Odroid N2+	Acquisition of 3D-camera data for additional object detection; station (AR-tag) detection; navigation, perception, and path planning using ROS
Arduino Nano with IO-Link shield	Sensor data acquisition and provisioning via IO-Link bus
3D Camera	ToF based infrared 3D-camera for supplementary object detection
USB camera	Station detection- AR Tags
RFID sensor	For the purpose of path planning and Localization
Ultrasonic sensor	For the purpose of Obstacle avoidance
IR-sensors	Line following and step detection sensing
Voltage sensors	Safety monitoring of multiple rails
Current sensors	Safety monitoring of multiple rails
Motor controller	Control of brushless DC motors
Brushless DC motors- E bike motor controller	Motion of the AGV
Hall sensors	Rotational monitoring of motors for odometry
12V Pb -battery modules	Energy source
DAC	To run the motors
Safety Bumper Sensor	Emergency stop

Switch cabinet enclosure	Housing of the AGV

Table 9 List of Components

2.2 System communication:

In the following sections, the respective communication systems of the AGV are presented. This is done step by step: First, the previous design of the communication network is briefly summarized and clearly presented. On this basis, changes and further developments to the communication are then made and elaborated.

2.2.1 Previous system design

The planned control network provides for two different communication layers in which the individual components communicate. This layer model serves solely to maintain a real-time capability; additional redundancies are not included. The figure below shows a summary of the components and their bus systems. The highest communication channel is formed by the PLC and the Odroid via the CAN bus. At this highest level, the Odroid can read the current speed of the motor controllers or the motors respectively in order to use them for perception, navigation, and path planning. It also provides information to the PLC via this bus whether an object has been detected by the 3D camera. However, this layer is not subject to real-time requirements since the Linux kernel and ROS used are not real-time capable. The second level is represented by the communication between PLC and the ”smart“ sensors connected to it. The sensors for proprioception as well as exteroception are evaluated by the ADCs of several ATMEGA328p and sent via the multidrop IO-Link line. Since hard real-time requirements apply to the system, the deterministic IO-Link protocol is used to guarantee cycle times and reliability.

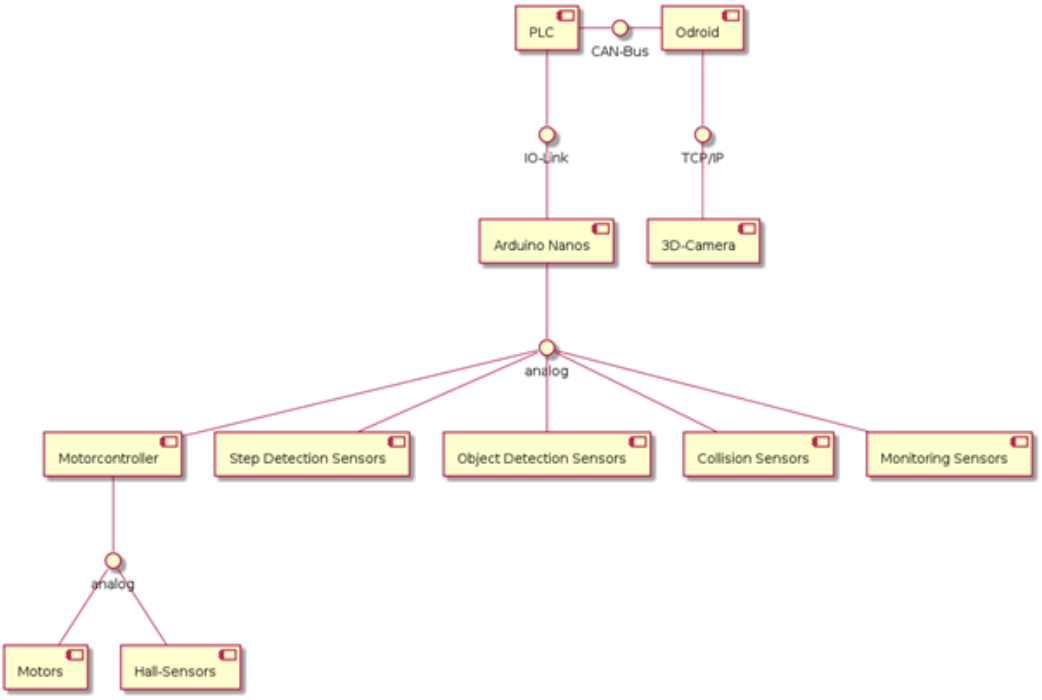


Figure 108 System Communication Flowchart

Up to now, the previous project groups have been able to test individual communication protocols. An elaboration of necessary requirements like cycle times, ID ranges or even a

processing logic has not been worked out yet and has to be defined and worked out in the following development steps.

Maximum Reaction Time Calculation

In order to select a suitable bus system for the safety-critical functions of the AGV calculations for determining the maximum response time of the overall system are made below. For this purpose, two limit cases are considered in particular.

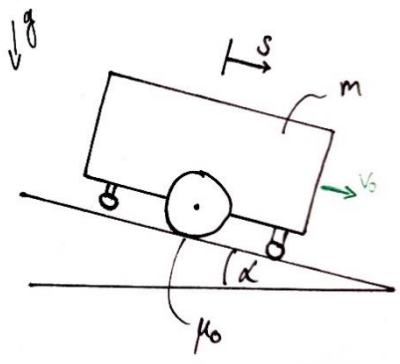


Figure 109 sketch to illustrate the following calculations.

Limit Case 1: Collision

An obstacle on the AGV's downhill track is only detected by collision and the associated triggering of the safety switch on the bumper. The AGV has to perform a full brake in order to not harm the obstacle. The bumper itself is assumed to absorb only a small amount of energy.

To approximate the braking distance the energy theorem is used.

$$E_{kin} + E_{pot} = W_{Break}$$

$$\frac{1}{2} * m * v_0^2 + m * g * s_{Break} * \sin\alpha = \mu_0 * m * g * s_{Break} * \cos\alpha$$

Resulting in a braking distance of

$$s_{Break} = \frac{v_0^2}{2 * g * (\mu_0 * \cos\alpha - \sin\alpha)} = 0{,}4m$$

With (values from non-functional requirements):

$\mu_0 = 0{,}4\ (factory\ floor - AGV\ tires)$
$\alpha = 15°$
$v_0 = 1\frac{m}{s}$

And thus, a braking acceleration of

$$a_{max} = \frac{v_0^2}{2 * s_{Break}} = 1{,}25\frac{m}{s^2}$$

With the given framework conditions, full braking downhill is not practicable because the bumper's dimensions are supposed to cover the entire braking distance, as it is only allowed to absorb a small amount of energy and would therefore be significantly too large. By reducing the permissible downhill speed to v0 = 0,7 m s-1, the braking distance can be lowered to sBreak = 0,196 m.

The bumpers length can then be calculated by

$$l_{Bumper} = s_{Break} + s_{reac} = 0{,}25m$$

With:

$s_{reac} = 5{,}4cm\ (reaction\ distance)$

Thus, this results in a reaction time of the entire system of

$$t_{reac} = \frac{s_{reac}}{v_0} = 77{,}14ms$$

Limit Case 2: Fall down

The AGV detects an elevation, such as a step, with the IR-sensors and must perform a full braking maneuver to avoid driving over the step and sitting on it or even falling off. For this case, it is assumed that the AGV is driving on the flat.

With the new assumptions from above this results in a braking distance of

$$s_{Break} = \frac{v_0^2}{2 * \mu_0 * g} = 6{,}24cm$$

With:

$\mu_0 = 0{,}4\ (factory\ floor - AGV\ tires)$
$v_0 = 0{,}7\frac{m}{s}$

Since the supporting wheels are located directly on the outer edge of the AGV body, it is clear that the IR sensors must be placed before the actual AGV body.

Communication Bus Selection

In order to ensure the data exchange of the individual modules within the AGV, a selection of suitable bus systems is discussed below. Here, a distinction is made between the safety-critical communication to the motor controller of the AGV with regard to its ability to perform an emergency stop as well as critical sensor data and the non-critical communication between the Odroid and its middleware ROS for additional object detection and higher navigation functionalities.

Safety-Critical Communication

The ODrive v3.6 motor controller selected for the project has different control modes such as speed control, position control or torque control. These higher functionalities are provided by

open-source drivers and several interfaces of the motor controllers MCU STM32F405RGT6, namely USB, CAN, UART and PWM. This allows different implementation ideas of the subsystem, which are considered in this subsection. A summary overview is given in Table 10 Furthermore, to ensure reliable readout of sensor data for obstacle detection and the associated emergency stop, possible concepts are compared.

Motion Control

UART

One implementation possibility could be the use of the UART interface in conjunction with the Odroid N2+ used and the ROS middleware running on it. ROS already offers the rosserial protocol for easy integration of standard ROS serialized messages. However, controlling the motor controller via ROS violates the real-time requirements of the project given the fact that neither the used Linux kernel nor ROS cannot guarantee a deterministic response.

CAN

Another implementation option is to use the CAN interface and the Wago PLC PFC200 already provided for the project. This variant guaranteed compliance with safety requirements due to the real-time capability of CAN under certain conditions together with the PLC. Currently, the manufacturer of the ODrive provides a rudimentary CAN protocol named ”CAN Simple“ with 25 commands, which will be expanded to include CANopen and SAE J1939 in later firmware updates. At the moment the given 25 commands can be used to control the motor controller by different input parameters as well as to make the simplest settings and to read errors.

IO-Link

In order to be able to use the full range of functions of the Odrive Motor Controller at the current state of development, it must be addressed via the UART or USB interface together with its native protocol proposed by the vendor. With minor functional restrictions, the ASCII protocol cold also be used. A proposed solution could be to operate the motor controller via an additional IO-Link-capable microcontroller and its UART interface together with a PLC via an IO-Link bus. This way, a deterministic data transfer as well as a deterministic program flow can be ensured to meet the real-time requirements of the project and the full range of functions of the motor controller can be used.

	ODrive-UART-Odroid	ODrive-CAN-PLC	ODrive-IO Link-PLC
Integration effort	API provided	Basic protocol provided	API provided
Functionality	Complete	Limited	Almost complete
Real Time capable	No	Yes	Yes

Table 10 Odroid-PLC communication methods

Conclusion:

The above-mentioned advantages of the IO-Link communication together with the native protocol outperform the other discussed architectures and are thus chosen for this system design.

Non-Safety Communication

In order to ensure a reliable communication between the required PLC PFC200 and the Odroid N2+ as well as a connection with additional sensors appropriate bus systems must be selected. Since these data transfers do not complement any safety-critical functions, they are not subject to the real-time requirements of the project. In this section an overview is given of the solutions to be considered.

Odroid functionalities

CAN

A straightforward solution would be the installation of a CAN bus. Depending on the choice of the safety-critical bus system, this CAN bus could either be merged with safety-critical components or work separated on a second channel. This way the requirements for the Odroid' s CAN transceiver would be less strict in case of a malfunction.

For the implementation a user defined CAN message protocol would have to be defined that ensures the correct sending of status, heartbeat, sensor, and control messages between the PLC and the Odroid. As an alternative, the use of the CANopen protocol could also be considered. A package for this has already been published by the ROS community.

In order to have a sound basis for decision-making, performance tests for CAN communication were carried out in a test setup with the hardware available for the project. The CAN bus was operated with an extended format, a data rate of 500 kbit s-1 and a fixed DLC of 8 packets. The Raspberry Pi served as a replacement for a Wago PFC 200 and as a transmitter of messages that changed the listening Odroid' s GPIO pin level as well as its own. In this way, a maximum frequency of 680 Hz could be determined with the test setup, which corresponds to a usable payload data rate of 42,67 kbit s-1. The latency between the two GPIO levels was 120 µs.

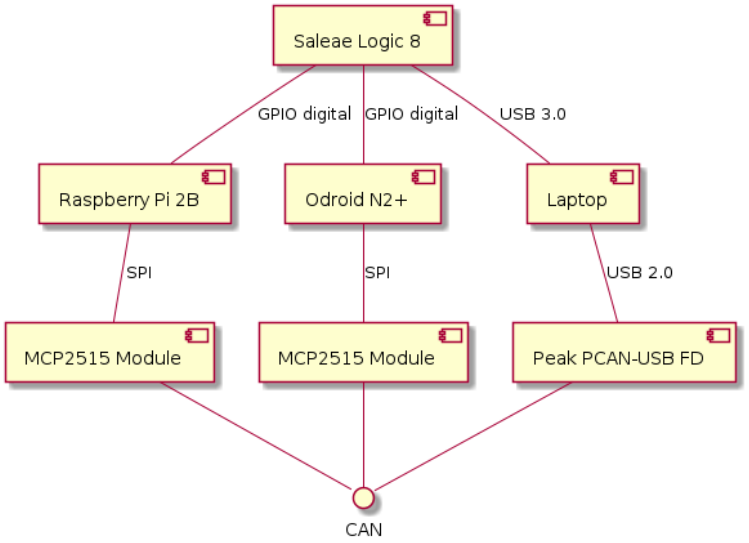


Figure 110 component diagram of the test setup

```python
#!/usr/bin/env python3

import can
import RPi.GPIO as GPIO
from time import sleep

TIME_S = 0.1 # seconds
LED = 18

GPIO.setmode(GPIO.BCM)
GPIO.setup(LED, GPIO.OUT)
GPIO.output(LED, True)

bus = can.interface.Bus(bustype='socketcan',
                        channel='can0',
                        bitrate='500000')

msg = can.Message(arbitration_id=0x500,
                  data=[0, 0, 0, 1, 2, 0xc0, 0xff, 0xee],
                  is_extended_id=True)

while True:
    bus.send(msg)
    GPIO.output(LED, True)
    sleep(TIME_S)
    bus.send(msg)
    GPIO.output(LED, False)
    sleep(TIME_S)
```

Figure 111 code that was executed on the Raspberry Pi 2.

can_node.py

```python
#!/usr/bin/env python

import rospy
from std_srvs.srv import SetBool
from std_msgs.msg import String
import can

def can_node():
    pub = rospy.Publisher('/can/rx_msg', String, queue_size=10)
    rospy.init_node('can_node', anonymous=True)
    rate = rospy.Rate(100) # 100hz
    bus = can.interface.Bus(bustype='socketcan',
                            channel='can0',
                            bitrate='500000')
    while not rospy.is_shutdown():
        msg = bus.recv()
        if msg.arbitration_id == 0x500:
            pub.publish("CAN msg received")
        rate.sleep()

if __name__ == '__main__':
    try:
        can_node()
    except rospy.ROSInterruptException:
        pass
```

Figure 112 publisher responsible to receive CAN messages and publish them on a ROS topic.

gpio_toggler.py

```python
import rospy
from std_srvs.srv import SetBool
from std_msgs.msg import String
import can
import odroid_wiringpi as wpi

LED_GPIO = 22
GPIO_OUTPUT = 1
LED_STATE = 1

def callback(data):
    global LED_GPIO
    global GPIO_OUTPUT
    global LED_STATE

    #rospy.loginfo(data.bool_d)
    if LED_STATE == 1:
        LED_STATE = 0
    else:
        LED_STATE = 1
    wpi.digitalWrite(LED_GPIO, LED_STATE)

def led_toggler():
    global LED_GPIO
    global GPIO_OUTPUT
    global LED_STATE

    wpi.wiringPiSetup()
    wpi.pinMode(LED_GPIO, GPIO_OUTPUT)
    rospy.init_node('gpio_toggler', anonymous=True)
    rospy.Subscriber("/can/rx_msg", String, callback)
    rospy.spin()
```

Figure 113 Subscriber node in the ROS system

The subscriber node in the ROS system that was responsible to toggle the connected LEDs and outputs that where measures with a logic analyzer at 100MSamples/sec.

OPC UA

By implementing an OPC UA server on the Odroid and an OPC UA client on the PLC accordingly, a communication between the Odroid and the PLC can be enabled in which the PLC can read and write published variables by ROS. This way, a user-defined data model can be used to read out sensor values, values of higher-level functions and the additional object detection via the 3D camera.

Just as in the previous subsection, a performance test was also performed for the OPCUA protocol, whose component structure is shown in the figure below. In this test setup, the Raspberry Pi again replaces the Wago PFC 200, whose CPU is similarly performant, and which is used as softPLC with Codesys 4.2.0.0 multicore runtime environment and activated OPCUA server. The Odroid subscribes as OPCUA client to the global variable published by the softPLC and changes its GPIO level like the softPLC itself. Thus, a maximum usable frequency of the GPIOs could be determined to 2 Hz. The latency between the level changes was averaging around 150 ms.

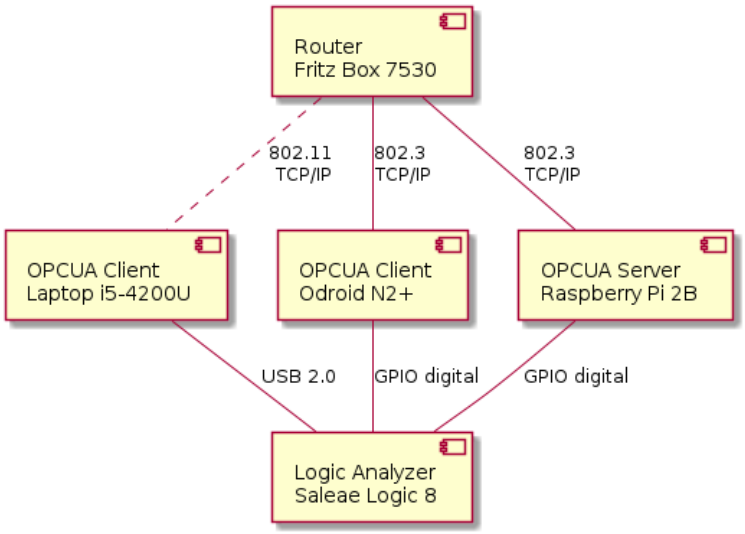


Figure 114 component structure of OPCUA protocol

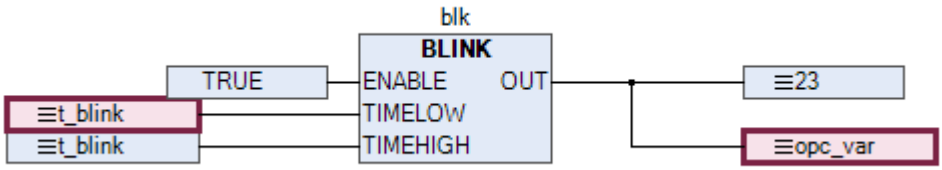


Figure 115 PLC code used to toggle the variable "opc_var" with respect to "t_blink".

```python
#!/usr/bin/env python

import rospy
from std_srvs.srv import SetBool
from std_msgs.msg import String
from ros_opcua_msgs.msg import TypeValue
import odroid_wiringpi as wpi

LED_GPIO = 0
GPIO_OUTPUT = 1
LED_STATE = 1

def callback(data):
    global LED_GPIO
    global GPIO_OUTPUT
    global LED_STATE

    rospy.loginfo(data.bool_d)
    if LED_STATE == 1:
        LED_STATE = 0
    else:
        LED_STATE = 1
    wpi.digitalWrite(LED_GPIO, LED_STATE)

def led_toggler():
    global LED_GPIO
    global GPIO_OUTPUT
    global LED_STATE

    wpi.wiringPiSetup()
    wpi.pinMode(LED_GPIO, GPIO_OUTPUT)
    rospy.init_node('gpio_toggler', anonymous=True)
    rospy.Subscriber("/opcua/opcua_client/topic", TypeValue, callback)
    rospy.spin()

if __name__ == '__main__':
    led_toggler()

```

Figure 116 code run on the Odroid with the "ros_opcua_communication" package

Conclusion

The two test setups show that for the available hardware and software the CAN bus is significantly better performing than the OPC UA server/client. For this reason, the CAN bus is selected for communication between Odroid and PLC.

Sensor Data Acquisition

Various sensors such as accelerometers or current sensors are required in the AGV for condition monitoring. The sensors have different communication interfaces, mainly I2C and SPI.

Odroid GPIO Rail

Since the Odroid already has a GPIO rail with a UART, I2C and SPI interface, the sensors could be connected directly to the Odroid. In this case, a Piggyback extension would be useful for easy connection.

Arduino

Alternatively, a central or multiple Arduinos could handle the communication with the sensors and transmit them bundled to the Odroid via the UART interface. In this way, the Odroid could be protected from possible overvoltage of defective sensors by the sensitive GPIO rail. A second alternative would be the transmission of the sensor information to the PLC via an IO Link

communication instead of the Odroid. This way reliable and deterministic data exchange is possible as well as a relief to the Odroid computation power.

Overview of the AGV communication

After deriving the communication buses and methods to be used, the figure below shows the system diagram derived from this, which connects the individual components with their associated information interfaces.

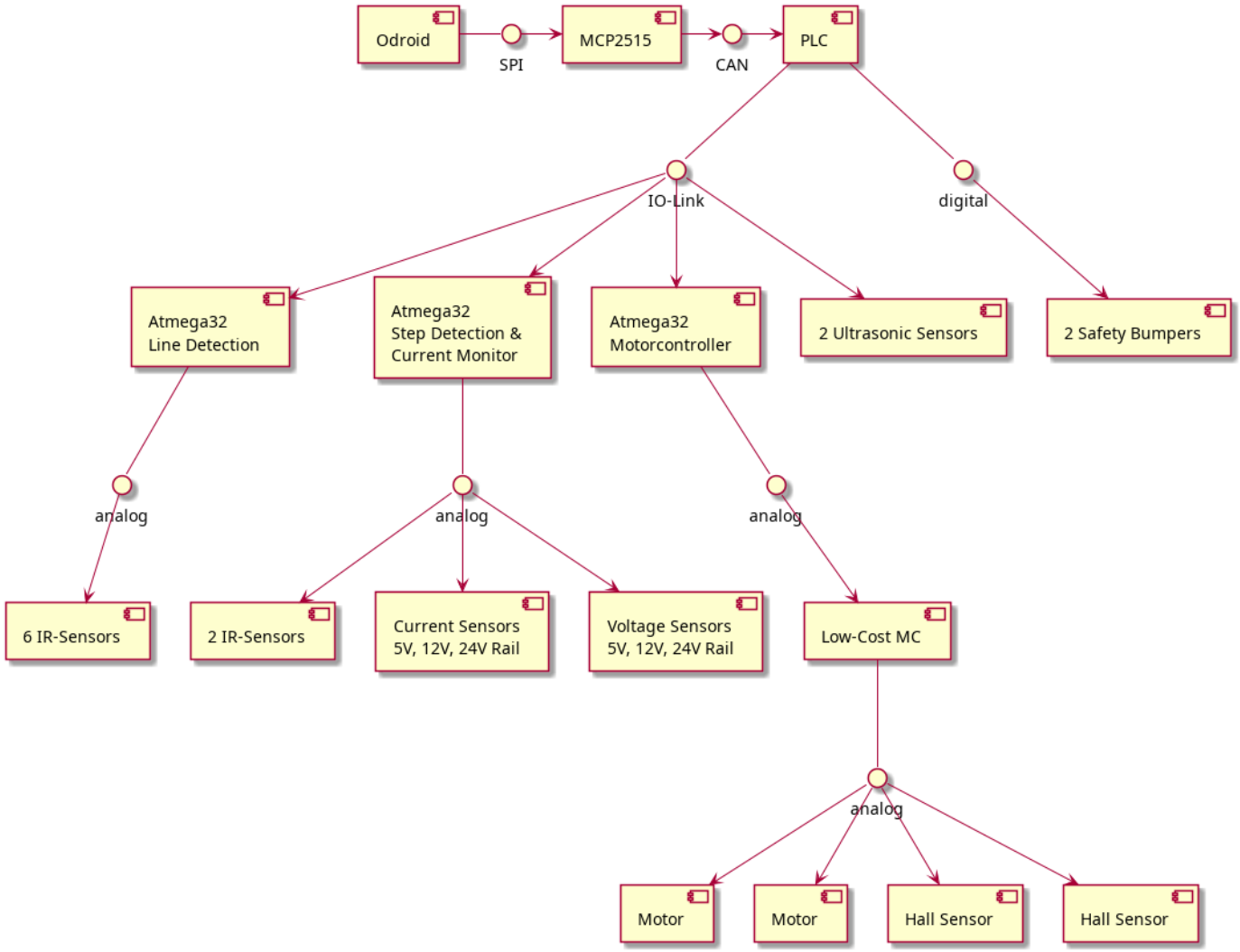


Figure 117 System diagram

Communication Definition for the CAN Protocol

Once the communication between Odroid and PLC has been set to a CAN network, the need to define a protocol follows. To make the implementation as simple as possible, the CANopen protocol was not used and instead an own protocol was defined, which is presented below.

Definitions

• Motor 1 is the motor mounted on the right side of forward driving direction of the AGV.

• Motor 2 is the motor mounted on the left side of forward driving direction of the AGV.

• Infrared/Ultrasonic sensor indexing starts at the rightmost sensor and proceeds to the left side of forward driving direction of the AGV.

Parameters

• Bus speed is fixed to 500 kbit/s

• 11-bit identifier format (CAN 2.0A)

• Byte order of all Signals is Intel (little-endian)

• Used frames are

    o Data Frames

    o Remote Frames

Nodes

The CAN network consists of only two nodes, the PFC200 and the Odroid N2+.

Messages

***Note**: These messages are call & response. The Odroid sends a message with the RTR bit set, and the PLC responds with the same ID and specified payload.

****Note:** This cyclic message is sent by the PLC on a timer without a request ever 500ms.

ID	Name	Sender	Signals	Start bit	Signal type	Bits	Factor	Offset	Min	Max	Unit
0x100	Heartbeat	PLC**	Heartbeat	0	Uint	8	1	0	1	1	-
0x101	IR Line Detection	Odroid*	IR_1_1	0	Uint	8	1	0	0	1	bool
			IR_1_2	8							
			IR_1_3	16							
			IR_1_4	24							
			IR_1_5	32							
			IR_1_6	40							
0x102	IR Step Detection	Odroid*	IR_2_1	0	Uint	8	1	0	0	1	bool
			IR_2_2	8							
0x103	US Object Detection	Odroid*	US_1	0	Uint	16	1	0	0	1	-
			US_2	16							
0x104	Power Monitor 24V	Odroid*	PWR_SENS_24V_V	0	Uint	16	1	0	0	65535	mV
			PWR_SENS_24V_I	16							mA
0x105	Power Monitor 12V	Odroid*	PWR_SENS_12V_V	0	Uint	16	1	0	0	65535	mV
			PWR_SENS_12V_I	16							mA
0x106	Power Monitor 5V	Odroid*	PWR_SENS_5V_V	0	Uint	16	1	0	0	65535	mV
			PWR_SENS_5V_I	16							mA
0x112	IR Step Calibration	Odroid	IR_STEP_CALIB	0	Uint	8	1	0	0	255	-
0x201	Motor Enable	Odroid	MTR_EN_1	0	Uint	8	1	0	0	1	bool
			MTR_EN_2	8							
0x202	Motor Speed	Odroid	MTR_SPEED_1	0	Int	16	1	0	-1000	+1000	-
			MTR_SPEED_2	16							
0x301	PLC Status	Odroid*	PLC_STATUS	0	Uint	8	1	0	0	255	-

Table 11 Messages sent by the PLC

Communication Definitions for the IO Link motor controller

This section describes the communication between the PLC and the IO-Link motor controller. It is closely related to the communication between PLC and Odroid via a CAN bus in terms of information to be exchanged.

Definitions

•Motor 1 is the motor mounted on the right side of forward driving direction of the AGV.

•Motor 2 is the motor mounted on the left side of forward driving direction of the AGV.

•PLC is the IO-Link Master.

•Motor Controller (MC) is the IO-Link Slave.

•Process Data Input (PDin) is the input process data from the PLC (master) view.

•Process Data Output (PDout) is the output process data from the PLC (master) view.

•Data is sent in big-endian byte order.

Messages

The PDout of the IO-Link consists of 7 bytes.

Within the PDin 3 message types are being sent to the MC:

- MTR_n_ENABLE
- MTR_n_SPEED
- MTR_ESTOP

The PDin of the IO-Link consists of 6 bytes.

3 message types are being sent from the MC:

- MTR_ENABLE_STATUS
- MTR_n_SPEED_STATUS
- MTR_ESTOP_STATUS

(n is a placeholder for the index of a Motor)

Messages to MC

In the following the structure of the message types is defined.

MTR_n_ENABLE

With this message Motor 1/2 is being activated / deactivated.

- PDin 1 Byte size
- data type uint_8
- Boolean value
- 1 means the MC should be activated
- 0 means the MC should be deactivated
- deactivation stops the motor thus any previous message of type MTR_N_SPEED has no effect any longer
- deactivation sets the current MTR_N_SPEED values to 0

MTR_n_SPEED

With this message Motor 1/2 speed values are set.
Prerequisite: Motor n has to be activated with the MTR_n_ENABLE message first.

MTR_n_SPEED							
Byte 1 (High Byte)							
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DIR	x	x	x	x	x	SPEED_HB	

Byte 2 (Low Byte)							
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
SPEED_LB							

x indicates a reserved bit.

- PDin **2 Bytes** size
- data type uint_8

- DIR (Byte 1, Bit 7) being 1 indicates a **counterclockwise** rotation of the wheel
- DIR (Byte 1, Bit 7) being 0 indicates a **clockwise** rotation of the wheel
- Speed values are **unitless**
- Speed values range from **0 to 1000 to map the entire range**

MTR_ESTOP

With this message both motors are shut down as fast as possible. It is used only if the PLC detects a safety hazard.

- PDin **1 Byte** size
- Data type uint_8
- **Boolean** value
- 1 means ESTOP should be **activated**
- 0 means ESTOP should be **deactivated**

NOTE*:* This message is implemented for debugging purses. A proper emergency stop that complies with safety standards has to be implemented on a different way.

Messages from MC

MTR_ENABLE_STATUS

With this message the current values stored in the MC corresponding to **MTR_n_ENABLE messages are transmitted.**

- PDout **1 Byte** size
- Data type **uint_8**
- sent periodically every 1sec
- possible values are:

MTR_ENABLE_STATUS	Motor 1	Motor 2
0	Disabled	Disabled
1	Enabled	Disabled
2	Disabled	Enabled
3	Enabled	Enabled

MTR_n_SPEED_STATUS

With this message the current value stored in the MC corresponding to MTR_n_SPEED message is transmitted.

- PDout 2 Byte size
- data type uint_8
- sent periodically every 1sec
- content identical to message type MTR_n_SPEED

MTR_ESTOP_STATUS

With this message the current value stored in the MC corresponding to MTR_ESTOP message is transmitted.

- PDout 1 Byte size
- Data type uint_8
- sent periodically every 1sec
- possible values are:

MTR_ESTOP_STATUS	ESTOP
0	Disabled
1	Enabled

ROS CAN Communication Handler

After the information exchange in the CAN network has been defined in the previous chapter, the implementation on the side of the ROS system follows in this chapter. For this purpose, the ROS package "can_handler" was developed and is available in the project repository on GitLab.

Since the Odroid N2+ itself does not have its own CAN controller and even CAN transceiver, a CAN extension is done via a small breakout board of the CAN controller module MCP2515, which communicates with the Odroid via an SPI interface. The software and hardware installation of the module into the Linux system is described in detail in the first chapter of this report.

In order to correctly map the information model with its IDs and signals, a CAN database file, or DBC file for short, was first created using the CANdb++ editor from Vector. The use of such a database file allows fast encoding and decoding of CAN messages and easy subsequent adaptation of individual messages without the need for major changes to the software. It is stored in the include directory of the ROS CAN handler.

In order for the ROS system to receive PLC status data as well as sensor data, CAN data frames must first be requested with a remote frame, as already described in the previous chapter, due to the Call & Response method. This allows a simple adjustment of the data rates and thus bus load during the test phase. With the help of the Python modules "can" and "cantools" the CAN controller MCP2515 is made available as a communication interface and received and to be sent messages are coded or decoded together with the .dbc database and then sent. For this the CAN communication handler opens different topics in the ROS system under the topic address /can_msgs/*. After receiving new sensor data, for example, the CAN communication handler publishes them already decoded under the corresponding topic /can_msgs/rx/*. Other nodes in the ROS system can subscribe to the topic and thus always receive the current data. The same applies to the sending of CAN messages. For this the Topics /can_msgs/tx/* are available. For example, another node can send new motor speeds to the PLC by publishing new speed values in [%] on the topic /can_msgs/tx/mtr_speed. The CAN communication handler encodes these accordingly and sends them to the MCP2515 module. All currently available topcis are shown in the figure below. The program code of the CAN communication handler is available on GitLab in the corresponding repository. There one can also find all necessary information for installation and usage.

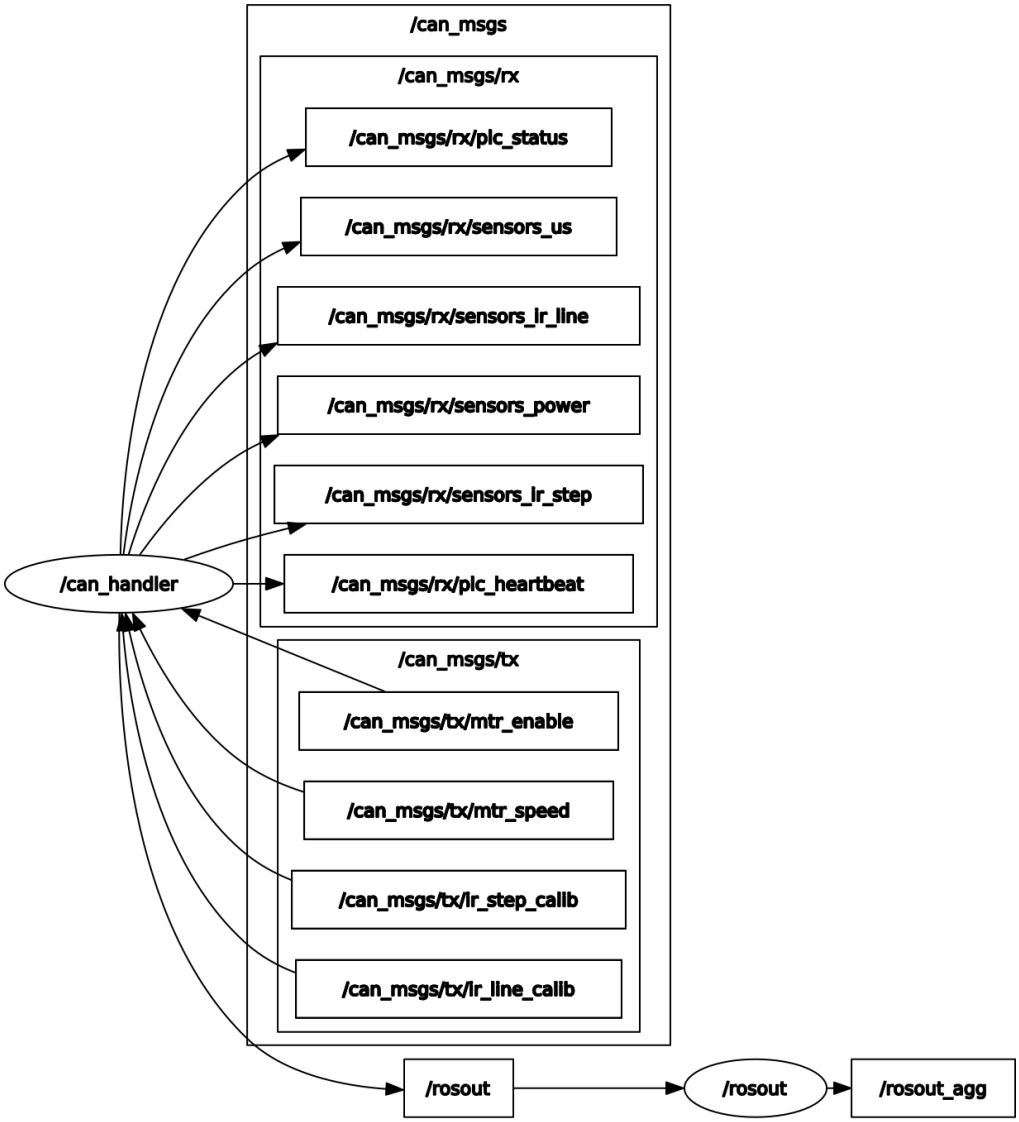


Figure 118 currently available topics

ROS Line Follower State Machine

When it comes to ROS, as the lead system component, there must also be a state machine that contains all the logic of the AGV and takes over its control. For this purpose, the ROS package "line-follower" was developed and is available in the project repository on GitLab.

Since a simple line follower is intended as the first implementation phase, the state machine consists of only a few states, which are shown in the figure below.

Despite the low complexity of the state machine, the ROS package is based on the ROS package "smach", which enables the implementation of complex state machines. In this way, the line

follower can later be extended by further states.

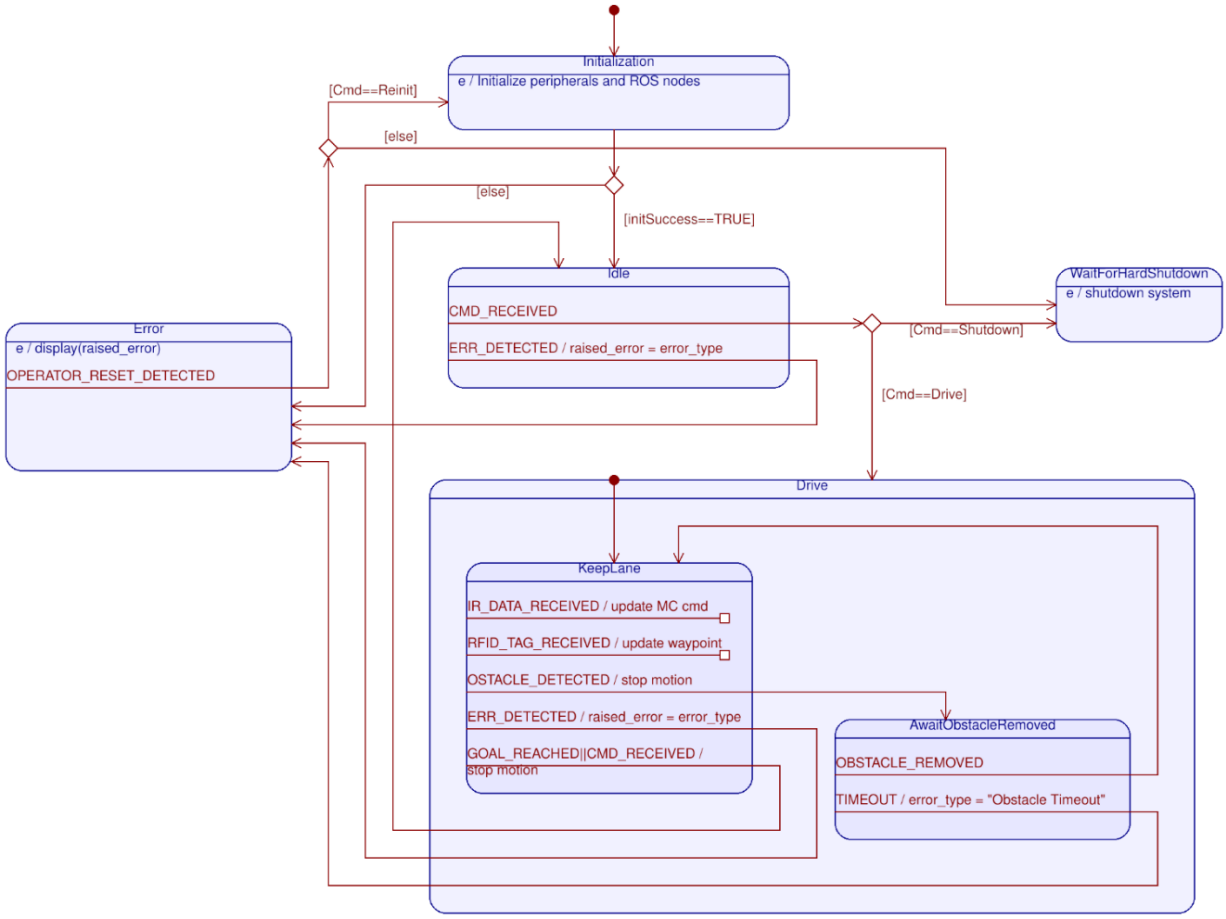


Figure 119 System State Machine

If the line-follower package is started with the launch file, then the ROS system can be mapped with the computation graph in the figure below. In addition to the can_handler, the line_follower node is started, which takes over the control of the AGV. For this purpose, it subscribes to the corresponding sensor topics and evaluates their data in order to then publish appropriately adapted motor values via the responsible topics.

In addition, the computation graph also shows the /user_ctrl/* topics. Via /user_ctrl/set_goal an existing target must be published first. These are currently still stored in the line-follower package and should be read in as ROS parameters via a YAML file in a later step.

A target station can be defined in the current configuration via the terminal input

$ rostopic pub /user_ctrl/set_goal std_msgs/String A -1

can be done.

The same applies to the /user_ctrl/cmd topic, which can be used to invoke the guards [Cmd] shown in the state diagram. For example, to start the AGV, the terminal command is entered:

$ rostopic pub /user_ctrl/cmd std_msgs/String Drive -1

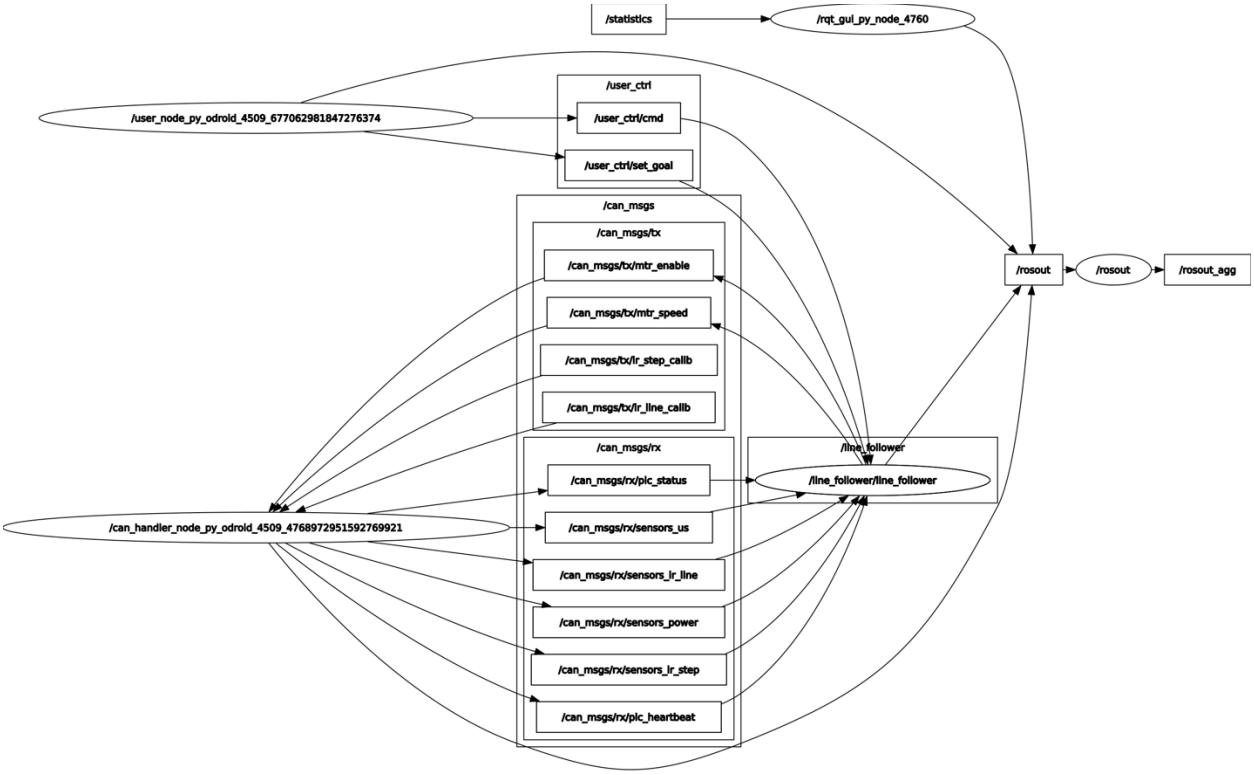


Figure 120 Line follower

2.3 Connections for Supply

The following diagram (Figure on the next page) will be used for power supply and voltage division to different sections of the system.

This uses sealed Lead Acid batteries 12V 7Ah connected in series to provide an output of 24V. To increase current output, 2 sets of such series connection will be required. These 4 batteries will be referred as battery pack in this context. The wire used for interconnecting these batteries will be of 6 mm^2 to allow proper current flow and avoid heating of wires during sudden spike in current flow. The batteries will be connected to Battery Equalizer to keep all of them at about the same voltage.

To charge the batteries, Adaptor (24V) will be used which will directly be connected to the battery pack. There will be a switch provided which will disconnect the system from the battery pack. This is provided to disable the system when charging and also when the system will be out of service for long durations. The wire here will also be 6 mm^2.

The output from battery pack is supplied through terminal block to respective hardware via fuse. Fuse is provided to avoid damage to hardware due to abnormal current surges or short-circuit. The value for fuses used will be about 1.25 % greater than the max current of the hardware of respective sections (2).

The DC/DC converters are selected such that they work in the permissible limits of 9 – 36V and are on DIN rail convertors so no external housing will be needed for the same.

These are the sections in which the supply will be distributed,

- 24 V (without DC/DC converter) – wire for connection used 6mm2 (uninterrupted current flow)
  1) Motor Driver
  2) Motors
  3) Voltage Sensor (voltage measurement)
  4) Fan

- 24V (with DC/DC converter) – wire for connection 0.75 mm^2
  1) PLC
  2) IFM Camera
  3) 24V Sensors (if required)

- 12V (with DC/DC converter) – wire for connection 0.75 mm^2
  1) Odroid N2+

- 5V (with DC/DC converter) – wire for connection 0.75 mm^2
  1) Arduino Nano
  2) 5V Sensors

Separate current sensors are planned for all supply voltages to monitor individual section.

PHOTO FROM CAMERA

For terminal block, we can use the below block.

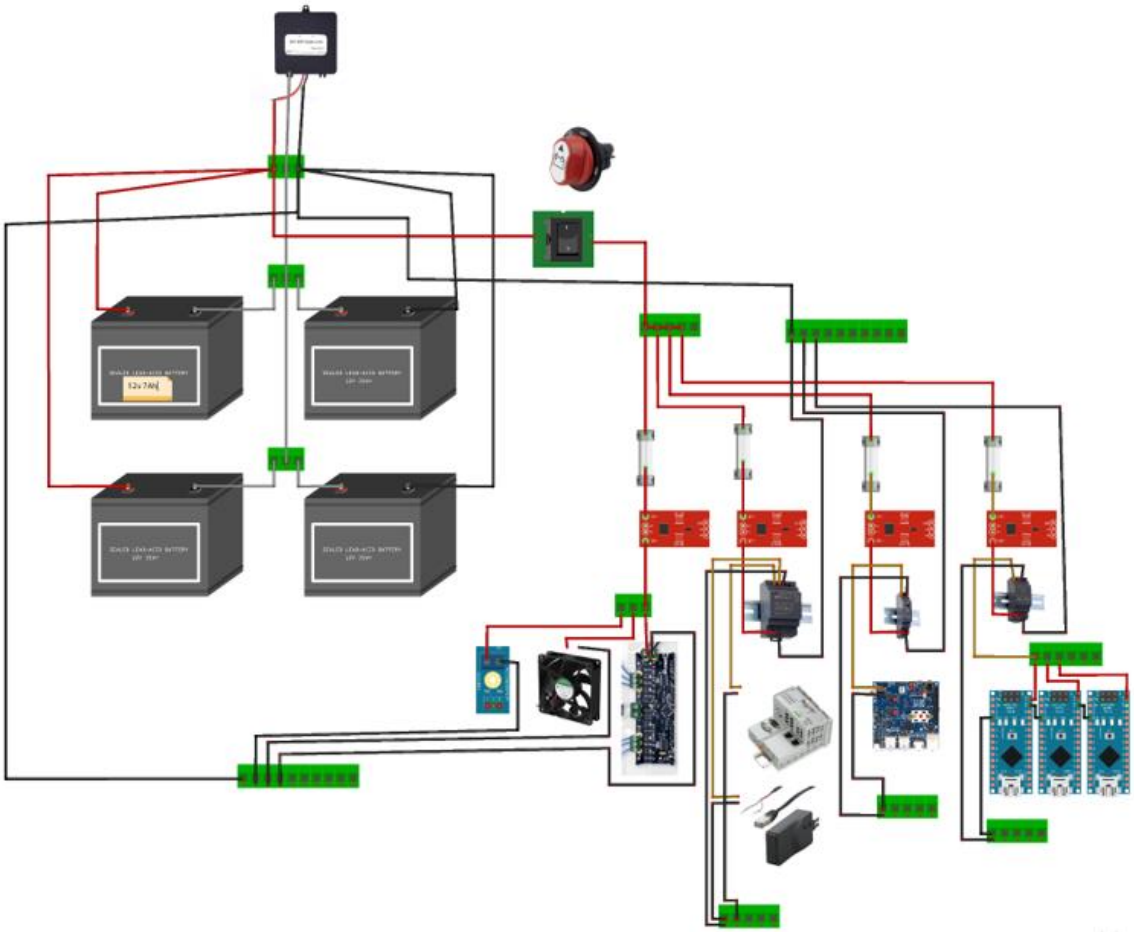


Figure 121 Power Supply distribution

2.4 Power Monitoring sensors

2.4.1 Current sensor:

For measuring current in a circuit, a sensor is required. ACS712 Current Sensor is the sensor that can be used to measure and calculate the amount of current applied to the conductor without affecting the performance of the system.

ACS712 Current Sensor is a fully integrated, Hall-effect based linear sensor IC. This IC has a 2.1kV RMS voltage isolation along with a low resistance current conductor.

Figure 122 Current Sensor

Working Principle:

Current Sensor detects the current in a wire or conductor and generates a signal proportional to the detected current either in the form of analog voltage or digital output.

Connections:

Three pins at the end of the sensor are connected to our Arduino. VCC to 5Volts, Output to analog3 pin and Ground to the GND of Arduino.

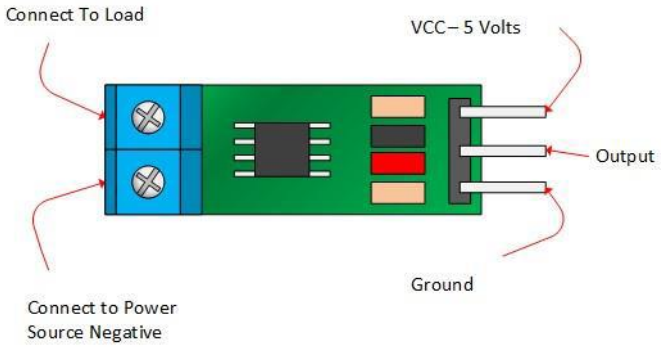
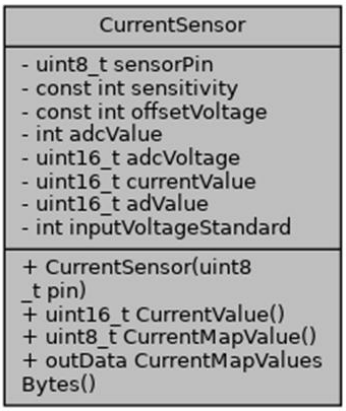


Figure 123 ACS712 Current Sensor connection pins



2.4.2 Voltage sensor

A voltage sensor is a sensor used to calculate and monitor the amount of voltage in an object. Voltage sensors can determine the AC voltage or DC voltage level. The input of this sensor is the voltage, whereas the output is the switches, analog voltage signal, a current signal, or an audible signal.

This sensor is fundamentally a 5 to 1 simple voltage divider using a 30K and a 7.5K Ohm resistor. Based on that, the voltage can be measured, and the output can be taken in one of many forms.

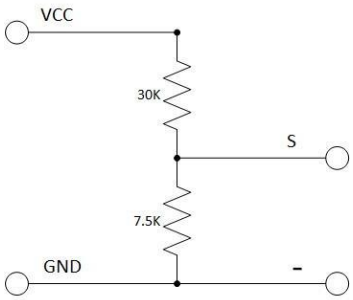


Figure 124 Voltage Sensor

The Arduino analog input is limited to a 5 VDC input. If you wish to measure higher voltages, you will need to resort to another means. One way is to use a voltage divider.

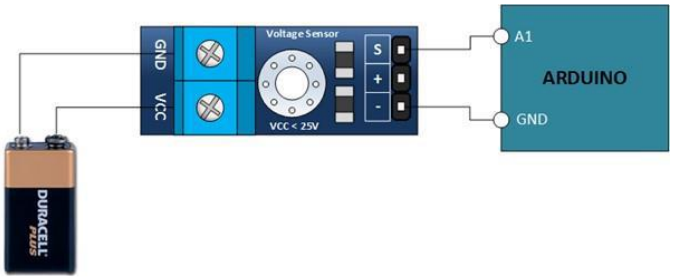


Figure 125 Voltage sensor Arduino connection

Circuit:

The sensor is connected to the analogue pins of the Arduino through a Zener diode. Zener diode is a type of diode that allows current to flow not only from its anode to its cathode, but also in the reverse direction, when the Zener voltage is reached. We used the Zener Diode in this circuit to protect the analogue pin. The break down voltage for this diode is 5 volts. So, when it is reached, circuit is short circuited, so the Arduino pin does not get damaged**.**

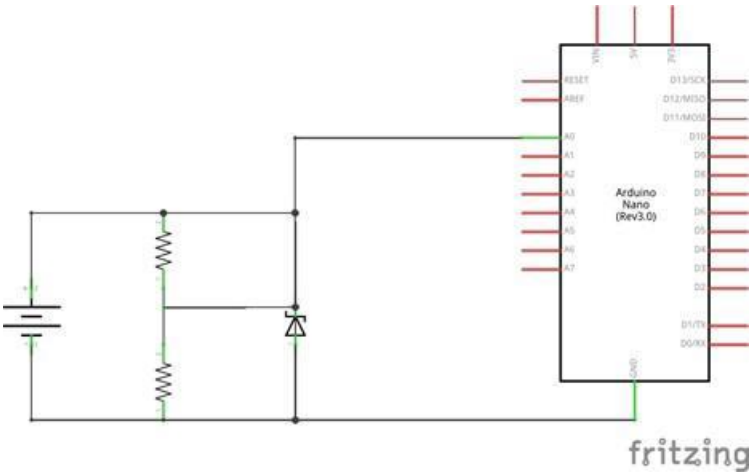
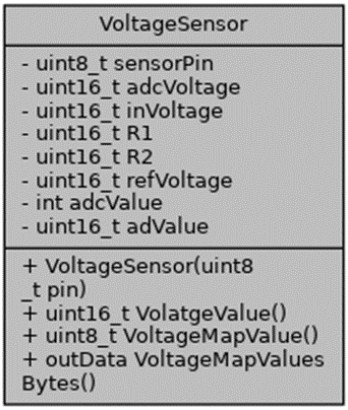


Figure 126 Voltage sensor schematic



2.5 IO LINK Master

The IO-Link masters for field applications are gateways that allow up to eight IO-Link devices, such as sensors, valves, or binary input/output modules, to be connected. Some versions include ports that are meant as B ports and provide supplemental energy for IO-Link actuator connections.

Figure 127 IO Link Master

IO Link Shield

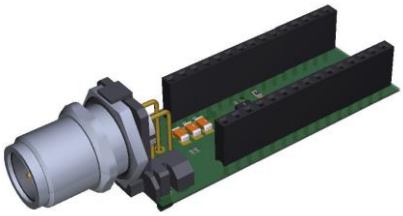


Figure 128 IO-Link Shield for Arduino

Since there are already development board kits that include the ATMEGA328p microcontroller for the Arduino framework, only the IO-Link transceiver with the appropriate setup was designed.

The transceiver used for the physical communication layer is the TIOL111-5 from Texas Instruments; the manufacturer indicates that the following considerations should be considered for the correct operation of the device:

- Each digital TTL output (WAKE and NFAULT) should have a 10 KΩ pull-up resistor.
- Minimum 100 nF capacitor between L+ and L-
- Minimum 1 µF capacitor between VCC_IN/OUT and Figure

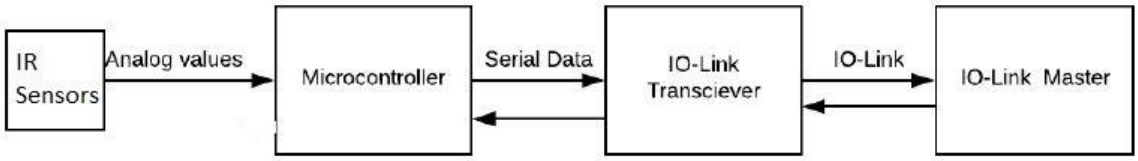


Figure 129 Block Diagram of IO-Link Communication for Sensors

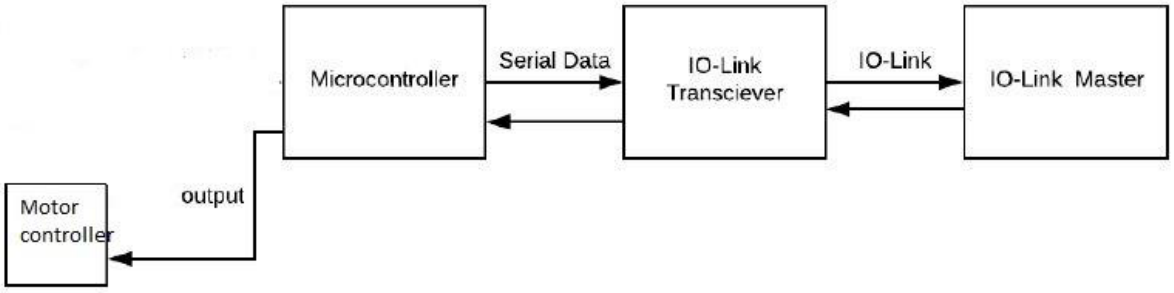


Figure 130 Block Diagram of IO-Link Communication for Microcontroller

The transceiver exchanges serial data with the microcontroller as shown and further sends the information to the IO-Link Master. To check the io link codes easily, the IFM IO Link master can also be used instead of the PLC.

To check the codes on the Master, the Python 3.10 IDE must be installed.

2.6 Motor controller

The data sent from the PLC is received by the IO-Link shield. The 7 bytes of data received is structured in the following order:

MTR_ESTOP, MTR_1_ENABLE, MTR_2_ENABLE, MTR_1_SPEED_H, MTR_1_SPEED_L, MTR_2_SPEED_H, MTR_2_SPEED_L.

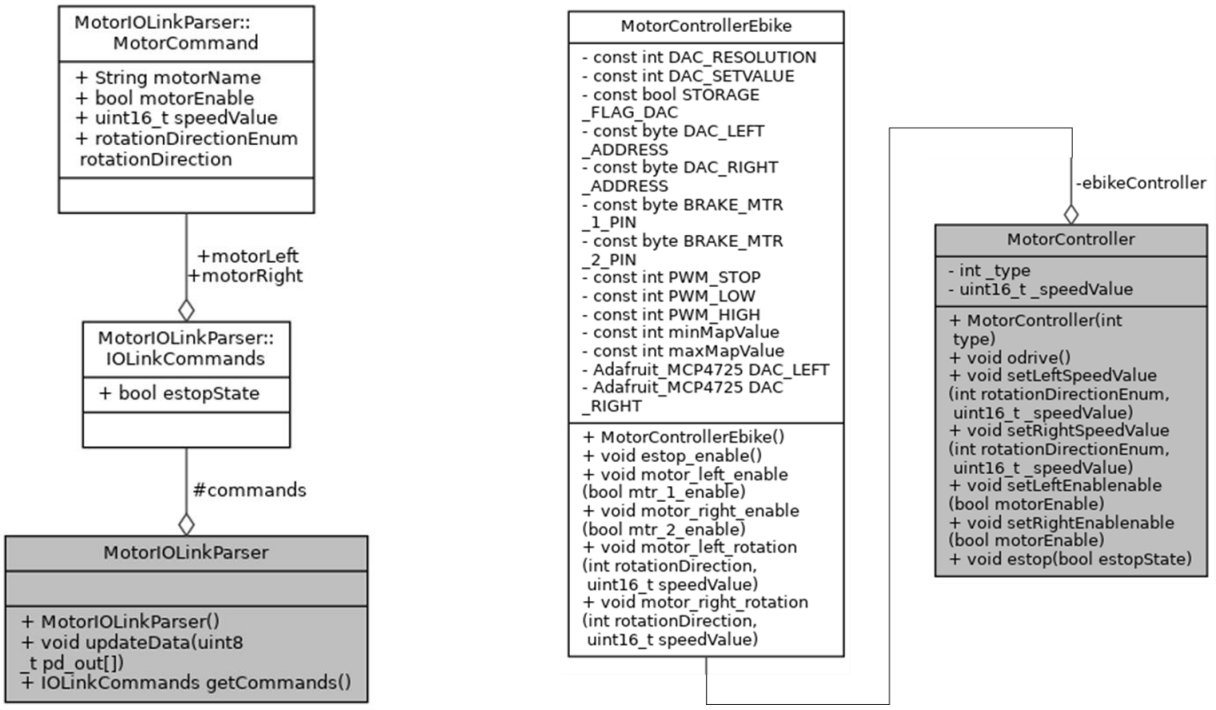
The Arduino nano also sends feedback to the PLC via IO-Link communication. The data is of 6 bytes structured in the following order:

MTR_ESTOP_STATUS,MTR_ENABLE_STATUS,MTR_1_SPEED_STATUS_H,MTR_1_SPEED_STATUS_L, MTR_2_SPEED_STATUS_H, MTR_2_SPEED_STATUS_L.

The above mentioned CAN messages (table 12) are used as the following in the code.

Signal (PLC to IO-Link)	Code reference (arduino code)
MTR_ESTOP	MTR_ESTOP
MTR_EN_1	MTR_1_ENABLE
MTR_EN_1	MTR_2_ENABLE
MTR_SPEED_1	MTR_1_SPEED_H
	MTR_1_SPEED_L
MTR_SPEED_2	MTR_2_SPEED_H
	MTR_2_SPEED_L

Signal (PLC to IO-Link)	Code reference (arduino code)
MTR_ESTOP_STATUS	MTR_ESTOP_STATUS
MTR_ENABLE_STATUS	MTR_ENABLE_STATUS
MTR_1_SPEED_STATUS	MTR_1_SPEED_STATUS_H
	MTR_1_SPEED_STATUS_L
MTR_2_SPEED_STATUS	MTR_2_SPEED_STATUS_H
	MTR_2_SPEED_STATUS_L



2.6.1 Motor control using ebike controller

Ebike controller is controlled using analog signal. The controller responds to analog signal between the range 1.2v to 3.8v. To supply analog output, DAC MCP4725 is used. It is connected to Arduino nano using I2C communication.

MCP4725 is a breakout board from Adafruit. It has a default I2C address of 0x62, but it can be changed by connecting A0 pin to High to change the address to 0x63. This allows connecting 2 DAC modules to the same microcontroller. To control MCP4725 Adafruit provides Arduino library defining all parameters allowing a plug and play usage.

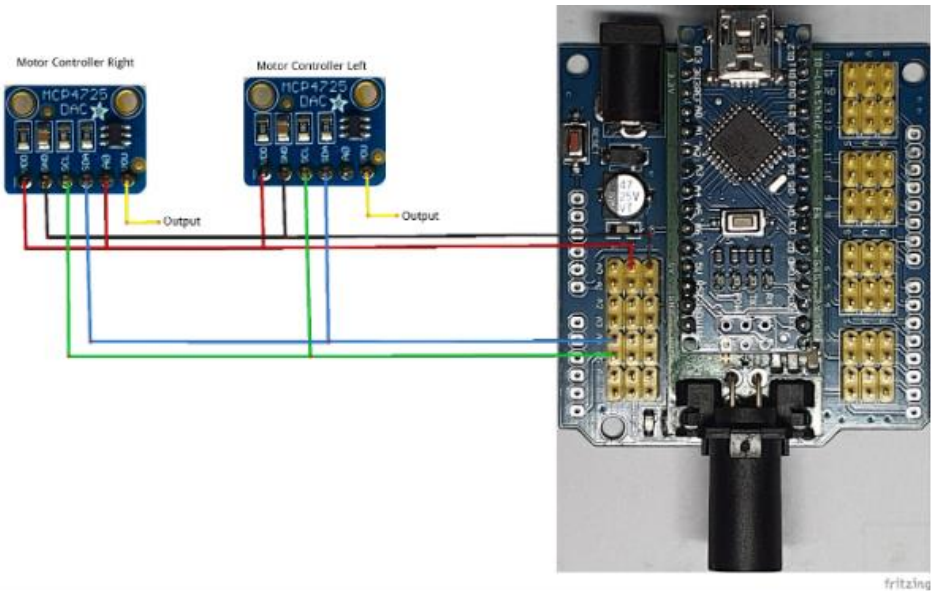


Figure 131 Connection diagram from IO Link to Ebike controller

For the project's application, the MCP4725 receives commands from PLC via IO-Link Shield. The shield mounts Arduino nano to which the MCP4725 module is connected. The output (VOU pin) is connected to the respective motor controller.

Motor Controller Right	Motor Controller Left	Arduino Nano
VCC	VCC	+5V
GND	GND	GND
SCL	SCL	A5
SDA	SDA	A4
A0	-	+5V

2.6.2 Odrive controller

The project uses hoverboard BLDC motors. Odrive motor controller allows controlling two BLDC motors simultaneously using one board. The commands can ve sent via various communication protocols such as UART, CAN and USB. The motors will be connected in similar manner. BLDC motors consists of hall sensors which are connected to ODrive board.

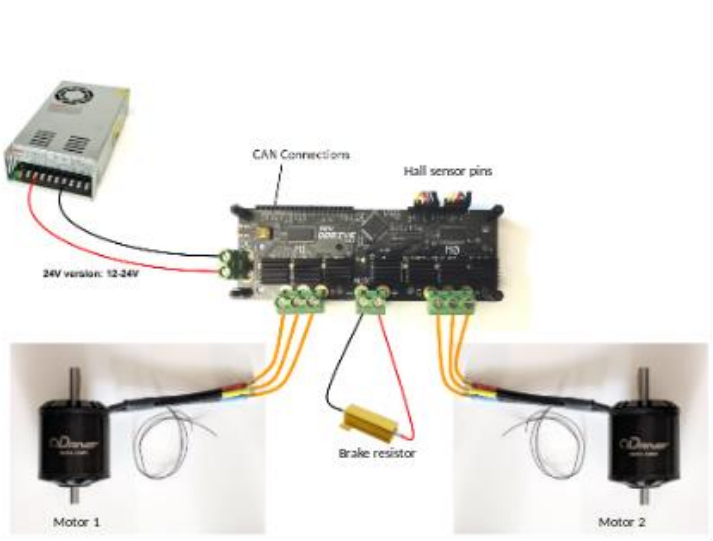


Figure 132 Connections with odrive controller

The calibration of motor controller can be done using the script found at the following link: https://git.fh-aachen.de/mover2021/io-link/motor-controller/odrive-cfg.

The parameters for the current BLDC motors are predefined in the script.

Current Status:

The motor calibration is not working properly. Various parameters which depend on BLDC motor were not known hence correct diagnostic could not be done. The hall sensor mode calibration gives errors which when diagnosed gets cleared but the motors still did not rotate. In sensor less mode, the command intended to spin the motors continuously did not spin it more than 1 complete rotation.

2.7 Functions performed by the Mover

The various functions and tasks that are performed by the mover are as follows.

2.7.1 Line Sensing

IR Sensor:

The basic function that is to be performed by the Mover is to follow the line that extends towards all the places and corners that the mover has to travel.

For this purpose, the Iduino IR sensors are used. The Iduino IR sensors are connected to the PLC via the Arduino Nano, and it sends the digital readings to the Nano which further transmits it to the PLC.

Figure 133 IDUINO IR Sensor

In the system, there are six IR sensors connected to an Arduino Nano via the Expansion Board. These six IR sensors are coded to follow the black line in front of it.

Calibration:

For the calibration of the IR sensor, an analog read of the IR sensor using the arduino nano is done.

The schematic diagram for the connection of the IR sensor along with the Arduino Nano is as follows:

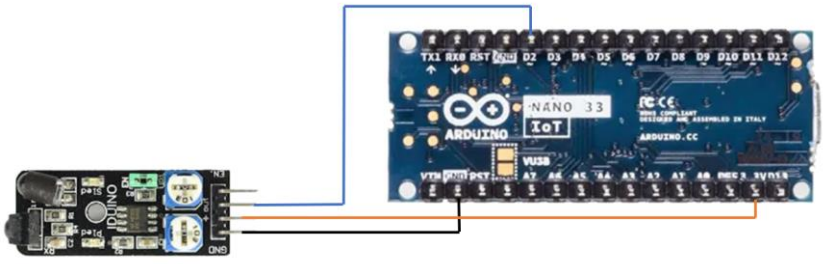


Figure 134 IR sensor to Arduino Nano Schematic Diagram

The enable pin on the IR sensor is not used here.

Once the calibration of the 6 IR sensors is done, the Arduino Nano needs to be connected to an expansion board. The expansion board helps in connecting multiple sensors to the Arduino Nano at the same time.

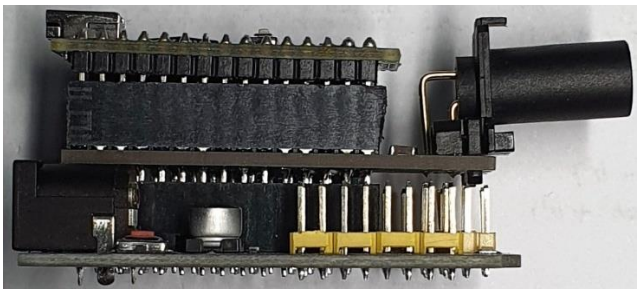


Figure 135 Arduino Nano connected to IO link shield and expansion board

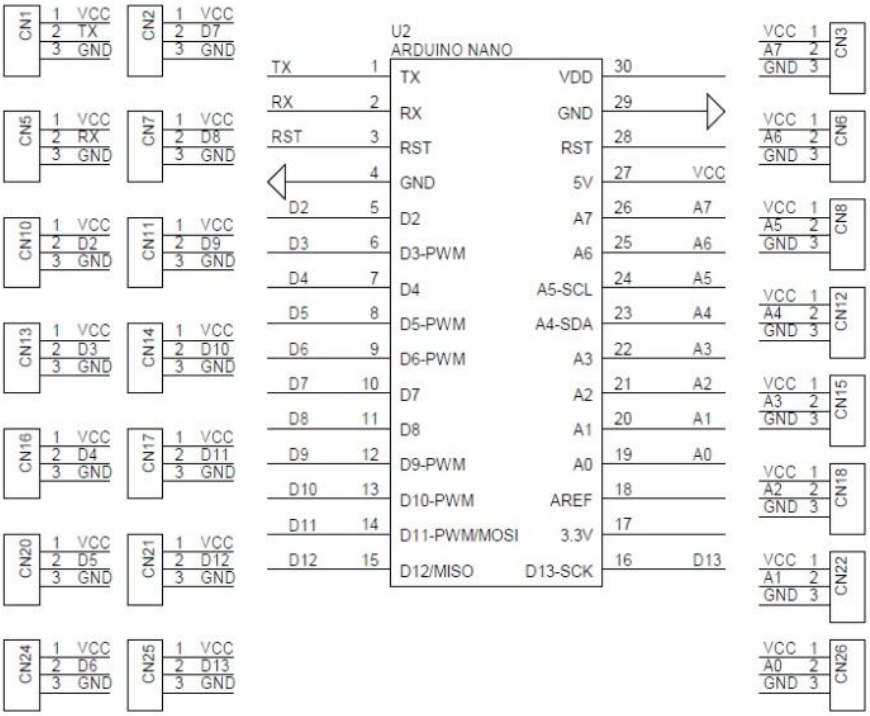


Figure 136 Schematic of the Expansion Board

To connect all the six IR sensors, the nano is fixed on the expansion board and the IR sensors are connected to it. The diagram below shows the connections between the IR sensors and expansion board assuming that the Arduino Nano is on top of it.

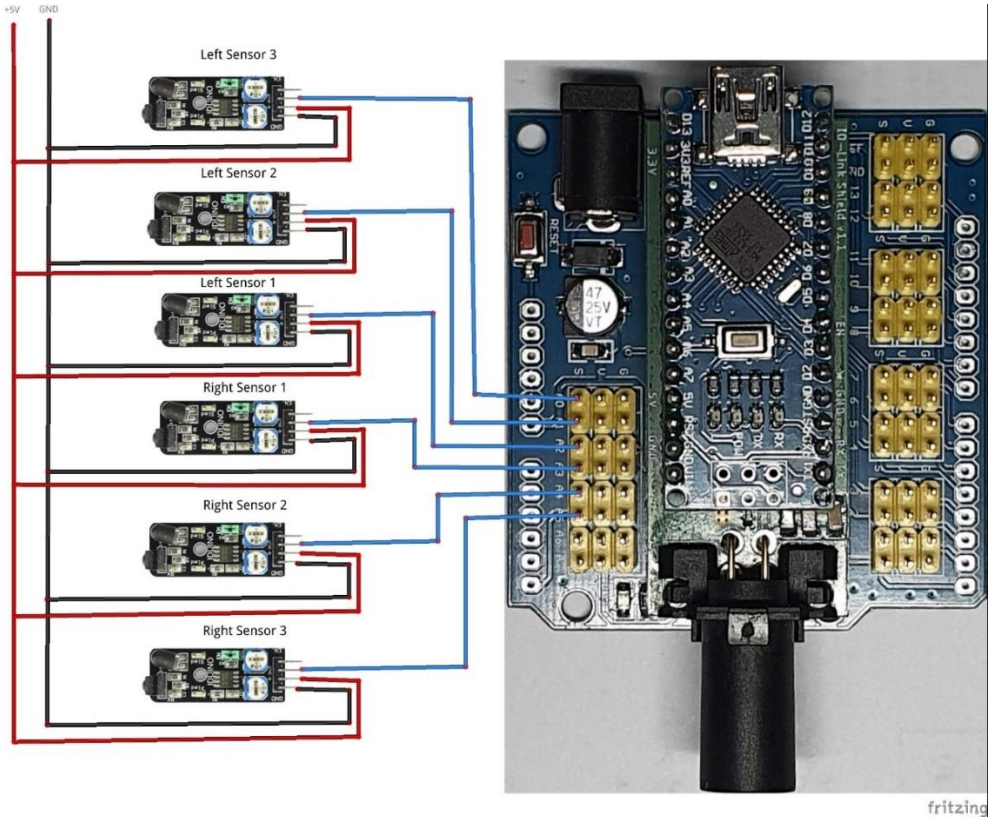


Figure 137 IR sensor to Arduino Nano schematic diagram

The power supply to the Arduino Nano is received from the PLC through the IO Shield but this power when transferred to the six IR sensors would not provide with enough power to the individual IR sensor. So, we need to provide another power supply for the six IR sensors to function properly.

Now this Arduino Nano is connected to the PLC to send the readings to the Odroid. The communication between the Arduino Nano and the PLC is via IO Link. To check the communication between the PLC and the Arduino Nano, a code was written.

The code is supposed to send a specified value to the PLC to make sure the IO Link communication works error-free.

Each IR sensor sends 2 Bytes of information. So, in total 6 IR sensor sends 12 Bytes of information.

When the IR sensor senses a black line, it will send a value of digital high.

So, when a value of 999 is sent, it will be 11 1110 0111. The PLC is programmed to receive 2 Bytes of data from each sensor. This information will be split into 1 Byte each sending 00000011 (decimal form: 3) and 11100111 (decimal form: 231).

IrSensorAnalog
- uint8_t sensorPin - uint16_t adcValue
+ IrSensorAnalog(uint8 _t pin) + uint16_t IrSensorAnalogValue() + uint8_t IrSensorMapValue() + outData IrSensorValuesBytes()

Run the following code on Arduino IDE to program the Arduino Nano to send the preset values.

To refer to the code, refer the following link.

https://git.fh-aachen.de/mover2021/io-link/linefollower-ir-sensors/-/blob/st9223s-main-patch-10189/ir_sensor_6_input/ir_sensor_6_input.ino

PLC to Arduino connection:

Arduino:

Once the programming of the Arduino is done, **remove the micro-USB from the arduino Nano.**

Connect the arduino with the arduino shield. If arduino nano is used, it can be connected directly but in case of using an Arduino Uno, the following connections are used**.**

IO LINK Shield v1.2:

Connect the arduino with the IO Link shield using jumper wires.

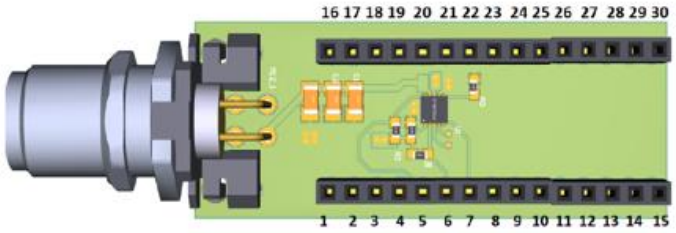


Figure 138 Pinout numbering from top view of Arduino IO link shield

IO-Link Shield Pin number	Connection with Arduino Pin
1	TX
2	RX
4	GND
5	D2
7	D4

Figure 139 Required IO Link connections between the shield and Arduino Uno

IO Link Module:

Once the Shield and the Arduino has been connected, connect the shield to the PLC using the M12 cable.

PLC IP Address:

The IP address of the PLC can be found using the “Wago ethernet settings” app after changing the Ip address of the system in the same range using the control panel.

It was found to be 192.168.1.18.

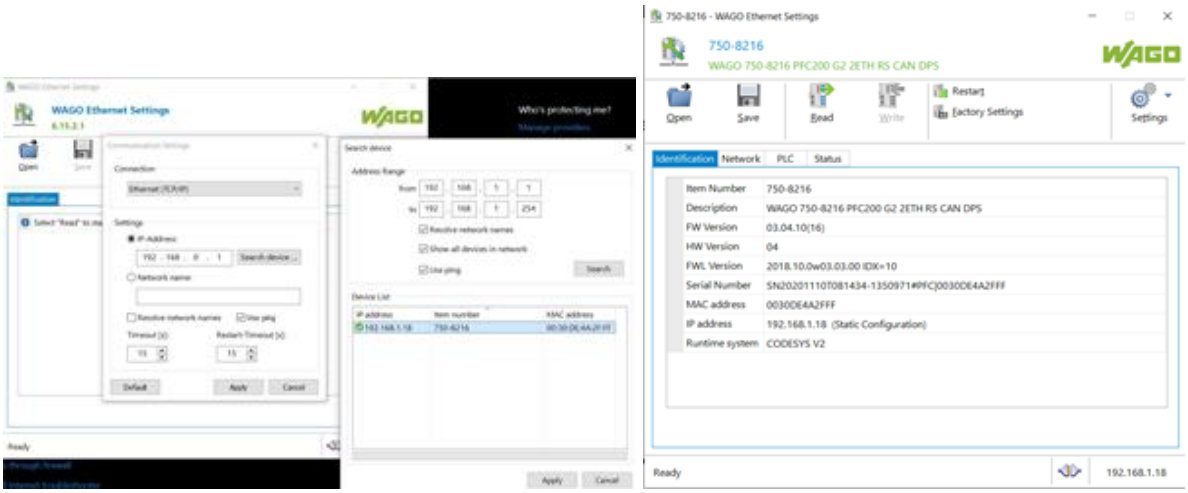


Figure 140 WAGO Ethernet Settings

Wago IO check 3:*Once Wago is connected, click the Identify button.*

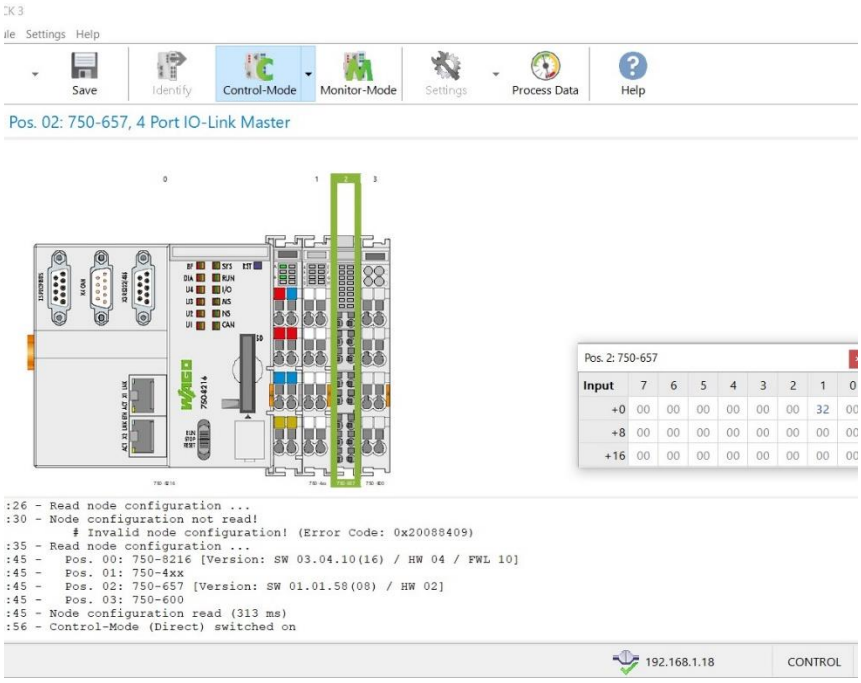


Figure 141 IO Link Module on WAGO IO Check 3 Software

When no sensor is connected, this is what appears in the process data.

The web visualization can be seen by using the IP address of the PLC.

Username: admin

Password: wago

In the Wago IO check software, select settings>communication and type the IP address of the PLC. Click Identify. Then select the settings of the particular module to change the settings.

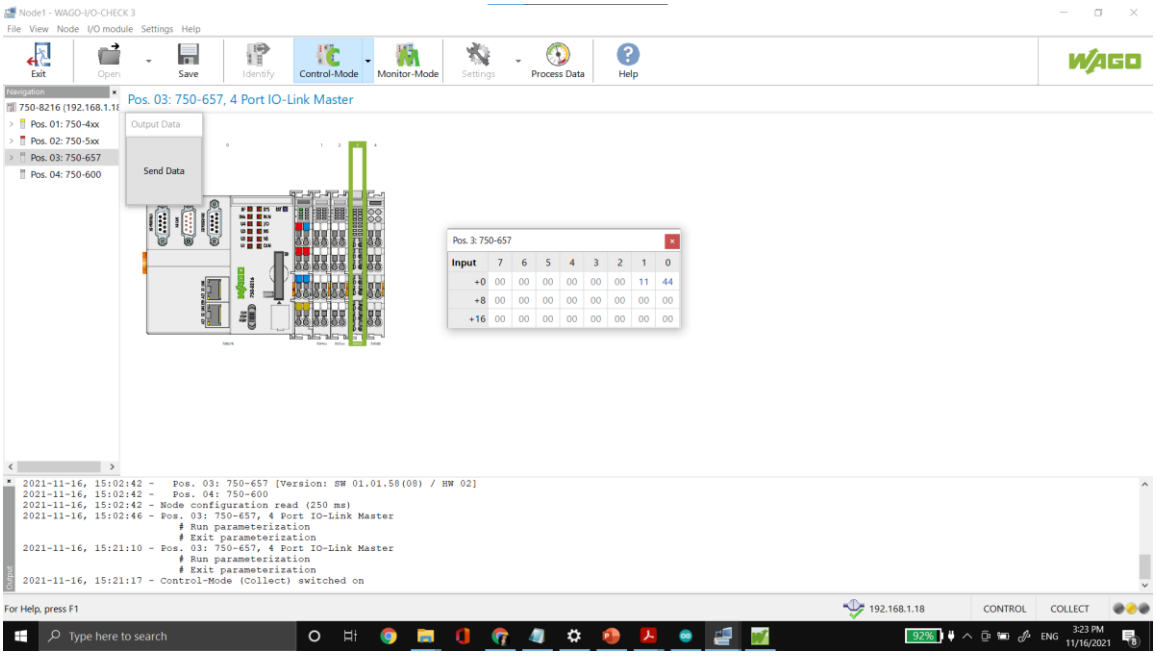


Figure 142 Reading when no sensor is connected

e!COCKPIT software:

The PLC is scanned using the scan option in the network tab.

NOTE: Make sure to change the IP address range in the settings option.

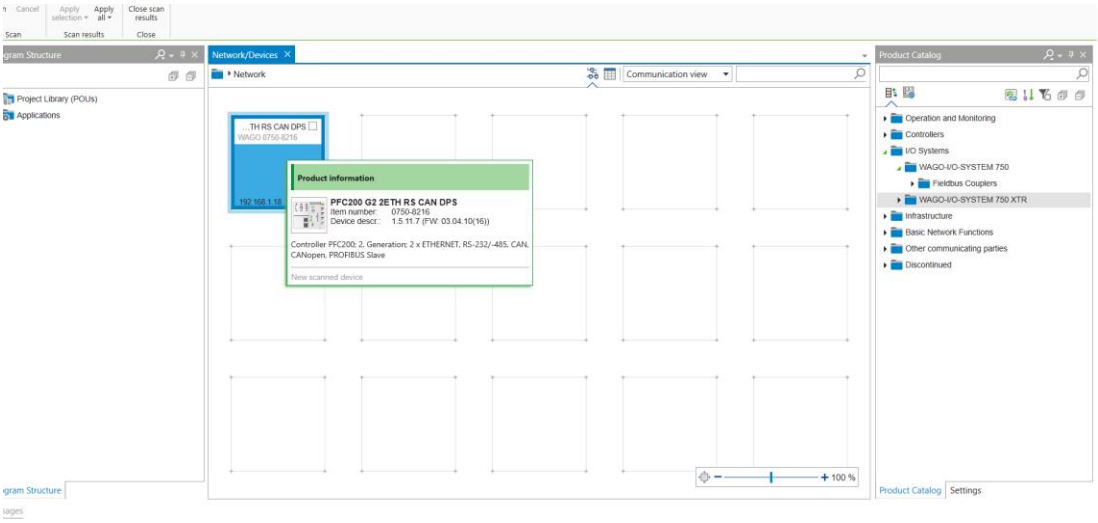


Figure 143 e!COCKPIT software scan

Project settings:

The setting of the file needs to be changed using the details shown in the apply all menu.

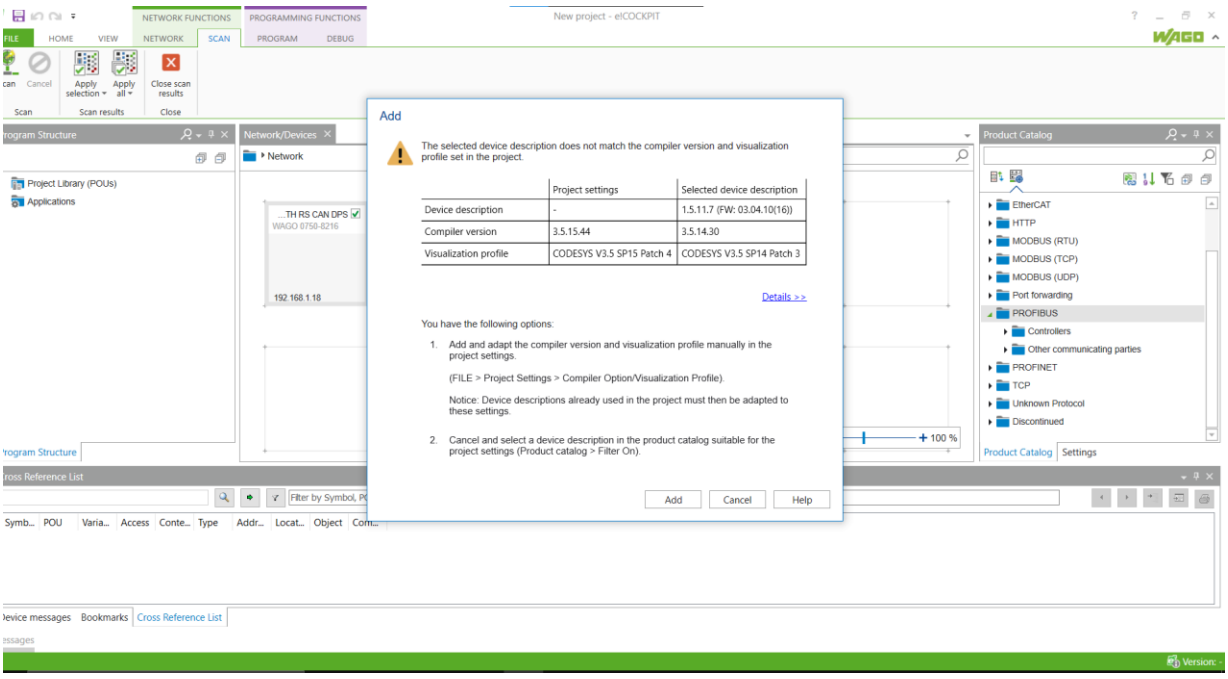


Figure 144 Project settings

All the modules need to be selected and added.

Once the modules are added, a simple LED program can be done to check the working of it.

Go to PLC task and add LED variable.

Go to PLC PRG and type LED=LED+1 and run the program on the PLC.

Once the process data length has been set up as per the requirement, the connection can be set up for the exchange of data.

IFM 3D Camera

The line following function was initially decided to be tested and carried out with the IFM 3D Camera. This is a part of the system for the purpose of redundancy. When there is a failure in the system because of any error that occurs due to the sensor values not being received by the Odroid, this component helps the system to continue working without any problem.

This is a part of the system for the purpose of redundancy. When there is a failure in the system because of any error that occurs due to the sensor values not being received by the Odroid, this component helps the system to continue working without any problem.

A 3D camera is an imaging device that enables the perception of depth in images to replicate three dimensions as experienced through human binocular vision. Some 3D cameras use two or more lenses to record multiple points of view, while others use a single lens that shifts its position. The camera version used here is the IFM 3D camera O3X100. The technical details of the camera can be found in the link below.

https://www.ifm.com/de/en/product/O3X100?tab=details

Connection:

The 3D camera is powered by the mains and is connected to the Odroid using the ethernet port and cable.

Figure 145 IFM 3D Camera

The following tasks were done to make use of the IFM Camera for the process of line following.

SOFTWARE:

OpenCV:

OpenCV which is Open-Source Computer Vision Library which is a library of programming functions and is mainly aimed at real time computer vision tasks. It is a tool used for image processing.

The OpenCV version that is supported with ROS melodic was installed on ODROID. To find more information about the Installation process of the software, refer the commands in the following website https://gist.github.com/tchamberlin/fe55da8afa266c544f1cd0f5f94f2f52

NOTE: These commands should be run on the terminal on ODROID.

OpenCV Error:

There was an error running OpenCV codes on VScodium when the file location was in any other folder other than Odroid home.

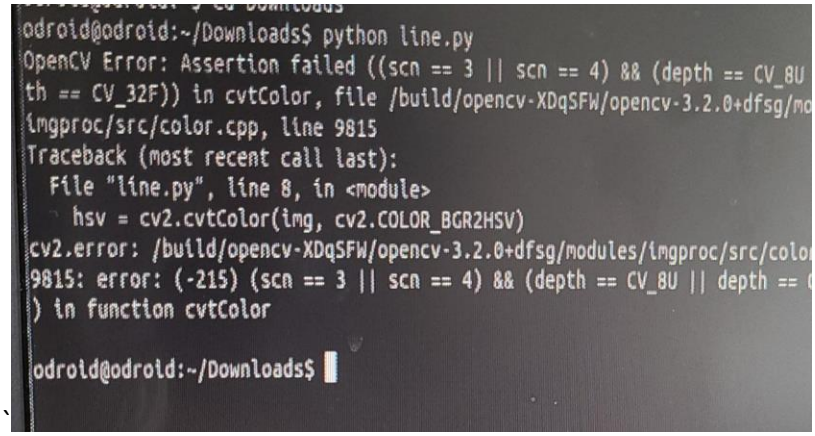


Figure 146 OpenCV Error

To resolve this, the file was created in a folder under Odroid home folder.

The next step is to get the different streams from the camera on RViz. There will be different streams, but the amplitude stream was used for image processing.

Figure 147 Amplitude Stream with camera perpendicular to the ground

At first, the IFM Camera was placed perpendicular to the ground at a distance of 1ft from it. The ground already had a white line drawn on top of it but due to the line being barely visible, the 3D camera was tilted by an approximate value of 60° with respect to the ground.

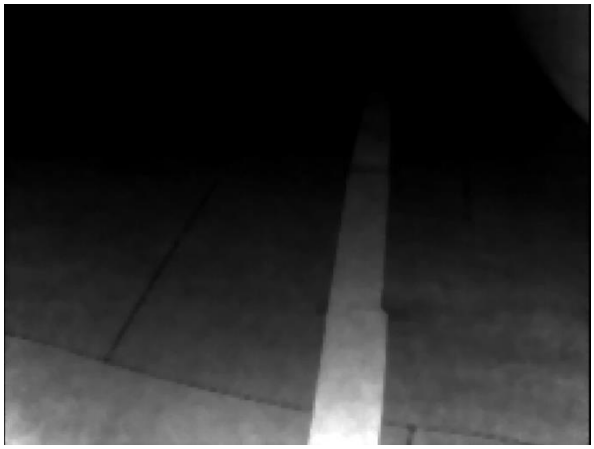


Figure 148 Amplitude Stream with camera at 60° to the ground

Image processing with HSV Filtering:

Image processing of the amplitude stream was carried out using the HSV filtering and masking to detect the line. The HSV (hue, saturation, value) filter was used to do the image processing here.

The reason we use HSV filtering for over RGB/BGR filtering is that HSV is more robust towards external lighting changes. In cases of minor changes in external lighting, HSV values vary relatively lesser than RGB values

In OpenCV, Hue has values from 0 to 180, Saturation and Value from 0 to 255. Thus, OpenCV uses HSV ranges between (0-180, 0-255, 0-255). The upper and lower values of HSV were defined and the masking was done.

While doing the HSV filtering, the HSV values of the white line interfered and coincided with the floor and other objects which would also be the case in the actual workspace scenario, hence the white line was replaced by a black line for testing.

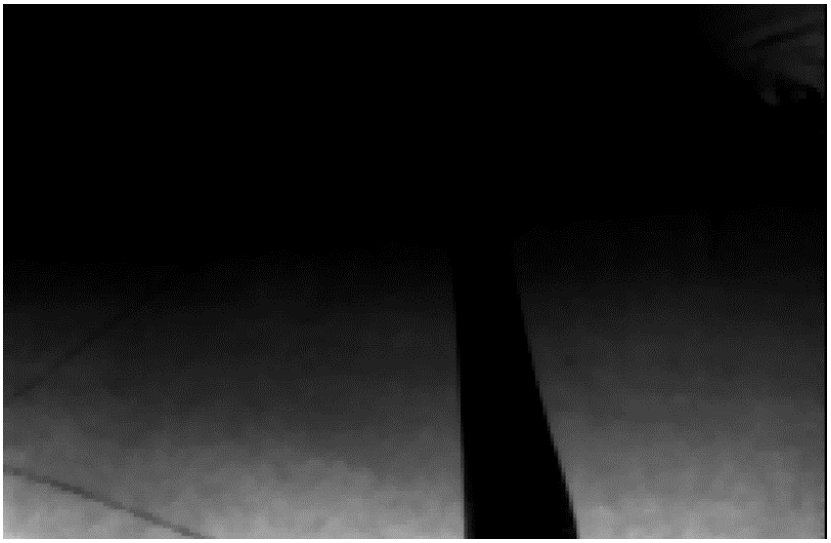


Figure 149 Amplitude stream of a black line on 3D camera

```
import cv2
import numpy as np
img = cv2.imread('black.jpg') #path of the image
cv2.imshow('Original', img)
cv2.waitKey(0)
while(1):
    # Take each frame

    # Convert BGR to HSV
    hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    # define range of black line in HSV
    lower = np.array([0,0,0])
    upper = np.array([180,255,0])
    # Threshold the HSV image to get only black colors
    mask = cv2.inRange(hsv, lower, upper)
    # Bitwise-AND mask and original image
    res = cv2.bitwise_and(img,img, mask= mask)
    cv2.imshow('img',img)
    cv2.imshow('mask',mask)
    cv2.imshow('res',res)
    k = cv2.waitKey(5) & 0xFF
    if k == 27:
        break
cv2.destroyAllWindows()
```

Figure 150 Code of HSV Filtering and masking

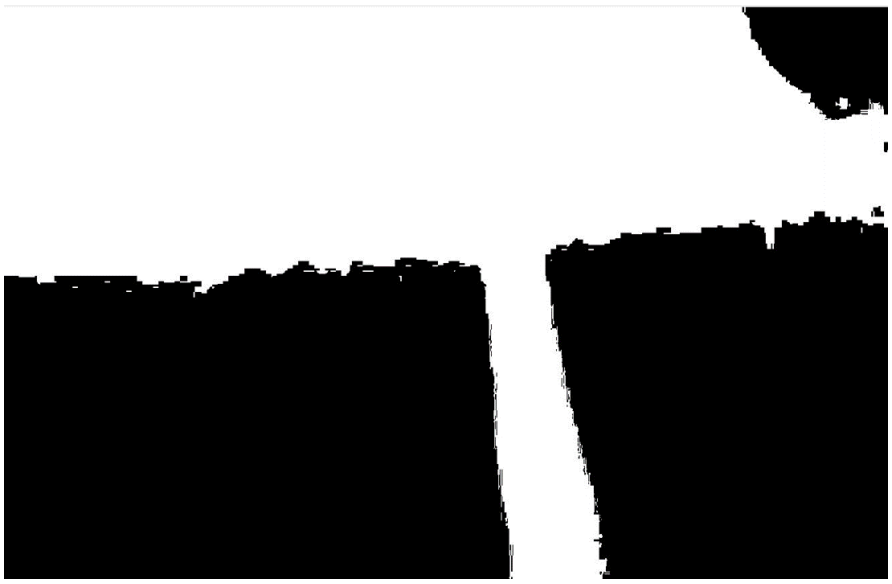


Figure 151 Image after HSV filtering and masking

Issues:

• The further objects were not in the image and appeared to be black and hence the HSV masking was also done for that part of the image.

• One way around that was to process only the bottom half of the image which or mount the camera parallel to the ground to get the complete image for better filtering. But then the camera would not be used for further processes such as obstacle avoidance or for localization and mapping.

• The IFM 3D camera gives a point cloud image and not the regular optical or RGB image to work with for image processing.

Conclusion

• Using optical camera with IR sensors for line detection for redundancy.

• Use the optical camera also for the station detection which would be done with the help of AR Tags.

• Using IFM camera only for obstacle detection.

2.7.2 Obstacle Detection

Ultrasonic Sensor:

The Microsonic pico+35/F M18 cylindrical ultrasonic sensor from microsonic with a transducer frequency of 400 kHz and a PNP/NPN switching output that is programmable as NCC or NOC has an operating range of 65-350 mm. NCC/NCO and rising/falling analogue characteristics can be set via pin 5 by means of the microsonic teach-in process. Its high class of protection IP67 and the M12 connector make use of the pico+35/F ultrasonic sensor possible in many industrial areas such as for positioning of products, presence detection of objects, people detection.

Figure 152 Ultrasonic sensor Microsonic pico+35/F

Advantages of Microsonic pico+ sensor

- Variant with 90° angled head
- IO-Link interface which supports of the new industry standard
- Automatic synchronisation and multiplex operation for simultaneous operation of up to ten sensors in close quarters
- UL Listed to Canadian and US safety standards
- Improved temperature compensation which leads to adjustment to working conditions within 120 seconds
- Smart Sensor Profiles which will have more transparency between IO-Link Devices
- Operates on 24Volts
- Well insulated and sturdy built
- Temperature compensation
- Accuracy of ± 1 %

Limitations of HC-SR04 Ultrasonic Sensor

- The object has its reflective surface at a shallow angle so that sound will not be reflected towards the sensor.

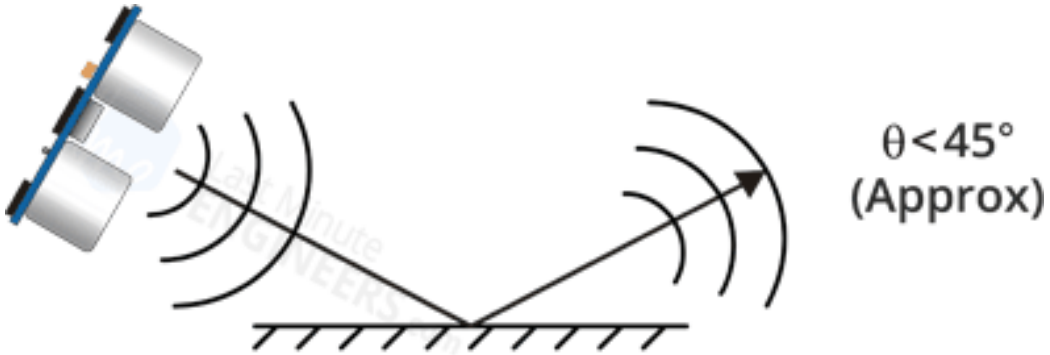


Figure 153 HC-SR04 Limitation - cannot detect object at a shallow angle

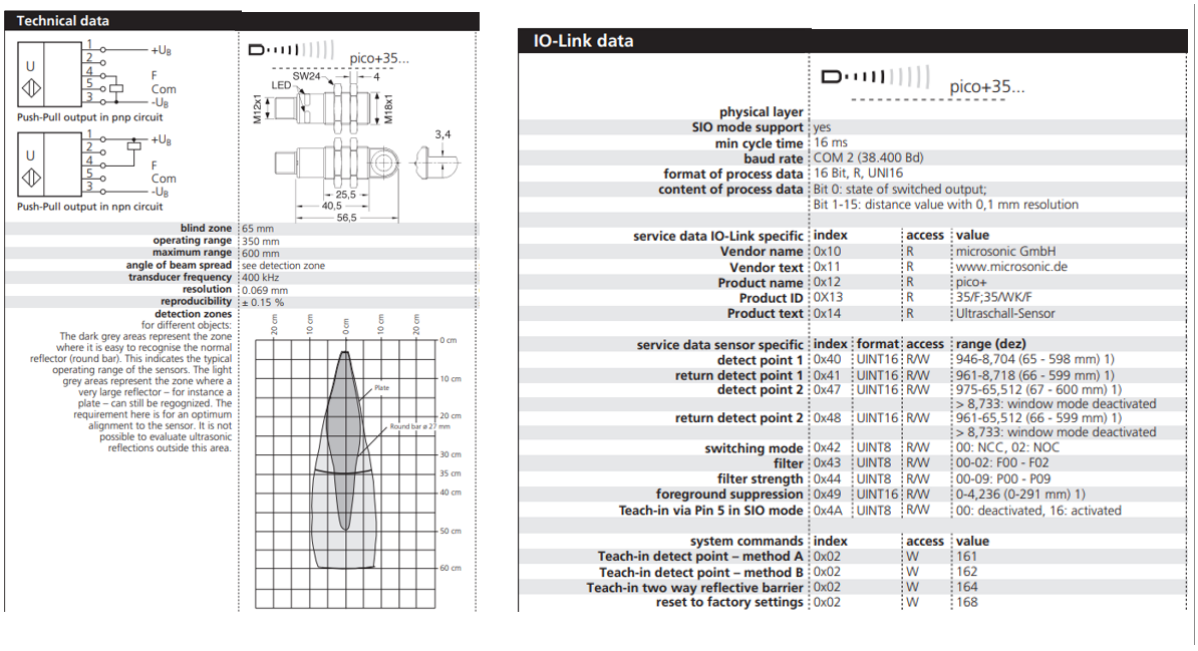
- The object is too small to reflect enough sound back to the sensor. In addition, if your HC-SR04 sensor is mounted low on your device, you may detect sound reflecting off of the floor.

Figure 154 HC -SR04 Limitation - cannot detect small objects

- While experimenting with the sensor, we discovered that some objects with soft, irregular surfaces rather than reflect sound and therefore can be difficult for the HC-SR04 sensor to detect.
- HC-SR04 Limitation – It cannot detect soft irregular surface or object
- Not suitable for outdoor purposes or in an unusually hot or cold environment.
- Quite delicate and might be damage easily.
- Loosely connected wires to the Arduino might cause the sensor to not work.
- Can operate only on 5 Volts.
- There is a higher resolution and accuracy of 0.3cm.

Microsonic Pico+ can overcome these disadvantages of HC-SR04!

Data sheet:



Technical data

Push-Pull output in pnp circuit

Push-Pull output in npn circuit

pico+35...

blind zone	65 mm
operating range	350 mm
maximum range	600 mm
angle of beam spread	see detection zone
transducer frequency	400 kHz
resolution	0.069 mm
reproducibility	± 0.15 %
detection zones	for different objects: The dark grey areas represent the zone where it is easy to recognise the normal reflector (round bar). This indicates the typical operating range of the sensors. The light grey areas represent the zone where a very large reflector – for instance a plate – can still be regognized. The requirement here is for an optimum alignment to the sensor. It is not possible to evaluate ultrasonic reflections outside this area.

IO-Link data

pico+35...

physical layer			
SIO mode support	yes		
min cycle time	16 ms		
baud rate	COM 2 (38.400 Bd)		
format of process data	16 Bit, R, UNI16		
content of process data	Bit 0: state of switched output; Bit 1-15: distance value with 0,1 mm resolution		
service data IO-Link specific	**index**	**access**	**value**
Vendor name	0x10	R	microsonic GmbH
Vendor text	0x11	R	www.microsonic.de
Product name	0x12	R	pico+
Product ID	0X13	R	35/F;35/WK/F
Product text	0x14	R	Ultraschall-Sensor
service data sensor specific	**index**	**format**	**access**	**range (dez)**
detect point 1	0x40	UINT16	R/W	946-8,704 (65 - 598 mm) 1)
return detect point 1	0x41	UINT16	R/W	961-8,718 (66 - 599 mm) 1)
detect point 2	0x47	UINT16	R/W	975-65,512 (67 - 600 mm) 1) > 8,733: window mode deactivated
return detect point 2	0x48	UINT16	R/W	961-65,512 (66 - 599 mm) 1) > 8,733: window mode deactivated
switching mode	0x42	UINT8	R/W	00: NCC, 02: NOC
filter	0x43	UINT8	R/W	00-02: F00 - F02
filter strength	0x44	UINT8	R/W	00-09: P00 - P09
foreground suppression	0x49	UINT16	R/W	0-4,236 (0-291 mm) 1)
Teach-in via Pin 5 in SIO mode	0x4A	UINT8	R/W	00: deactivated, 16: activated
system commands	**index**		**access**	**value**
Teach-in detect point – method A	0x02		W	161
Teach-in detect point – method B	0x02		W	162
Teach-in two way reflective barrier	0x02		W	164
reset to factory settings	0x02		W	168

Figure 155 Microsonic Pico+ sensor data sheet

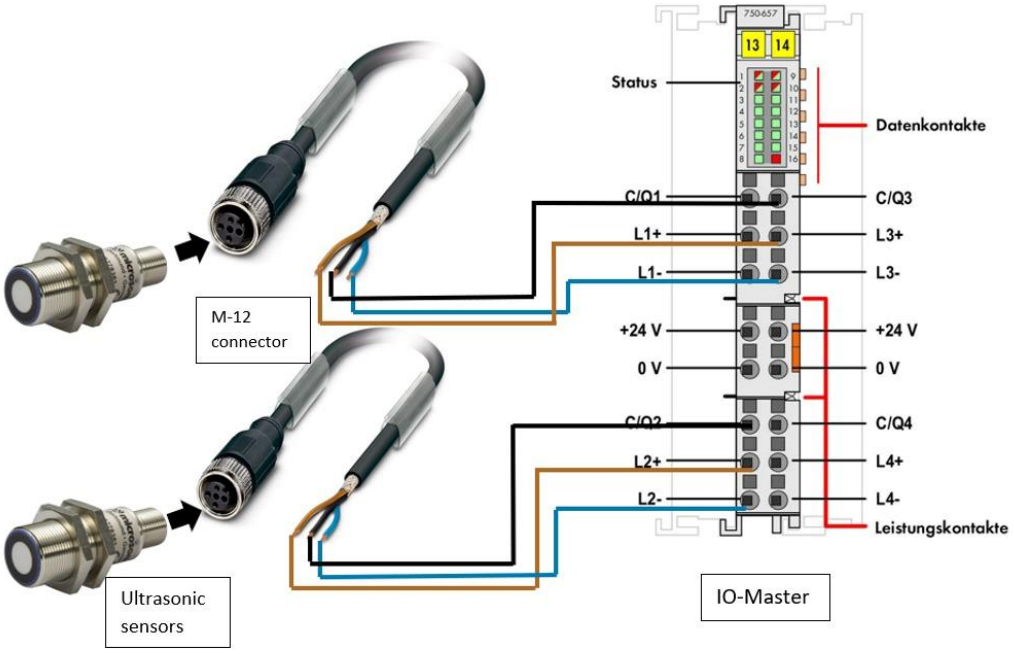


Figure 156 Connecting Ultrasonic sensor to the PLC Master via the M12 connector

Connecting to the Wago PLC:

1. The Wago Ethernet Settings needs to be installed in the PC.

2. The PLC is powered and the ethernet cable is connected either directly or through an ethernet to the type c/ USB port

3. The IP address of the system through Control panel is changed under Network & Internet. Click on View network status and tasks and under Unidentified network, you will find connections to Ethernet 4. Click on Properties to find Internet Protocol Version 4 (TCP/IPv4). When you double click on it, the IP address is to be changed to anything other than 192.168.1.1 , for example 192.168.1.2 and the subnet mask will automatically be changed but if not, it is changed to 255.255.255.0.

4. The changed IP address of the PC needs to be checked again on command prompt. Type ipconfig on command prompt and under Ethernet adapter Ethernet 4, you should see IPv4 Address : 192.168.1.2.

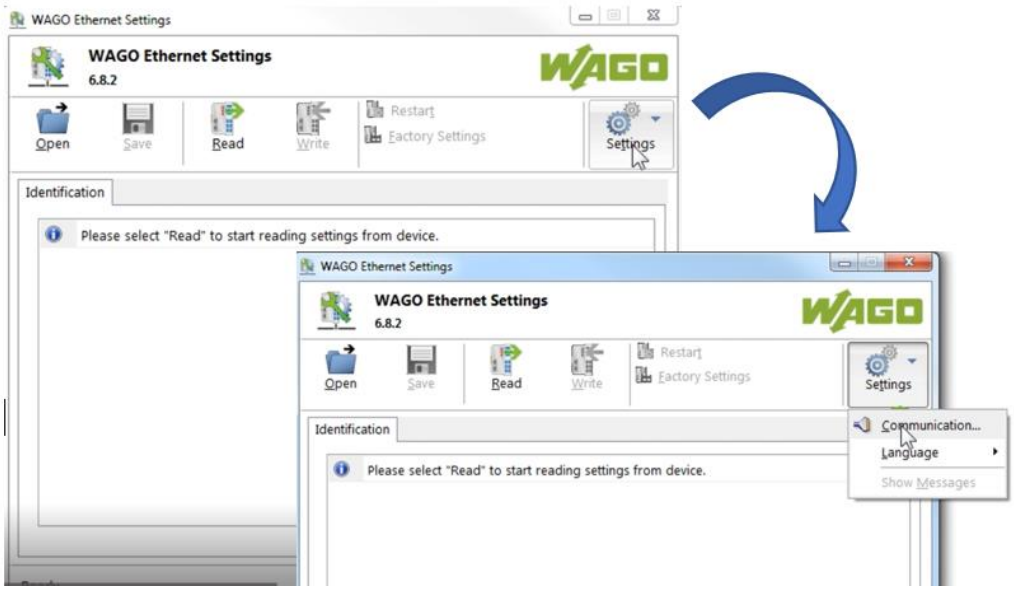


Figure 157 Finding the Ip address of the PLC Initial settings

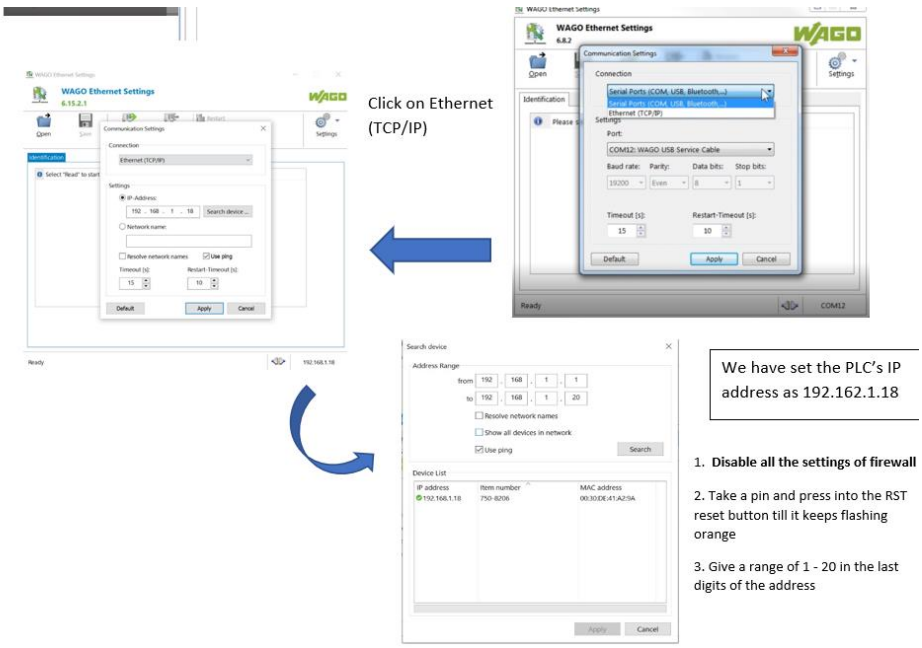


Figure 158 Finding the Ip address of the PLC Final settings

To reset the IP address from the 192.168.1.18 (not needed):

This procedure temporarily sets the IO address for the X1 interface to the fixed address "192.168.1.17".

When the switch is enabled, the fixed address is also used for interface X2. When the switch is disabled, the original address setting for interface X2 is not changed.

No reset is performed.

To make this setting, proceed as follows:

1. Set the mode selector switch to STOP and

2. Press and hold the Reset button (RST) for longer than 8 seconds.

Execution of the setting is signaled by the “SYS” LED flashing orange.

To cancel this setting, proceed as follows:

• Perform a software reset or Switch off the controller and then switch it back on.

• Perform the previous steps in WAGO Ethernet software.

• Click on the IP address to change it

• Run the Web based management.

• To login and use the credentials are **Username** : admin & **Password** : wago

• After logging in click on Networking, under TCP/IP Configuration, Static IP Address will be found, which needs to be changed to 192.168.1.18.

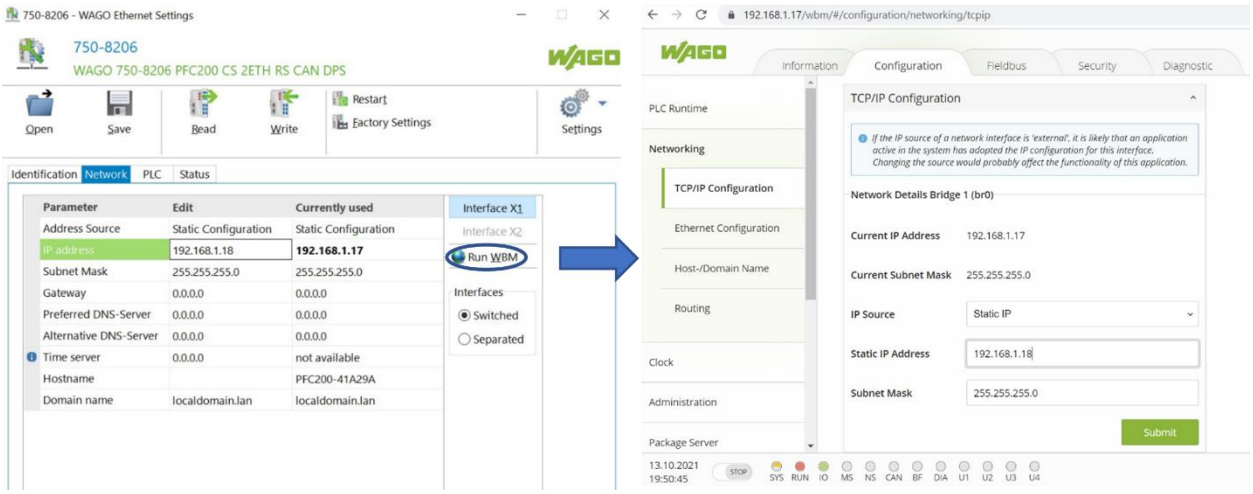


Figure 159 Resetting the IP address

PLC and E!COCKPIT

1. Go to settings and give a range of 192.168.1.1 to 192.168.1.20.

2. Scan the PLC and find the PLC of 192.168.1.18.

3. Select it and add all of its modules.

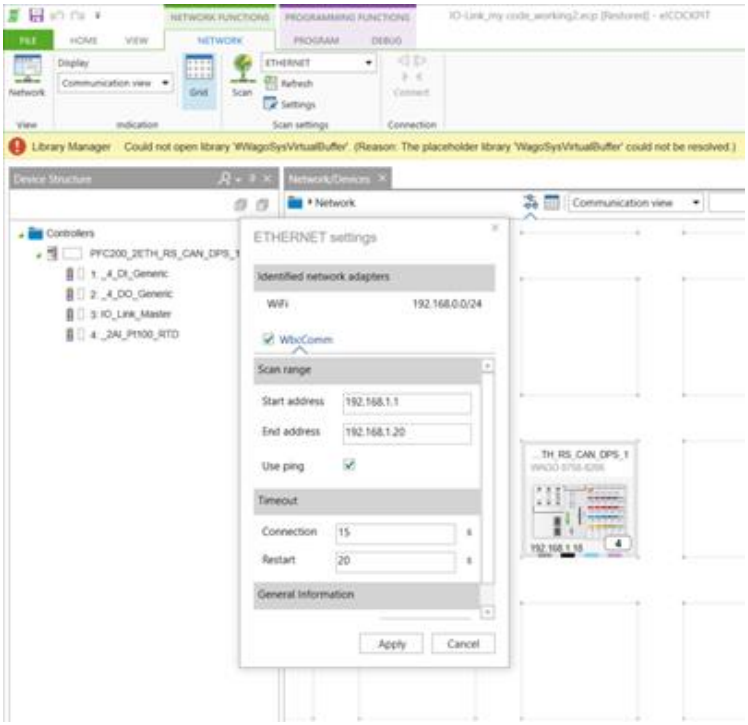


Figure 160 Finding the PLC in E-cockpit!

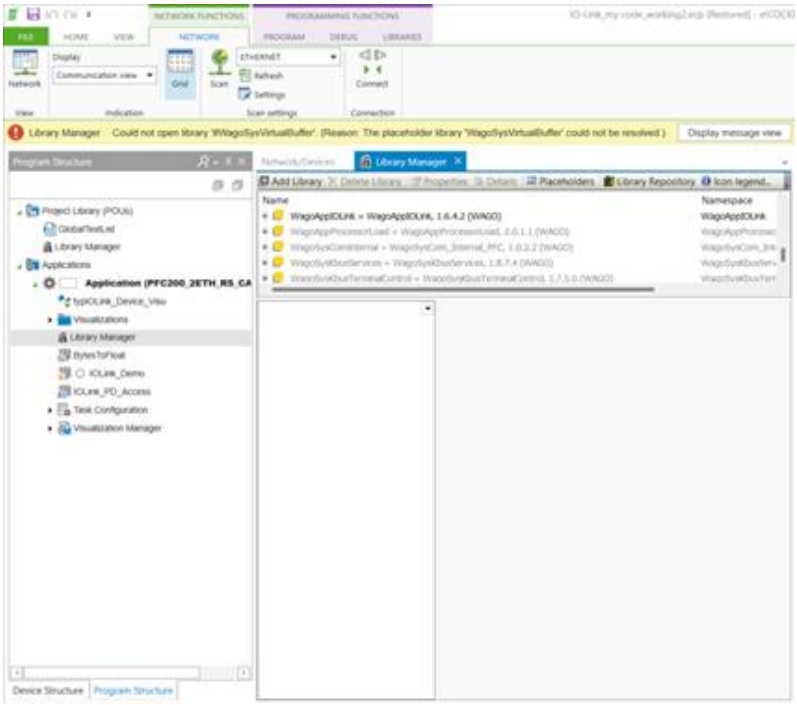


Figure 161 Adding the libraries in E-cockpit!

4. For the purpose of receiving the data on the Wago PLC side, the library called 'WagoAppIOLink' and through that FBIOL_PORTDATA is used to process data :

```
//Access the IOLink Process Data
IOLProcessData( xEnable:=bEnable,
            bIO_LinkPort:=byIOLinkPort,
            I_Port:=IO_Link_Master, //Name of the IO-Link Master in the "Device Structure" Tab of Ecockpit
            pTxBuffer:=ADR(typIOLink_VisuData.abyPDOut), //Process data output
            uDiTxnbytes:=SIZEOF(typIOLink_Device_Visu.abyPDOut), //Length of the data to send through PD Out
            pRxBuffer:=ADR(typIOLink_VisuData.abyPDIn),  //Process data input
            udirxbufferSize:=SIZEOF(typIOLink_VisuData.abyPDIn), //Size of the buffer used for recieving PDIn
            xTxTrigger:=PDOutTrigger, //Trigger to send proocess output data
            xCommunicationReset:=xCommReset, //Reset communication
            xError=>typIOLink_VisuData.bStatusInvalid,
            xValid=>typIOLink_VisuData.bStatusValid,
            udiRxNBytes=>typIOLink_VisuData.iPDInSize //Total recieved bytes (Process data input)
            );
IOLProcessData.oStatus.ShowResult(sDescription=>typIOLink_VisuData.sDeviceInfo); //Get a String error from the FB
```

Figure 162 Contents of FBIOL_PortData

5. Access of IO-Link process data, e.g., the information from the Sharp IR distance sensor or Ultrasonic sensor is done by the function block "FbIOL_PortData". Each port needs an own instance of this function block.

6. Each IO-Link Sensor connected to one of the ports needs this function block: FBIOL_PortData (FB).

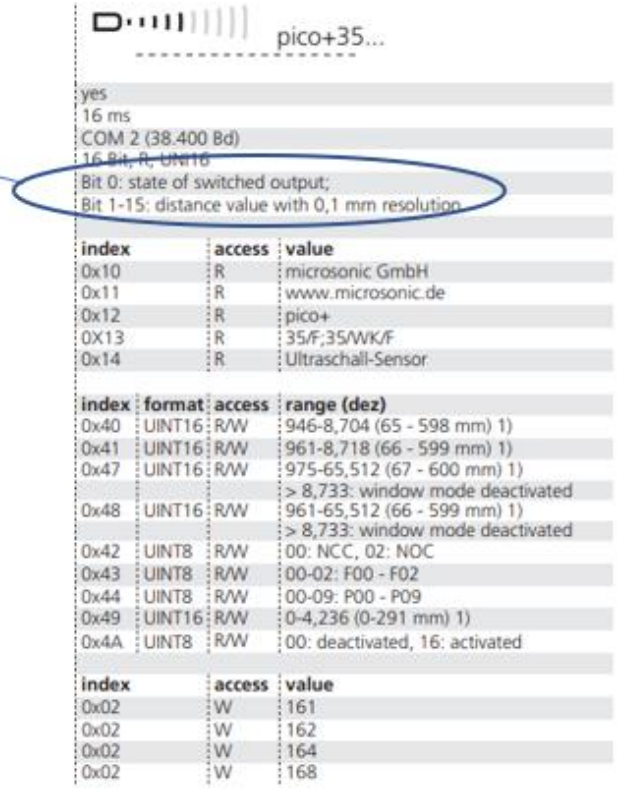
Interface variables

Scope	Name	Type	Comment
Input	xEnable	BOOL	Enable function block
	I_Port	WagoTypesModule_75x_657.I_Module_75x_657	Access to the module
	bIO_LinkPort	BYTE	Port 1..4
	pTxBuffer	POINTER TO BYTE	Pointer to the area, which should be transmitted, use ADR operator (e.g. ADR(TxData)
	udiTxNBytes	UDINT	Data count to be transmitted, max 32 Byte
	pRxBuffer	POINTER TO BYTE	Pointer to the area, where the received data should be stored, use ADR operator (e.g. ADR(RxData)
	udiRxBufferSize	UDINT	Size of receive Buffer, use SizeOf operator, e.g. SizeOf(RxData)
Inout	xTxTrigger	BOOL	Trigger the transmission of data to the sensor, variable will be reset by function block
	xCommunicationReset	BOOL	reset MBX2 communication in case of fragmented mode
Output	xValid	BOOL	Data from sensor is valid
	xBusy	BOOL	Configuration of the port in progress
	xError	BOOL	Error occured
	udiRxNBytes	UDINT	Number of received bytes
	bRxRefreshCounter	BYTE	Counter for received messages if used with a fragmented port
	oStatus	WagoSysErrorBase.FbResult	ReadingSettings ->first step during configuration of the channel MBX_NotReady-> second step OK ->process values will be delivered InProgress ->wait to reach step MBX_NotReady

Table 12 Interface variables of FBIOL_PortData function block

7.If the data sheet is observed:

- 2 Bytes will be processed.
- Bit 0 should be discarded, and Bit 1-15 should be processed.



pico+35...

yes	
16 ms	
COM 2 (38.400 Bd)	
16 Bit; R; UINT16	
Bit 0: state of switched output;	
Bit 1-15: distance value with 0,1 mm resolution	

index	access	value
0x10	R	microsonic GmbH
0x11	R	www.microsonic.de
0x12	R	pico+
0X13	R	35/F;35/WK/F
0x14	R	Ultraschall-Sensor

index	format	access	range (dez)
0x40	UINT16	R/W	946-8,704 (65 - 598 mm) 1)
0x41	UINT16	R/W	961-8,718 (66 - 599 mm) 1)
0x47	UINT16	R/W	975-65,512 (67 - 600 mm) 1)
			> 8,733: window mode deactivated
0x48	UINT16	R/W	961-65,512 (66 - 599 mm) 1)
			> 8,733: window mode deactivated
0x42	UINT8	R/W	00: NCC, 02: NOC
0x43	UINT8	R/W	00-02: F00 - F02
0x44	UINT8	R/W	00-09: P00 - P09
0x49	UINT16	R/W	0-4,236 (0-291 mm) 1)
0x4A	UINT8	R/W	00: deactivated, 16: activated

index	access	value
0x02	W	161
0x02	W	162
0x02	W	164
0x02	W	168

Figure 163 Datasheet of pico+35

Bit shifting is required to be done in IOLINK_Demo and the final value is displayed in Distance.

```
oIOLinkSensor(I_Port:=IO_Link_Master,bEnable:=TRUE,sName:=sSensorName);
IF oIOLinkSensor.info.bStatusValid THEN
    DistanceWord := oIOLinkSensor.info.abyPDIn[1] +  SHL(TO_WORD(oIOLinkSensor.info.abyPDIn[0]), 8);
    DistanceWord := SHR(DistanceWord, 1);
    Distance :=  DistanceWord / 10 ; //Distance in mm
```

Figure 164 Bit-shifting done in the code

LIDAR

2D LiDAR sensors are suitable for performing detection and ranging tasks on surfaces. They provide long distance measurement with accuracy. For the low-cost project, we considered using RP LIDAR a1 by Slamtec. The lidar can provide a variable frequency of 5Hz and 10Hz. It can detect objects within the circumference with 12 meters radius. RP LIDAR transmits infrared laser and after reflection from the target object's surface, it Is received by acquisition system. the working principle is based on laser triangulation, based on the angle between sent light and received light, the distance of object is determined.

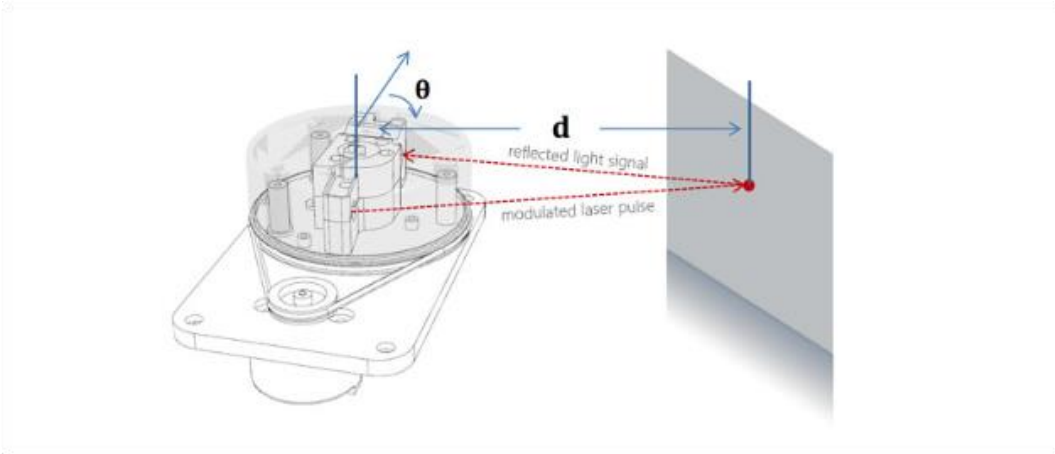


Figure 165 RP LIDAR Working

Figure 166 RP LIDAR A1

Lidar provides 360 (degree) coverage; hence it acts as best sensor for obstacle detection. The objects near the mower can be detected and the mower can respond accordingly either by stopping or by turning.

RP LIDAR is equipped with USB adapter which allows plug and play interface.

Figure 167 Connection from RP LIDAR to USB Adapter

Figure 168 Connection from USB Adapter to PC via micro-USB Cable

Once the device is connected, the drivers can be installed and libraries from: https://github.com/slamtec/rplidar_ros.

The instructions to use RP LIDAR can be found on gitlab (project repository).

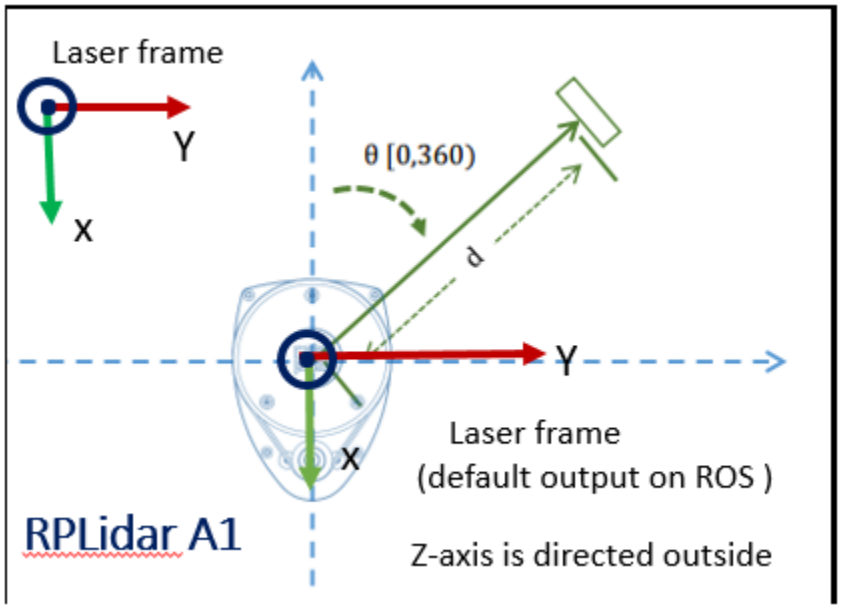


Figure 169 frame reference of rplidar

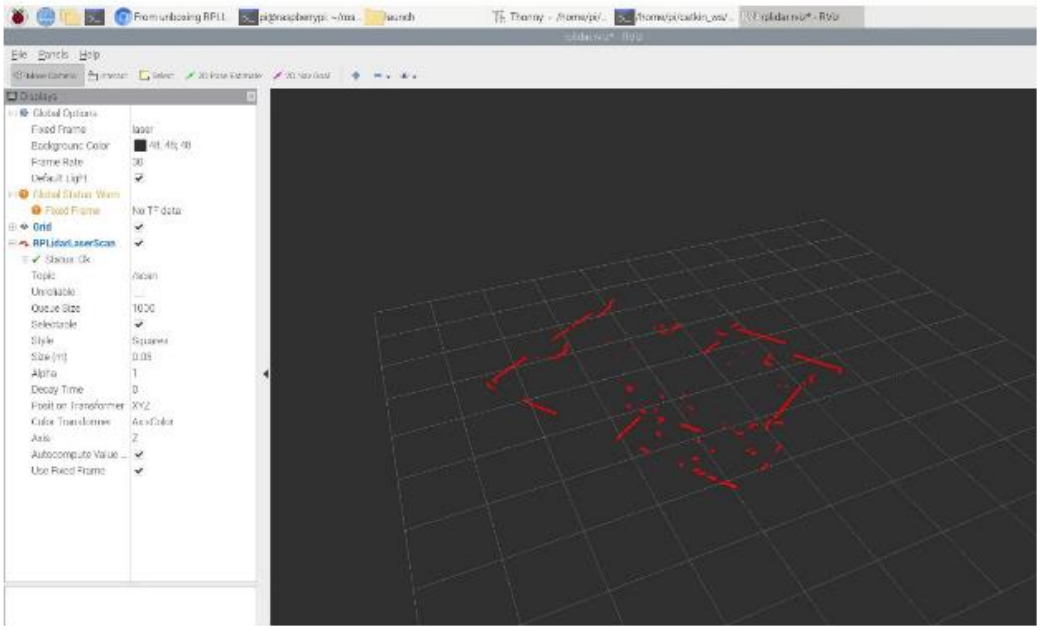


Figure 170 lidar points in rviz

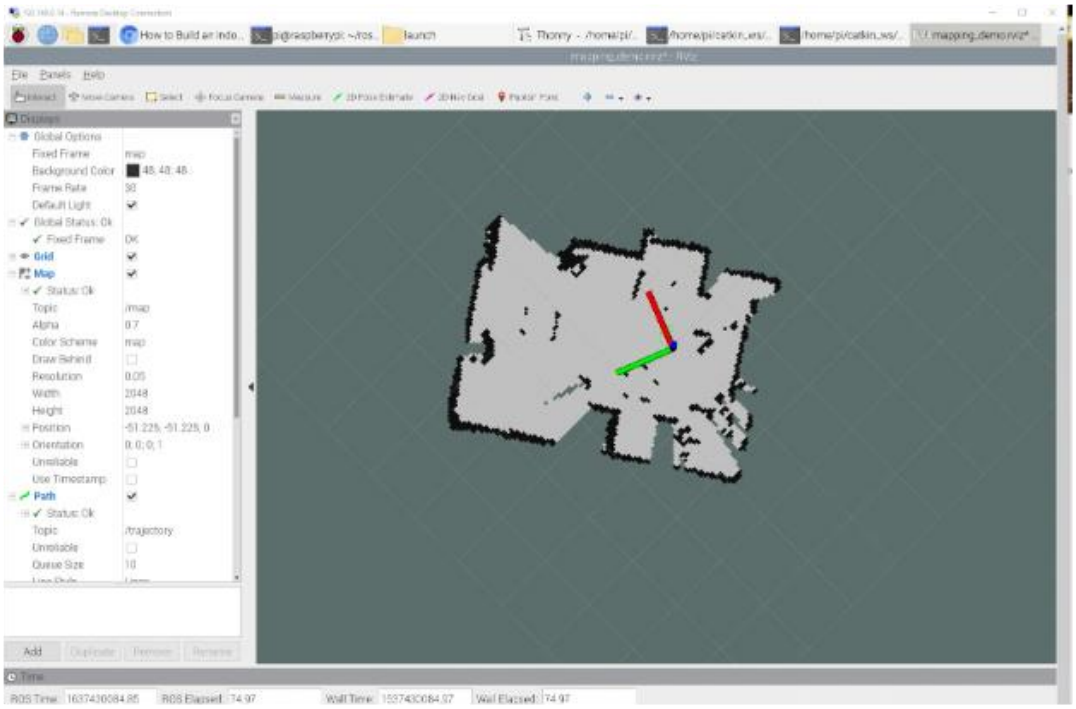


Figure 171 future implementation for navigation with map generation

2.7.3 Station Detection

Station detection basically means a function of being able to detect a station as per the programmer's choice to do the necessary action. The AGV bot must detect different stations to

perform specific tasks at each station with the help of AR Tags. For this purpose, station detection must be implemented with line following.

At first, the augmented reality tags (AR Tags) detection was tested with the help of USB camera and OpenCV.

Augmented Reality Tags (AR Tags)

AR Tags are represented by a black and white square image with specific patterns within the black border. There are different varieties of AR tags which are generated by different algorithms.

AR Tags can be used to facilitate the appearance of virtual objects, games, and animations within the real world. They have video tracking capabilities and can calculate a camera's position and orientation relative to physical markers in real time. In augmented reality applications, once the position of camera is known, a virtual camera can be positioned at the same point which reveals the virtual object at the location of the AR tag.

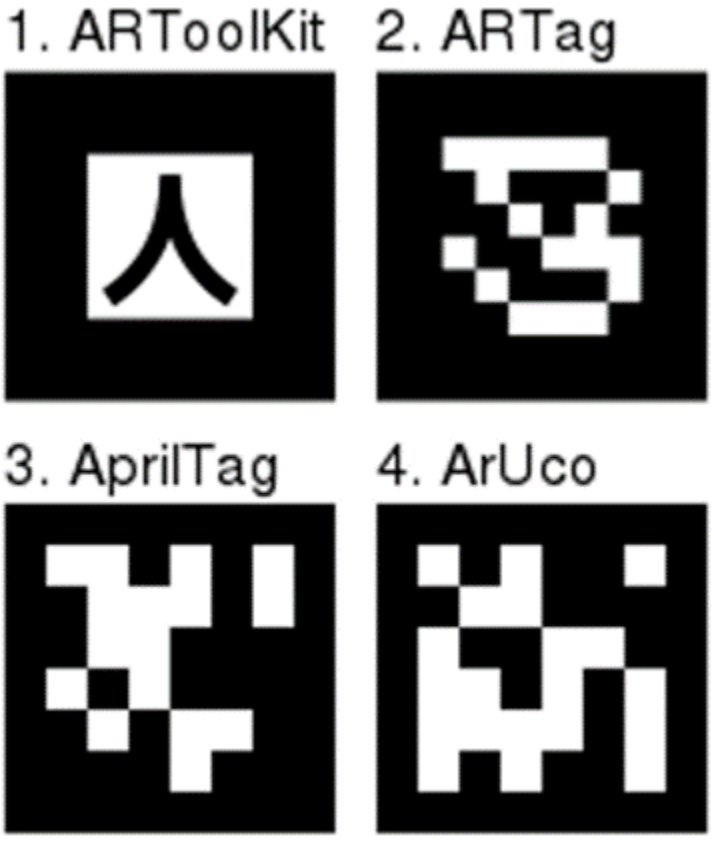


Figure 172 Examples of different kinds of AR Tags

Flow chart for station detection using AR Tags

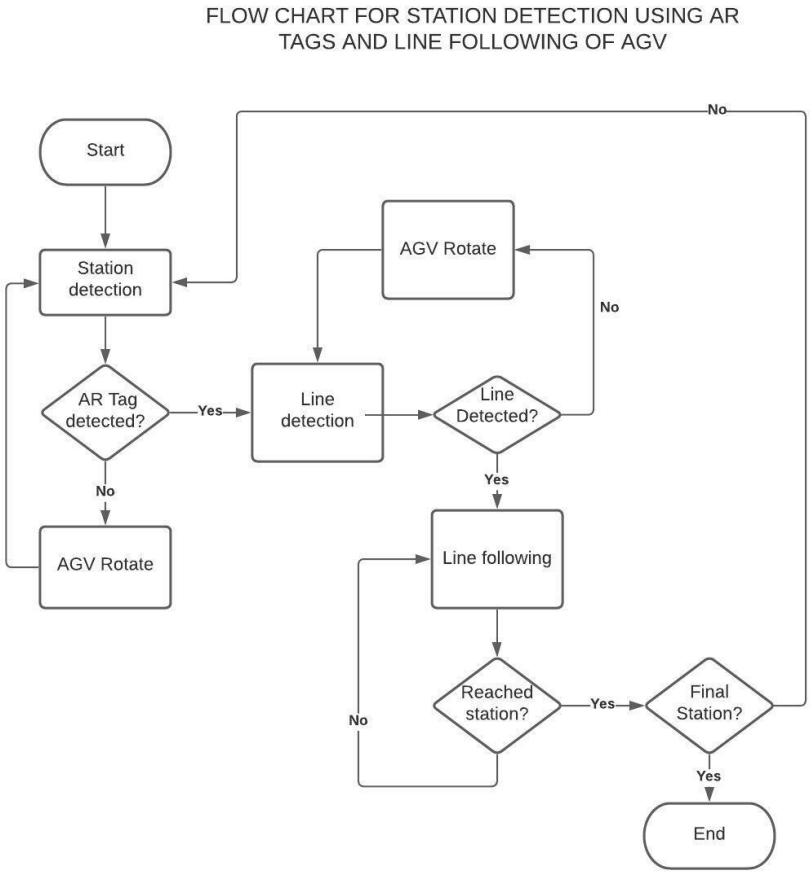


Figure 173 Station Detection Flow Chart

The above flow chart represents how the station detection using AR Tags with line following would function.

• The bot, once turned on, goes into the station detection algorithm and if the AR Tag is not detected in the field of vision of the camera, the bot rotates on its axis until the AR Tag is detected.

• After detection of the AR Tag, the bot goes to the line detection algorithm and aligns itself with the line by once again rotating on its own axis.

• Once the bot is aligned with the line, it follows the line to reach a station and once the station is reached and it's not the final station, the whole process is repeated and if it's the final station, the bot comes to a stop.

Testing of AR Tag detection

For the testing of AR Tag detection, ArUco markers were used. They are binary square fiducial markers that can be used for camera pose estimation. The main advantage of ArUco markers is that their detection is robust, fast, and simple. The ArUco module in OpenCV includes the

detection of these types of markers and the tools to employ them for pose estimation and camera calibration.

Camera calibration

It is essential to know the parameters of a camera to use it effectively as a visual sensor. The process of estimating the parameters of a camera is called camera calibration i.e., to get all the information about the camera required to determine an accurate relationship between a 3D point in the real world and its corresponding 2D pixel in the image captured by the calibrated camera. This was also done with the help of OpenCV. Camera calibration is usually performed using the OpenCV 'calibrateCamera()' function

There are different types of camera calibration methods. In our project, the so called 'Calibration pattern' method was used. First, several images of a pattern from different dimensions were taken. Chess board pattern was used here. The advantages of chess board patterns are

• They are distinct and easy to detect in an image.

• The corners of the squares on the checkerboard are ideal for localizing them because they have sharp gradients in two directions.

• These corners are also related by the fact that they are at the intersection of checkerboard lines and are used to robustly locate the corners of the squares in the chess board pattern.

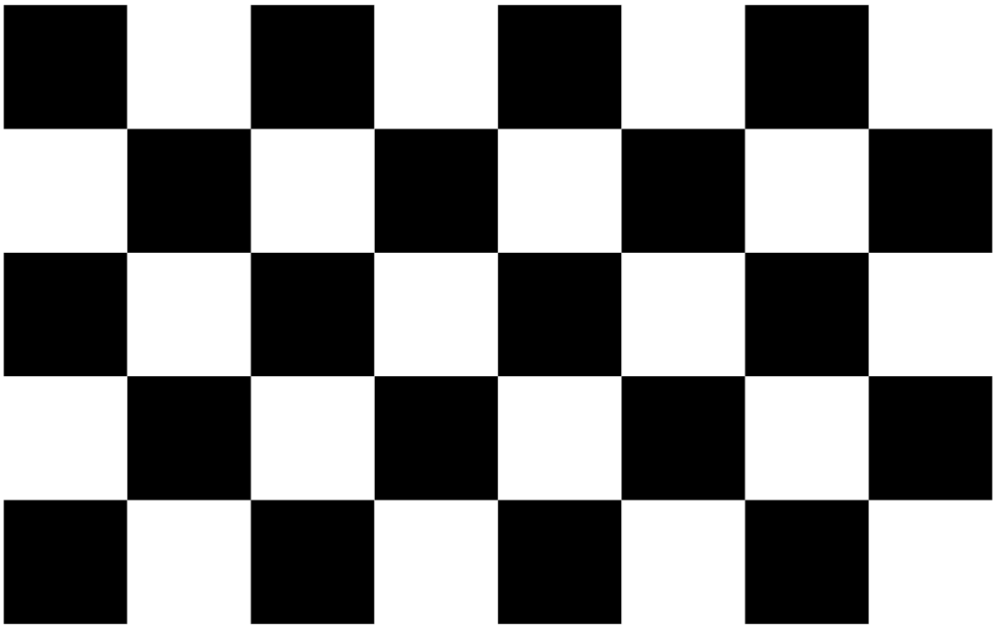


Figure 174 Chess board pattern used for camera calibration

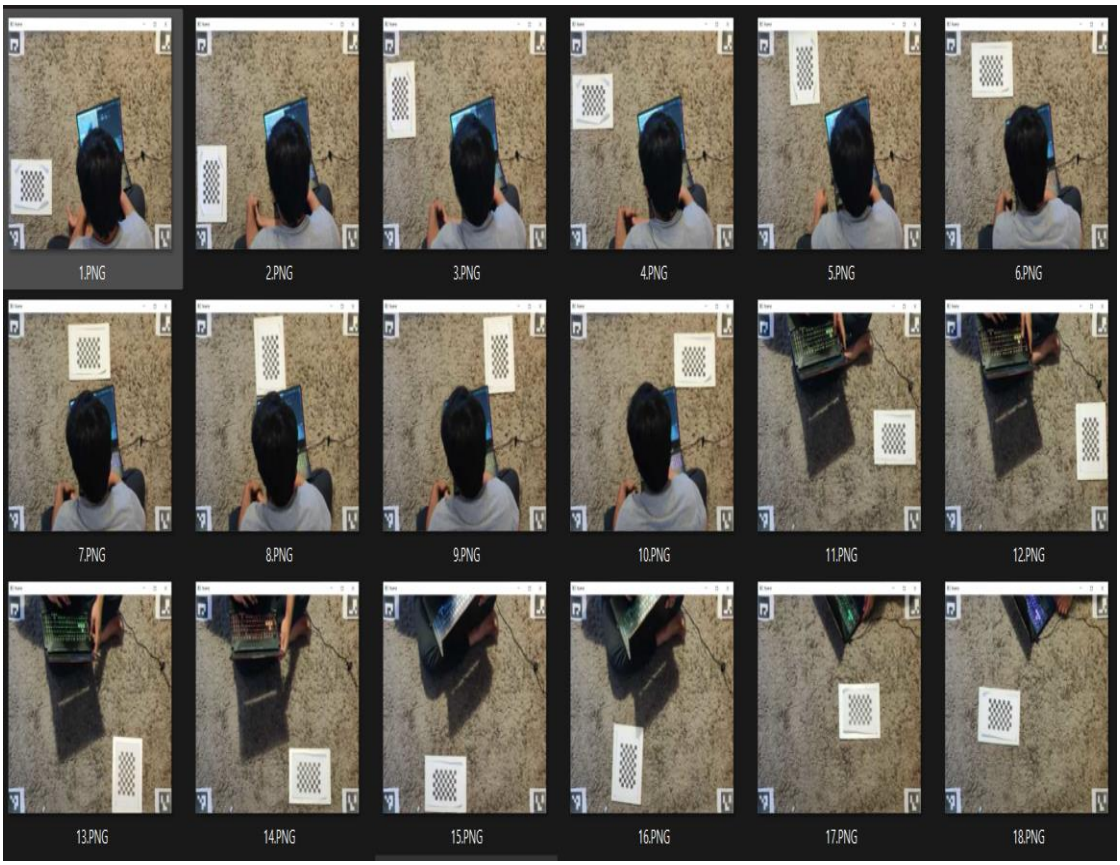


Figure 175 Multiple images of chess board at different location with different orientations

After getting the 3D location of points on the checkerboard in world coordinates, we need to get 2D pixel location of these chess board corners in the images. OpenCV provides a built in function called findChessboardCorners that looks for a checkerboard and returns the coordinates of the corners.

For better results, it is important to get the location of corners with sub-pixel level of accuracy. OpenCV's function cornerSubPix takes in the original image, and the location of corners, and looks for the best corner location inside a small neighborhood of the original location.

The final step of calibration is to pass the 3D points in world coordinates and their 2D locations in all images to OpenCV's calibrateCamera method.

```python
import numpy as np
import cv2
import glob

# Wait time to show calibration in 'ms'
WAIT_TIME = 100

# termination criteria for iterative algorithm
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)

# generalizable checkerboard dimensions
# https://stackoverflow.com/questions/31249037/calibrating-webcam-using-python-and-opencv-error?rq=1
cbrow = 6
cbcol = 9

# prepare object points, like (0,0,0), (1,0,0), (2,0,0) ....,(6,5,0)
# IMPORTANT : Object points must be changed to get real physical distance.
objp = np.zeros((cbrow * cbcol, 3), np.float32)
objp[:, :2] = np.mgrid[0:cbcol, 0:cbrow].T.reshape(-1, 2)

# Arrays to store object points and image points from all the images.
objpoints = [] # 3d point in real world space
imgpoints = [] # 2d points in image plane.

images = glob.glob(r"C:\Users\aadit\OneDrive\Desktop\chessboard_calib\*.png") #path of the images for calibration

for fname in images:
    img = cv2.imread(fname)
    gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

    # Find the chess board corners
    ret, corners = cv2.findChessboardCorners(gray, (9,6),None)

    # If found, add object points, image points (after refining them)
    if ret == True:
        objpoints.append(objp)

        corners2 = cv2.cornerSubPix(gray,corners,(11,11),(-1,-1),criteria)
        imgpoints.append(corners2)

        # Draw and display the corners
        img = cv2.drawChessboardCorners(img, (cbcol, cbrow), corners2,ret)
        cv2.imshow('img',img)
        cv2.waitKey(WAIT_TIME)

cv2.destroyAllWindows()
ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints, gray.shape[::-1],None,None)
```

```python

# ---------- Saving the calibration -----------------
cv_file = cv2.FileStorage(r"C:\Users\aadit\OneDrive\Desktop\chessboard_calib\hhhh.yaml", cv2.FILE_STORAGE_WRITE)
cv_file.write("ret", ret)
cv_file.write("camera_matrix", mtx)
cv_file.write("dist_coeff", dist)
print(rvecs)
print(tvecs)

# note you *release* you don't close() a FileStorage object
cv_file.release()
```

Figure 176 Code for camera calibration using chessboard

Once the camera calibration was done, ArUco markers that were to be detected were generated. The link for the generation of ArUco marker: https://chev.me/arucogen/.

ArUco marker detection

Given an image containing ArUco marker, the detection process returns the detected marker. The detected marker includes the position of the center point of the marker, the ID of the marker and the orientation of the marker.

The marker detection algorithm has two steps:

- Marker candidate detection – In this step, the image is analyzed, and square shapes are found that are the candidates for markers. First adaptive thresholding is done to segment the markers and then contours are extracted from the threshold image. The ones that are not convex or do not approximate to a square shape are discarded. Extra filtering is also applied to remove contours that are too small or too big and contours that are too close to each other.
- After the detection of candidates, it is important to determine if they are markers by analyzing the inner codification. This step starts by extracting the marker bits of each marker. To do so, a perspective transformation is first applied to obtain the marker in its canonical form. Then, the canonical image is brought to a threshold using Otsu to separate white and black bits. The image is divided into different cells according to the marker size and the border size. Then the number of black or white pixels in each cell is counted to determine if it is a white or a black bit. Finally, the bits are analyzed to determine if the marker belongs to the specific dictionary. Error correction techniques are employed when necessary.

The detection in ArUco module is done by the function ‘detectMarkers()’. This function is the most important one because all of the functionality based on detected markers are returned by ‘detectMarkers()’.

After ‘detectMarkers()’, it has to be checked whether the markers have been correctly detected. The ArUco module provides a function to draw detected markers in the input image. The function ‘drawDetectedMarkers()’ is used.

Once the code was run and executed, the marker ID was detected when the marker was placed in front of the camera. As the USB camera was currently unavailable, the laptop webcam was used instead. The center point of the marker was obtained by simple math after getting the 4 different corners of the square. The location of this center point in the 2D camera feed was got in terms of the X and Y location of the pixel and the orientation of the marker was also obtained.

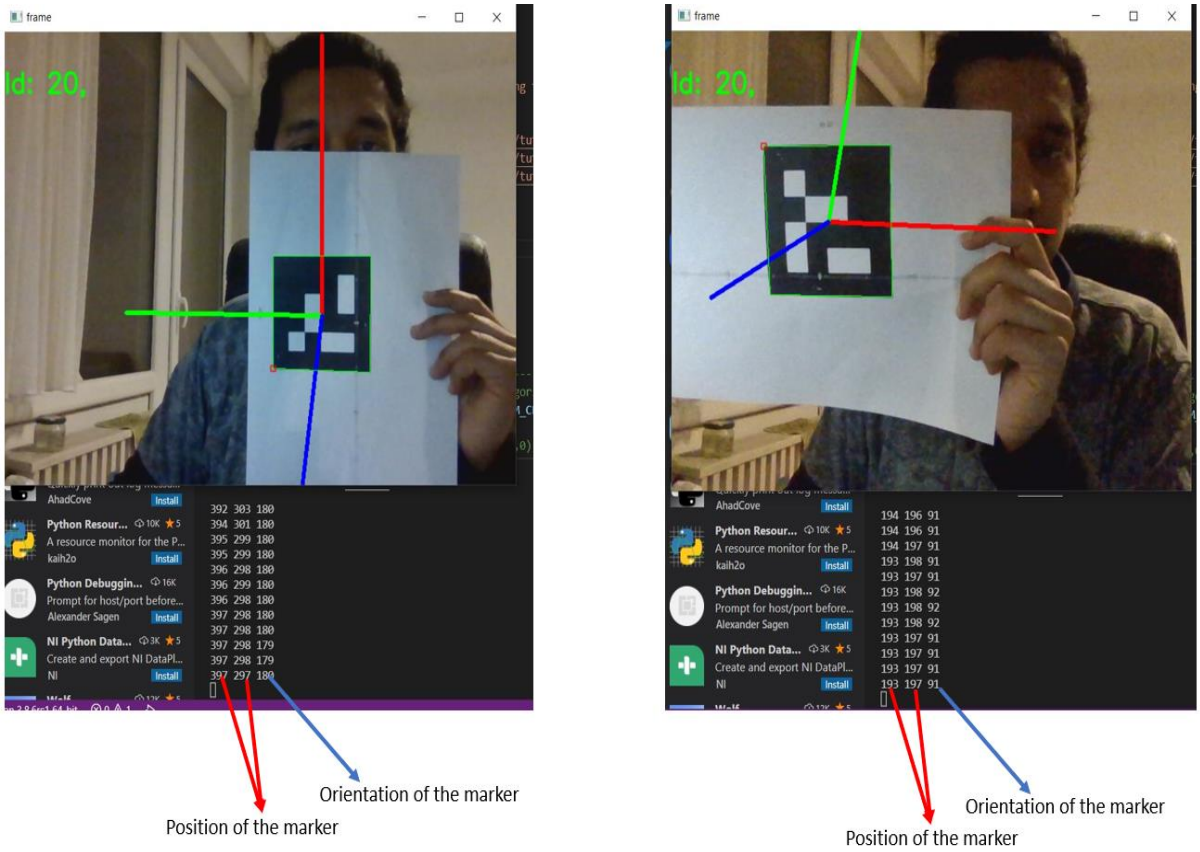


Figure 177 Output of the AR Tag detection code

RFID TAGS

For an alternate way to detect station and could also be used as a redundant system, testing of RFID Tags/Card detection was done which were detected by PN5180 -NFC Reader. At first, a NodeMCU ESP32 was used to test the PN5180 reader.

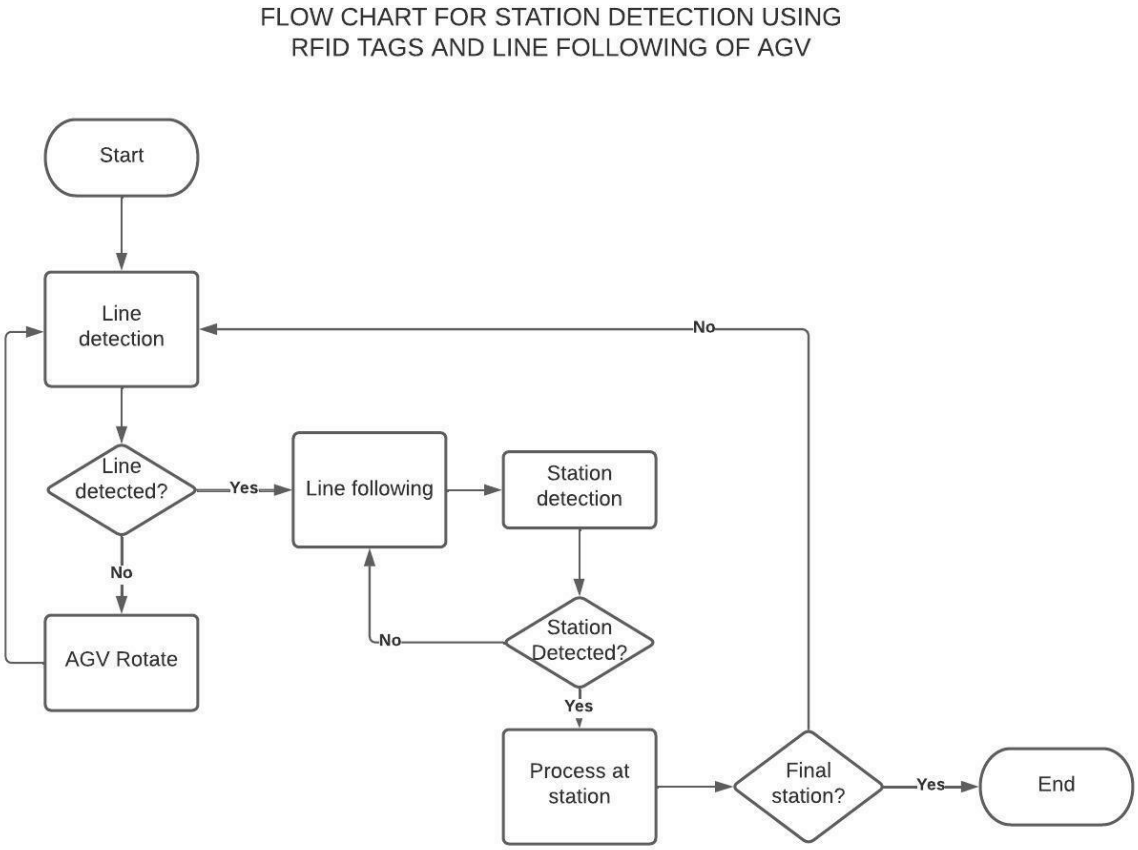


Figure 178 Flow chart for station detection using RFID Tags

The above chart represents how the station detection using RFID Tags with line following would function. It is like how AR Tag detection works but with minor changes in the algorithm.

- The bot once started goes detects the line and rotates on its own axis and aligns itself to the line to perform the line following algorithm.
- During line following, the station detection algorithm runs in parallel and until the station is detected, the bot moves along the line.
- Once the station is detected, the bot performs the specific task at that station and checks if it's the final station. If not, it goes back to the line detection and following algorithm and if it is the final station, the bot comes to a stop.
- If used with AR Tags for station detection, it becomes a redundant system. Only when both AR Tag is detected with the RFID Tag, it is concluded that the station is detected and the station detection becomes more accurate.

Radio frequency identification (RFID)

It refers to wireless system that consists of two components that are tags and readers. The reader is a device that has one or more antennas which emit radio waves, and they receive the signals back from the RFID tags. Tags use radio waves to communicate their identity and other

information to the nearby readers. There can be passive or active tags. Passive tags are powered by the reader and do not require a battery. Active ones are powered by batteries. For this project, a PN5180 NFC Reader was used with the RFID Card as a tag.

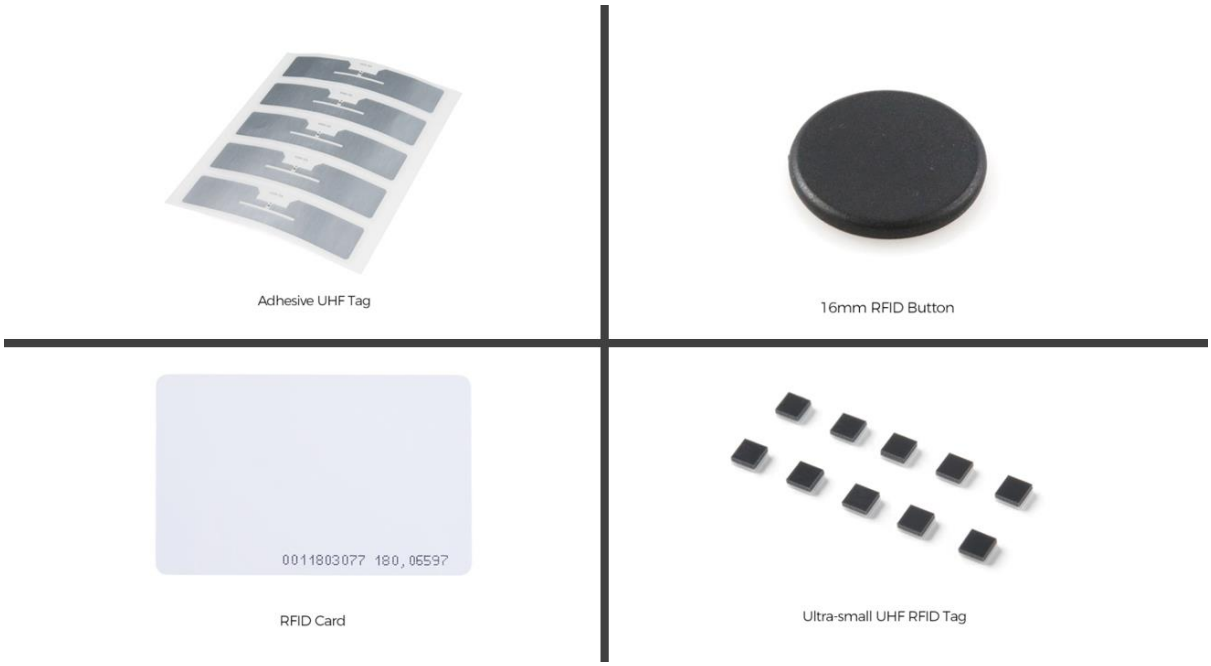


Figure 179 Examples of RFID Tags

PN5180 NFC Reader

The PN5180 is a high-performance full NFC Forum-compliant frontend IC for various contactless communication methods and protocols. The independence of a real-time host controller interaction makes the PN5180 a good choice for systems, which operate a pre-emptive multitasking OS like Linux or Android.

Figure 180 PN5180 NFC Reader

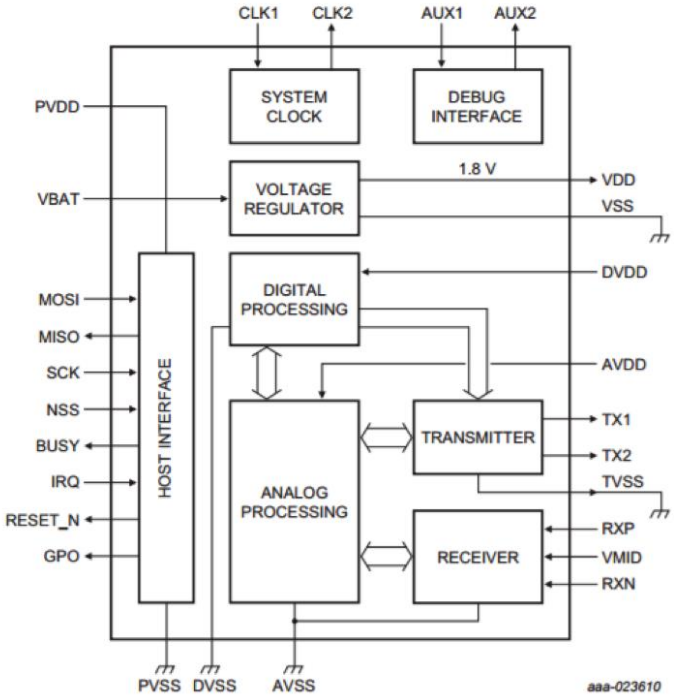


Figure 181 PN5180 block diagram

Testing of PN5180 Reader

The PN5180 NFC reader was connected to the GPIO pins of the NodeMCU and communicated with it using SPI communication protocol. There is repository on github which was referred for the testing of the reader with the RFID card following the instructions. The link for the repository is -https://github.com/playfultechnology/arduino-rfid-PN5180.

Once the connections were done as shown in the figure, the .ino code was modified according to the connections and executed on Arduino IDE. When the RFID card was brought near the reader, it used to detect it and gave its ID on the serial monitor of Arduino IDE. The link for the code is - https://github.com/playfultechnology/arduino-rfid-PN5180/blob/master/arduino-rfid-PN5180.ino.

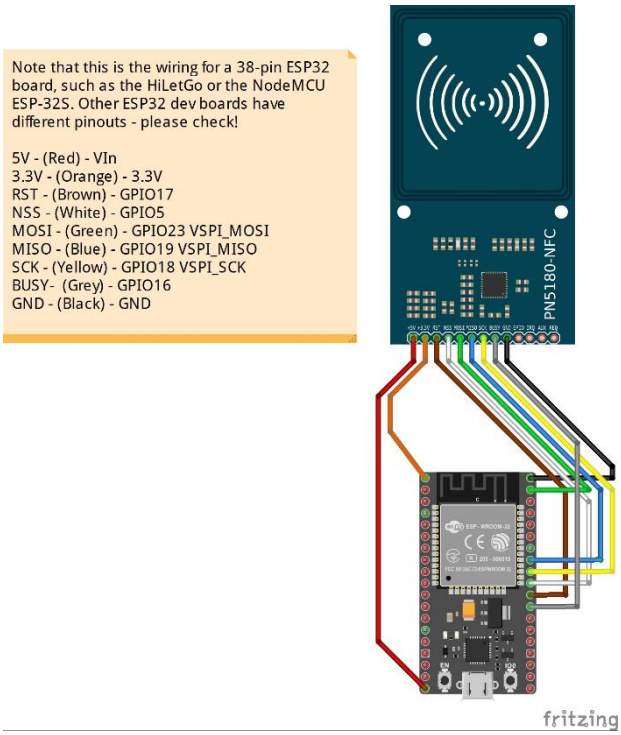


Figure 182 Fritzing diagram of the connections between NodeMCU and the PN5180 Reader

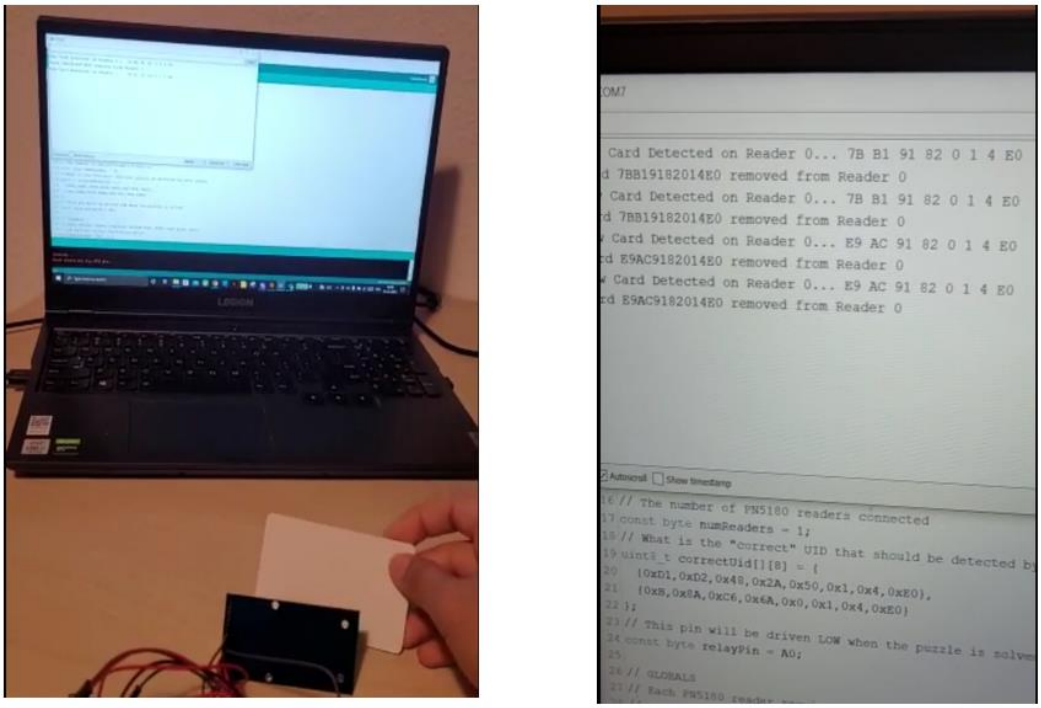


Figure 183 Testing of the reader and detection of the RFID Card with its ID

Enabling the SPI communication in Odroid

The detection of RFID tags using the NFC Reader on nodeMCU ESP32 worked on SPI communication interface. To make it work on droid, the SPI interface had to be enabled on the Odroid.

Serial Peripheral Interface

Serial peripheral interface (SPI) is one of the most widely used interfaces between microcontroller and peripheral ICs such as sensors, ADCs, DACs, shift registers etc.

SPI is a synchronous, full duplex main-sub node-based interface. The data from the main or the sub node is synchronized on the rising or falling clock edge. Both main and sub node can transmit data at the same time. The SPI interface can be either 3-wire or 4-wire.

4-wire SPI devices have 4 signals and they are:

- Clock (SPI CLK, SCLK)
- Chip select (CS)
- main out, sub node in (MOSI)
- main in, sub node out (MISO)

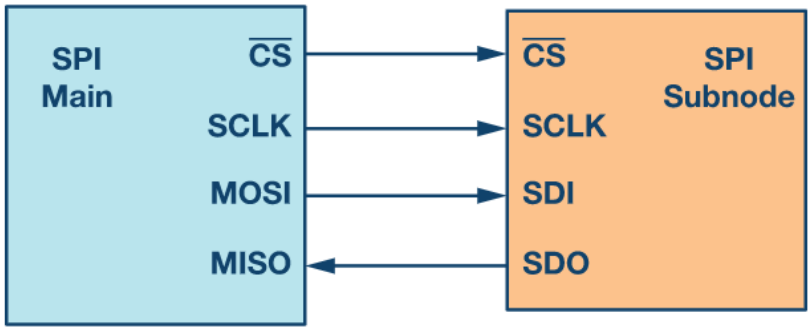


Figure 184 SPI configuration with main and a subnode

More information can be found here-

https://www.analog.com/en/analog-dialogue/articles/introduction-to-spi-interface.html.

Steps

• Device Tree Overlay was used to enable the GPIO functions. To know more about it https://wiki.odroid.com/common/application_note/software/device_tree_overlay

• Enabling the communication protocols SPI, I2C, UART and PWM using device tree overlay. https://wiki.odroid.com/common/application_note/gpio/enable_spi_i2c_uart_with_dtbo.

• Enabling the SPI feature on the board.

• https://wiki.odroid.com/odroid-xu4/application_note/gpio/spi#tab__odroid-n2.

• To check the details of the GPIO of the Odroid, Wiringpi was installed and used.

https://wiki.odroid.com/common/application_note/gpio/wiringpi.

Testing of SPI

Even if there are no other SPI hardware, one can test if the SPI feature works with a simple jump cable. Before testing, make sure that a jump cable is connected between SPI MOSI and MISO pin directly. Connect between pin #19 and #21. More details can be found in the following link. https://wiki.odroid.com/odroid-xu4/application_note/gpio/spi#tab__odroid-n22.

Figure 185 Pin layout of Odroid N2+

Preparation

Download a source code and compile.

target

```
sudo wget http://dn.odroid.com/Accessory/examples/spidev_test.c
sudo gcc -o spidev_test spidev_test.c
```

The help of the test utility.

target

```
# There's no option to show the help on this file.
# It shows if you enter this command with invalid arguments.
# So the '--help' option will show the help.
root@odroid:~# ./spidev_test --help
./spidev_test: unrecognized option '--help'
Usage: ./spidev_test [-DsbdlHOLC3]
  -D --device   device to use (default /dev/spidev1.1)
  -s --speed    max speed (Hz)
  -d --delay    delay (usec)
  -b --bpw      bits per word
  -l --loop     loopback
  -H --cpha     clock phase
  -O --cpol     clock polarity
  -L --lsb      least significant bit first
  -C --cs-high  chip select active high
  -3 --3wire    SI/SO signals shared
```

Figure 186 Preparation for the SPI Test

target

```
root@odroid:~# ./spidev_test -D /dev/spidev0.0
```

Figure 187 Terminal code to run the test

Figure 188 Results of SPI test

Yellow indicates that the SPI MOSI(19) and MISO(21) pins are connected using a jump cable and the SPI communication is taking place between them and red indicates no connection.

Enable the modules using **device-tree-compiler**.

target

```
sudo apt install device-tree-compiler
```

Change the status to **okay** of the SPI nodes on the device tree.

target

```
# SPICC0
sudo fdtput -t s /media/boot/meson64_odroidn2.dtb /soc/cbus@ffd00000/spi@13000 status "okay"

# SPIDEV0
sudo fdtput -t s /media/boot/meson64_odroidn2.dtb /soc/cbus@ffd00000/spi@13000/spidev@0 status "okay"
```

Check if it changed.

target

```
# SPICC0
root@odroid:~# fdtget /media/boot/meson64_odroidn2.dtb /soc/cbus@ffd00000/spi@13000 status
okay

# SPIDEV0
root@odroid:~# fdtget /media/boot/meson64_odroidn2.dtb /soc/cbus@ffd00000/spi@13000/spidev@0 status
okay
```

Then reboot to apply the changes.

Then you can check if the modules loaded.

target

```
root@odroid:~# lsmod | grep spi
spidev                  20480  0
spi_meson_spicc         20480  0
```

Check the **/dev/spidev0.0** file out as the below.

target

```
root@odroid:~# ls /dev/spidev*
/dev/spidev0.0
```

2.8 Safety Integrity Level (SIL) Basics -IEC 61508

IEC 61508 helps to ensure the functional safety of software for many industries. The 8 Parts of IEC 61508

The eight parts of the standard:

Part 0: Functional safety as it relates to the standard.

Part 1: General requirements.

Part 2: Requirements for E/E/PE safety-related systems.

Part 3: Software requirements.

Part 4: Definitions and abbreviations.

Part 5: Examples of methods for the determination of safety integrity levels.

Part 6: Guidelines on the application of Parts 2 and 3.

Part 7: Overview of techniques and measures.

Parts 1–3 contain the requirements of the standard. The rest spell out the guidelines and provide examples for development.

Safety Integrity Level (SIL) Basics

The focus of the safety standard is on functional safety. An important aspect of functional safety is assigning a Safety Integrity Level (SIL).

A safety function's SIL is a measure of how much risk it reduces.

The frequency and severity of dangers are related to SIL ratings. They calculate the level of performance required to maintain and achieve safety, as well as the likelihood of failure.

SIL 1, SIL 2, SIL 3, and SIL 4 are the four SILs. The higher the SIL, the more likely it is to fail. The stronger the safety criteria, the greater the danger of failure.

Safety Integrity Level	Probability of Failure on Demand	Risk Reduction Factor
SIL 4	$\geq 10^{5}$ to $<10^{4}$	100,000 to 10,000
SIL 3	$\geq 10^{4}$ to $<10^{3}$	10,000 to 1,000
SIL 2	$\geq 10^{3}$ to $<10^{2}$	1,000 to 100
SIL 1	$\geq 10^{2}$ to $<10^{1}$	100 to 10

Figure 189 Probability of failure depending on the SIL

Hazard and Risk Analysis for Determining SILs

Ensuring functional safety requires a hazard analysis and risk assessment of equipment under control.

A hazard analysis identifies all possible hazards created by a product, process, or application. This determines the safety function requirements for the safety standard.

You'll need to do a risk assessment for each hazard you discover. This evaluates the possibility or frequency of a hazard occurring, as well as the severity of the repercussions if it does. The safety standard's safety integrity requirements are determined through risk assessments. They're also important for evaluating the SIL needed to mitigate danger.

You can use either qualitative or quantitative analysis to assess risk. A specific method isn't required. One way you can assess risk is to create a requirements traceability matrix and do a failure modes and effects analysis (FMEA).

MISRA Definition

MISRA provides coding standards for developing safety-critical systems.

MISRA is made up of manufacturers, component suppliers, and engineering consultancies. Experts from Perforce' s static code analysis team (formerly PRQA) are members of MISRA, too.

MISRA first developed coding guidelines in 1998. These were specific to the C programming language. Since then, MISRA has added a coding standard for C++.

Advantages of MISRA Standards

You can use MISRA standards to ensure your code is:

Safe

Secure

Reliable

Portable

The MISRA C coding standard was originally written for the automotive embedded software industry. But today, MISRA standards for C and C++ are widely used by embedded industries — including aerospace and defense, telecommunications, medical devices, and rail.

Steps to Achieve MISRA Compliance

Achieving MISRA compliance takes knowledge, skill, and the right tools.

Here are seven steps to comply with MISRA:

1. Know the Rules

You need to know the MISRA coding rules pertinent to which version of C or C++ you're using.

2. Check Your Code Constantly

Continuously inspecting your code for violations is the best way to improve quality.

3. Set Baselines

Embedded systems come with legacy codebases. By setting baselines, you can focus on making sure your new code is compliant.

4. Prioritize Violations Based on Risk

You could have hundreds or even thousands of violations in your code. That's why it's important to prioritize rule violations based on risk severity. Some static code analysis tools can do this for you.

5. Document Your Deviations

Sometimes there are exceptions to the rule. But when it comes to compliance, every rule deviation needs to be well-documented.

6. Monitor Your MISRA Compliance

Keep an eye on how MISRA compliant your code is. Using a static code analyzer makes this easier by automatically generating a compliance report.

7. Choose the Right Static Code Analyzer

Choosing the right static code analyzer makes everything else easy. It takes care of scanning your code — new and legacy — for violations. It prioritizes vulnerabilities based on risk.

CHAPTER THREE – CONCLUSION

3.1 Further System Development

Due to specific time limits, the Mover is limited to its function. It can however be developed by using further programming and using more accurate components.

Further development in Line following and station detection are,

- IFM3D camera is not an efficient sensor to use for line following. Hence use an optical camera (a simple USB camera) for line following. 3D camera must be used for obstacle detection and avoidance and in future can be used for localization and mapping as well
- AR Tag detection testing was done using the laptop web camera. It must be implemented and tested on a USB camera.
- Both line following and Station detection using AR Tags must be then tested and implemented on ROS on the Odroid N2+.
- RFID Tags detection was tested using the PN5180 NFC reader using a NodeMCU ESP32. It works on SPI communication. SPI communication on Odroid N2+ was enabled and further the code for RFID Tag detection has to be tested on Odroid N2+ using SPI Communication and then has to be implemented on ROS for an alternative way to detect stations.

REFERENCES

Microsonic GmbH, (2021, December 12). Microsonic pico+35_F_A. In https://www.microsonic.de/, Retrieved 19:01, April 2, 2022 from https://www.microsonic.de/DWD/_111327/pdf/1033/microsonic_pico+35_F_A.pdf

Wikipedia contributors. (2021, December 24). IO-Link. In *Wikipedia, The Free Encyclopedia*. Retrieved 18:53, April 2, 2022, from https://en.wikipedia.org/w/index.php?title=IO-Link&oldid=1061795522

Wikipedia contributors. (2022, March 9). CAN bus. In *Wikipedia, The Free Encyclopedia*. Retrieved 18:43, April 2, 2022, from https://en.wikipedia.org/w/index.php?title=CAN_bus&oldid=1076161431

Wikipedia contributors. (2022, April 1). Universal asynchronous receiver-transmitter. In *Wikipedia, The Free Encyclopedia*. Retrieved 18:54, December 2, 2021, from https://en.wikipedia.org/w/index.php?title=Universal_asynchronous_receiver-transmitter&oldid=1080549184

Odroid Wiki. (2021, July 13). Device Tree Overlay. In https://wiki.odroid.com/, Retrieved April 3, 2022, from https://wiki.odroid.com/common/application_note/software/device_tree_overlay.

Odrive Robotics. (2017, April 06). Odrive simple CAN Protocol. In Odrive Documentation. Retrieved April 3, 2022, from https://docs.odriverobotics.com/v/latest/can-protocol.html

Odrive Analog Devices Inc. (1997, Feb 10). SPI. In analog.com. Retrieved April 3 2022, from

https://www.analog.com/en/analog-dialogue/articles/introduction-to-spi-interface.html

Odroid Wiki. (2021, July 13). WiringPi and Python Wrapper. In https://wiki.odroid.com/, Retrieved April 3, 2022, from https://wiki.odroid.com/common/application_note/software/device_tree_overlay.

Automation24 GmbH. (2010, April 12). Microsonic pico+35_F_A. In Automation24 GmbH. Retrieved April 3, 2022, from https://www.automation24.biz/ultrasonic-sensor-microsonic-pico-35-f.