Technische Universität Dortmund
Department of Biochemical and Chemical Engineering
Chair of Process Dynamics and Operations
Prof. Dr. Sebastian Engell

DEVELOPMENT OF LOCAL POSITIONING SYSTEM FOR A PIPE-LESS PLANT

Automation & Robotics
Group Project SS18

Group Members:

Abdulrahman Abouelkhair(198803)
Medhini Rajagopal Balamurugan(198735)
Stefan Rottstegge(191455)
Stephan Vette(198907)

Supervisors:

Afaq Ahmad
Marina Rantanen-Modéer

Abstract

Introduction

The pipe-less plant at the Process Dynamics and Operations group is an experimental setup of Automated Guided Vehicles (AGVs) moving between various stations. The AGVs dynamically change trajectories in operational mode based on a Model Predicative Control (MPC) scheme with the objective to get from one station to the other while at the same time avoiding each other. The current positioning system is based on pattern recognition where the system tracks each AGV based on a unique pattern of LEDs via a camera that overlooks the plant.

Motivation for new positioning system

The vision based positioning system displays some flaws and should be replaced by a system more adapted to the actual operational environment of the experimental plant. The following problems have been identified in the current setup of the system:

- No position feedback in bright day-light conditions
- A "*fish-eye camera lense problem*", meaning that the position error grows with the distance from the center of the image caused by the distortion a wide angle lense creates
- Problems related to the software implementation restricting the usage of incoming information from the camera

Project objective

The project aimed at first evaluating different potential positioning systems, selecting one of them based on defined metrics and then to develop a proof-of-concept based on the chosen technology for the experimental pipe-less production plant in a model driven fashion. After going through four different alternatives including a triangulation based methods for indoor applications, another pattern recognition based method (such as QR-codes), map-based localization, and Radio-Frequency Identification (RFID), the latter was chosen for further evaluation and implementation.

Theoretical Background

RFID is a versatile technology with multiple application areas, e.g. access control, race tracking and positioning. Automated multi-agent systems are increasingly utilizing RFID for localization as the technology has been proven to have many advantages over vision based positioning systems.It uses electromagnetic fields to automatically identify and track tags attached to objects. There are two potential ways to implement an RFID localization system. One option is based on active RFID transmitters and reader with wide coverage areas that can be placed along the edges of the plant area and on each AGV. The other option is based on comparatively many passive tags, uniformly placed, on the ground of the plant area and active readers on the AGVs. The latter option was chosen for the project based on cost efficiency, system scalability and from literature proven applicability.

Hardware

An RFID system is made up of two parts: a tag and a reader. RFID tags are embedded with a transmitter and a receiver. The RFID component on the tags has two parts: a microchip that stores and processes information, and an antenna to receive and transmit a signal, which partly contains the unique ID of the tag. The hardware components which are added to the AGVs comprise an RFID antenna, an RFID reader and a WiFi module. All the devices are supplied with 5V. The antenna is connected to the reader via a coaxial so called "*SubMiniature version A*"(SMA) cable and the reader communicates via an RX/TX interface. The RFID reader and the WiFi-module are connected via wired serial communication. The WiFi module then transmits the reader data through the local network, using TCP/IP communication. The

system data is lastly received by a PC that represents the control hub of the plant through a framework implemented in C Sharp (C#).

Communication

The WiFi-Module is connected to the same router as the PC on which the system user interface is running. A TCP-Client was established in the C# framework in order to handle incoming RFID data. The WiFi-module continuously sends data and the algorithm calculating AGV position prompts for this data as position and/or orientation of an AGV needs to be computed.

Localization

The implemented positioning algorithm requires the ID and the received signal strength indication (RSSI) of at least three RFID tags to calculate the position of the antenna. The RSSI gives a relation between the detected tag and the distance to it, in other words, a radius. The system has a record of the position of each tag and the ID of each tag hence holds information about the uniquely defined position of the tag. With three positions and the three corresponding radii, one can use trilateration to compute the position of the antenna.

Initialization Procedure

Initially the system cannot know the orientation of the AGV even if it can read enough RFID tags, as the tags only provide the x and y coordinates of the antenna itself (which is located off center on the robot). The main goal of the initializing procedure is to estimate both the starting position and orientation. During the turn, the robot will make a short stop every 45Â° and estimate the position of the antenna at that point. At the end of the procedure, the algorithm calculates the center of the eight recorded points. It takes two positions plus their corresponding positions at the other side (plus 180°) and computes the intersection of the linear function which goes through the position and its corresponding point. To estimate the orientation the algorithm computes the angle between the estimated position and the position of the antenna at 360°/0°.

Demonstrator and results

A proof-of-concept for an RFID based localization system has been built in a model-driven fashion. The new system was simulated using Matlab to evaluate its applicability and overcome engineering problems at an early stage of the design phase. An initialization algorithm has been designed and implemented to calculate the starting position and heading of the robot. Simulations show an average position error of 0.5 cm. Analogously, the orientation of an error of around 10°. The system prototype consists of a reader and an area of nine tags (3x3). The measured position error in the experimental setup was on average 2.5 cm. The error of the orientation was on average about 23°. This significant difference to simulated results was caused by two main reasons: Firstly, some tags could not be detected by the reader. Secondly, the test board was quite small as it consisted of so few RFID tags which meant that it was not representative enough of the intended design. To summarize, it can be said that on the one hand a rather low-cost, scalable solution was designed which can work in any area independent for any light conditions. On the other hand, the accuracy of the technology is not very reliable according to the experiments made. This can be improved in further work with a better system setup and an improved localizing algorithm.

Contents

List of Figures

List of Tables

1 Introduction

In today's world, the development and implementation of the positioning system for the autonomous vehicle in a confined space remains to be a major issue and hindrance to a better control system. Though there exists many types of local positioning systems, the precision remains to be still a challenge. This problem becomes critical in a place of no GPS access. This project aims at investigating various methods of indoor localization and to develop a proof-of-concept for the existing pipeless plant setup.

In the past years students and researchers at the Process Dynamics and Operations group at the TU Dortmund have developed this plant with vision based positioning system which need to be replaced to improve the overall efficiency of the system. Both the old and newly implemented techniques are written in C# that sends the position update to the Python based controller code.

In this project, various techniques were discussed and "*Radio Frequency Identification*" was chosen to be the ideal technique which would be further discussed in section 3 and 4. The hardware implementation is explained in section 7. Results of the simulation and experiment is discussed in section 6 and 7.

Based on the experiment, conclusion and future work are given in section **??** and **??** respectively.

2 Pipeless Plant

In chemical industry, pipeless plants are used due to its high flexibility level. In these type of plants, "*Automated Guided Vehicles(AGVs)*" are used to transport the vessel from one processing station to another. Thus, for each and every batch, the AGV transport the vessel to various stations to create an end product, making the pipeless plant multi-product and multipurpose chemical production. In this way, piping and the associated cleaning is eliminated, thus aiding in cost and energy efficiency.

2.1 Existing setup

The pipeless plant framework in TU Dortmund, developed by the Process Dynamics and Operations Group (DYN), consists of four AGVs, two color stations, one mixing station and one storage station. Each station has a role in producing batch of plaster art, for example mixing or filling the product in the vessel. The stations and the AGV are controlled by a Programmable Logic Controller. The vessel on the AGV is moved from one station to another based on the schedule to produce a batch. Regarding the positioning system, the plant uses camera to identify the LED pattern on the AGV. Each AGV has a unique pattern that distinguishes it from another.

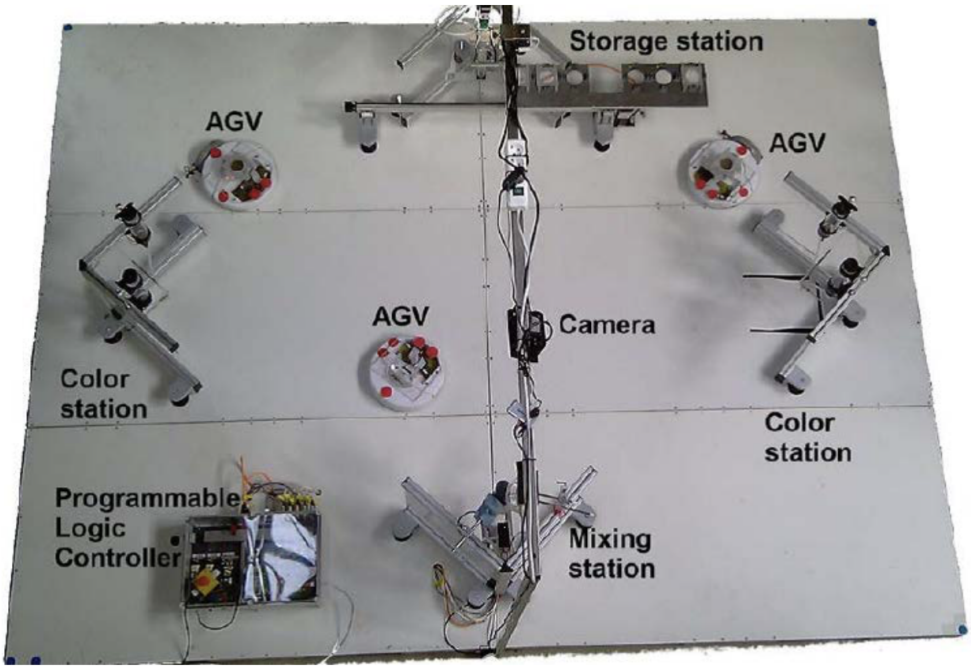


Figure 1: Existing setup

2.2 Problems with the Existing Setup

Since the existing setup uses vision based positioning system, the plant suffers various disadvantages as described below:

- During bright day-light conditions, no position is updated by the camera. This is due to the fact that the threshold of the sunlight and the LED on the AGV becomes equal that the camera fails to detect any pattern, thus affecting the whole system efficiency.
- The camera suffers the so called "*fish-eye camera lense problem*", meaning that the position error is proportional to distance from the center of the image. This is caused caused by the distortion a wide angle lens.
- The restriction of usage of incoming information from the camera during software implementation.
- The percentage error and the processing time increases with the increase in number of robots to be localized thus affecting the controller input.

All these cons added up together cause a deterioration in the accuracy of the position of the AGV, which in turn affects the controller leading to a drop in system efficiency.

2.3 Motivation

The disadvantages of the vision based positioning system as discussed above leads to the urge for creating a localization that overcomes all the cons already suffered.

This projects aims to research about alternative positioning system and to evaluate with selected method by simulation. The ultimate goal is to develop a positioning system with improved position precision and to compare and develop practical proof-of-concept.

3 Selection Process

Since the main aim of the project is to improve the positioning system of the existing setup, several other techniques were discussed, ending up in four methods namely "*triangulation*","*pattern recognition*","*radio frequency identification*", "*map-based localization*". The pros and cons were listed and the mentioned techniques were compared. The following section deals with a brief description of the above mentioned techniques.

3.1 Triangulation

Since the plant has a specified size in which the location of multiple objects has to be performed the method of triangulation is one promising technic in which research was made. Triangulation was already a common principle of measurement in the 18th century and it is divided into active and passive triangulation. Passive triangulation is a geometrical method based on two measurement stations which positions are known exactly. At these two measurement points angels of the desired point in space are measured to compute the localization in the specified coordinate system (x, y, z) with trigonometrical formulas. With respect to the system setup used in the 18th century nowadays two cameras are installed to perform a geographical method of 3D object-data estimation as shown in fig. 2 [2].

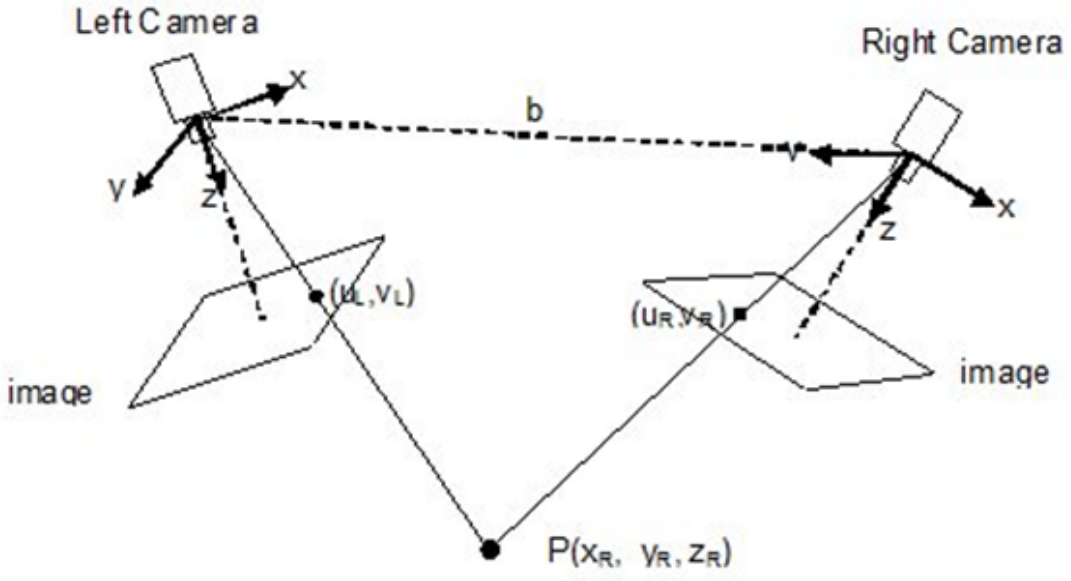


Figure 2: Passive triangulation setup with two cameras [3]

To solve the problem of position estimation, it is necessary to know the parameters of the left and the right camera visualized in the figure. In theory the triangulation is trivial, since each

and every point of the images of the respective cameras maps to a line in 3D space. If a pair of corresponding points, in the case of the pipesless plant it would be an AGV is found, the projection of a point x in 3D space can be computed. Active triangulation in comparison to passive triangulation needs one camera and at least one source of structured light (e.g. Laser). The geometrical location and orientation of the camera and light source in space need to be known. Two possible setups with either a laser point or a stripe as structured light are shown in fig. 3 [3][?].

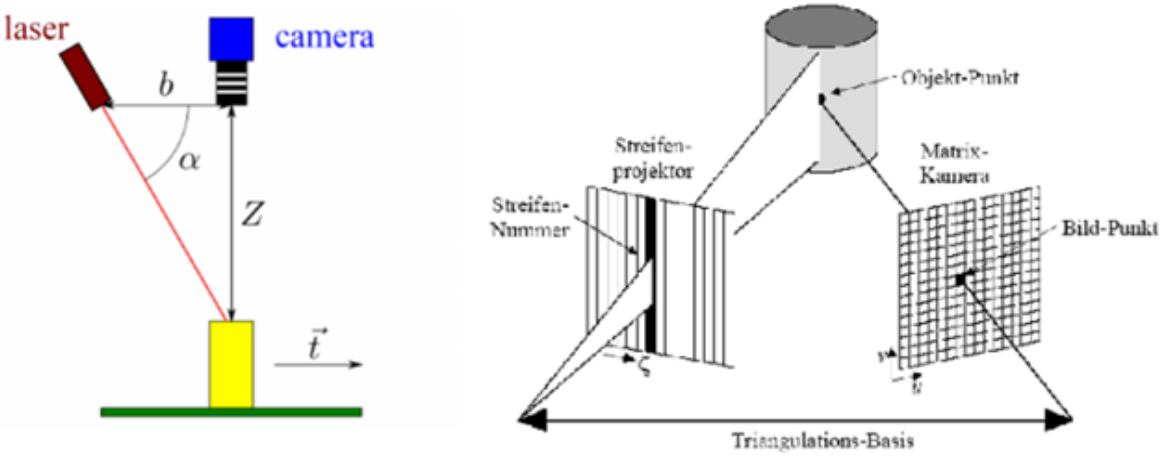


Figure 3: Active triangulation

To solve the active triangulation problem, the structured light has to point an object which location is desired to estimate. If this point is found on the 2D image of the camera, a triangulation with basic trigonometrical formulas which are using the properties and parameters of the camera and light source can be performed and the position of the AGV can be estimated.

Implementation

One possible way to implement a solution for the passive triangulation is to attach 2 high resolution cameras with USB 3.0 on two edges of the plant as shown in fig. 4.

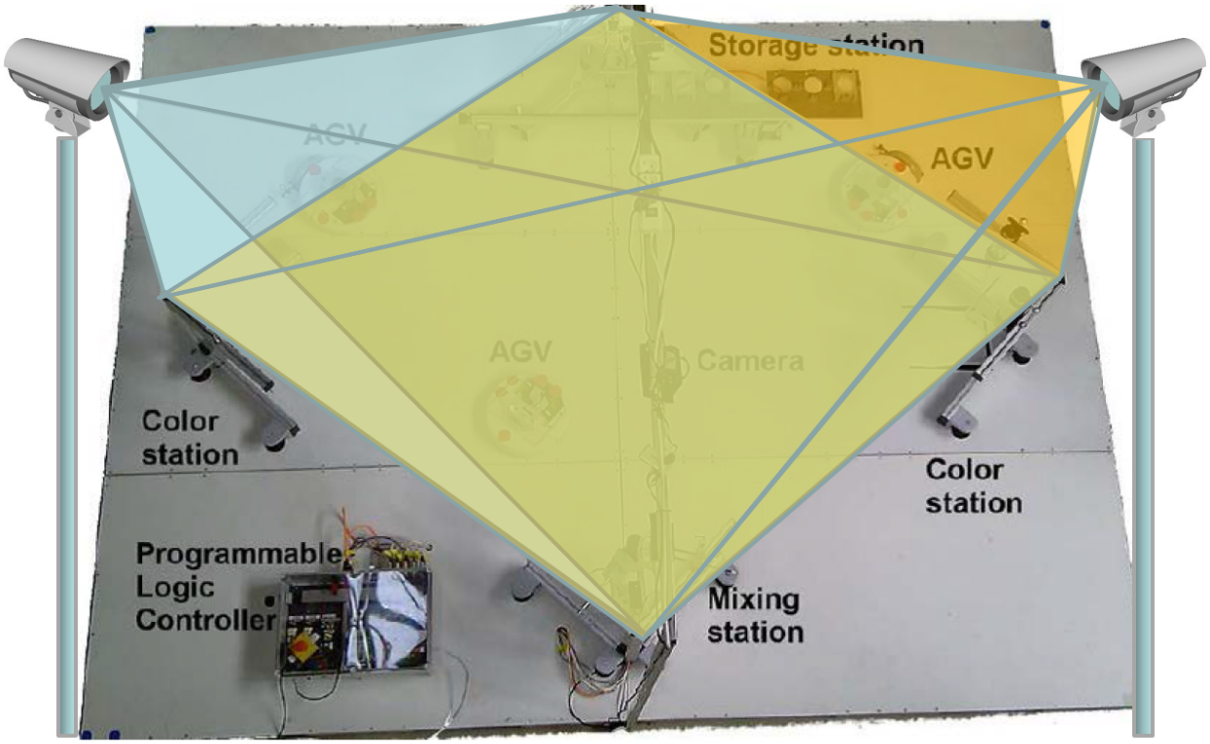


Figure 4: Implementation of passive triangulation

The left and right camera are sequentially taking pictures which are transmitted to the plants computer where the image processing takes place.
Based on the research made , two tables with advantages and disadvantages of the two triangu-

lation systems are created.

Passive Triangulation	
Pro	**Con**
Upgrade to USB 3.0 for faster data transmitting possible	Light dependent
Upgrade to a camera with higher resolution to reduce measurement error possible	New concept of orientation may be needed
No Fish-Eye-Lense problem	Limited range of observation
Low cost	

Table 1: Pros and cons points of passive triangulation

Active Triangulation	
Pro	**Con**
Upgrade to USB 3.0 for faster data transmitting possible	New unknown laser technology is needed
Upgrade to a camera with higher resolution to reduce measurement error possible	High costs for several lasers (one per AGV)
Easy detection of laser points on camera image	Laser needs to move while AGVs are moving
	Limited range of observation
	Light dependent

Table 2: Pros and cons points of active triangulation

3.2 Pattern Recognition

Summary

In this type of localization, estimation of the robot is done in indoor environments using only on-board sensors, namely a web-cam and a compass. The ceiling of the plant is constructed with a pattern of static landmarks whose positions are known a priori. All landmarks are indistinguishable against each other and might additionally be distributed along the ceiling in a periodic pattern. The landmark attached to the ceiling can be lights, QR codes, sensors or other reference points. The ceiling is used, since it is immune to changes. A camera is installed on the robot, which takes snapshots of the ceiling. The robot pose relative to the landmark is calculated with the help of the distance of the landmark to the center of the image and its angle relatively to the direction of the robot motion. An IMU device is additionally used to give the absolute orientation of the robot in the plant. The Markov Localization (ML) algorithm is used to estimate the belief grid of the robot position inside the environment.

Figure 5: Ceiling with periodic patterns of lamps acting as landmarks.

Implementation

The goal is to compute the pose of a mobile robot inside an indoor environment using a camera and an IMU device. As mentioned, Markov Localization is used to create a belief grid of the robot in the plant environment. This is done with the help of the snapshots of the ceiling taken

by the camera. As seen in the figure 6, the blue and black areas have lower belief and green and yellow areas have higher belief. The obtained pattern is evaluated and based on the pattern, the position of the robot is estimated. Thus with the help of the camera and the IMU device, both the position and orientation is obtained.

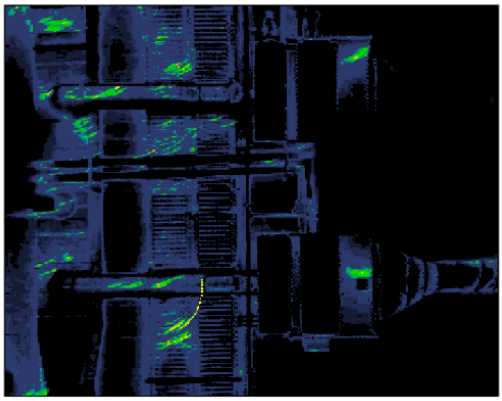


Figure 6: Belief grid of the robot in the plant

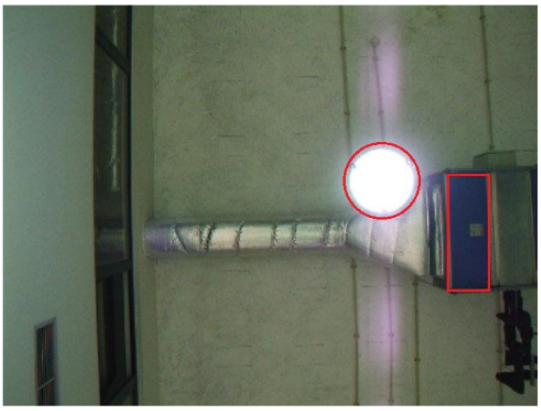


Figure 7: Snapshot of the ceiling

Pro and con

Based on the research,the advantages and disadvantages of Mobile Robot Localization based on Pattern Recognition are created.

Pro	Con
The ceiling is usually immune to changes as a reference and implement landmarks on the ceiling itself	Complex and many changes have to be added to the plant
No Fish-Eye-Lense problem	Cost intensive
	Light dependent

Table 3: Pros and cons points of Mobile Robot Localization based on Pattern Recognition

3.3 RFID[1]

Summary

One of the possible solutions to solve the challenging problem of indoor localization is the use of the Radio-frequency Identification (RFID) technology. The main areas of this technology is indeed still supply chains, transport, manufacturing, personnel access, animal tagging, toll collection [4], but also has become popular in localizing objects and persons. Where in the main applications only the identification has to be realized, also the strength of the signals is important to estimate the position of a certain object.
The main idea of those systems is that a reader detects a tag and reads its information. The technology can be divided into three main types: passive, semi-passive and active systems. A passive system, like it is been shown in fig. 8, consists of a reader, which is connected to an antenna and a computer and a passive tag.

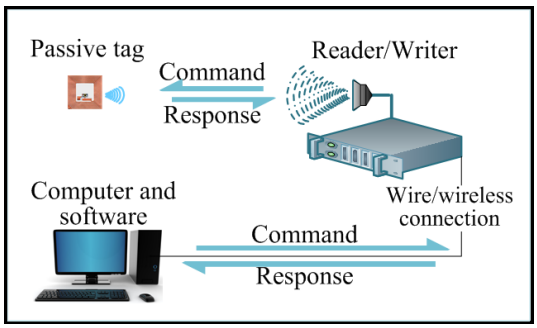


Figure 8: Passive RFID System

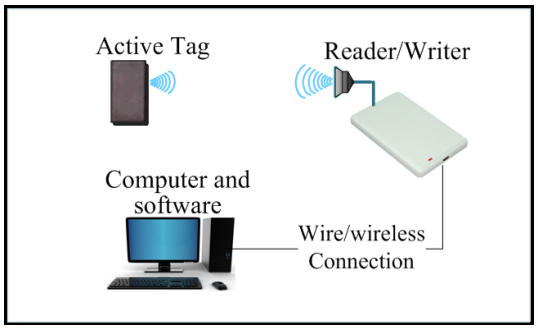


Figure 9: Active RFID System

The system is called passive, because the power supply is realized by the radio signal of the reader. In case where the tag is in the reading range of the reader, the tags gets enough power to send predefined information (for example ID) back. The active system (see fig.9) in comparison has an active tag which has an own power supply. The semi-passive tag has a battery build in that the tag has more power to communicate, but is not used to generate radio frequency signals.
Another classification of RFID systems is the frequency of the radio waves. It can reach from 0.135 MHz (Low Frequency) to 5875 MHz (Super High Frequency). The table 4 gives an overview about the systems related to reading ranges, reading rates and the ability to read near metal or water.

It can be seen that the passive systems in general have a smaller reading range then the active systems and has a bigger data rate. But it has also to be take into account, that passive tags are cheaper then active tags.

[1]Stephan

	LF	HF	UHF	SHF
FR (MHz)	< 0.135	3~28	433-435, 860-930	2400~2454 5725~5875
RR(P)	≤ 0.5 m	≤ 3 m	≤ 10 m	≤ 6 m
RR(A)	≤ 40 m	300 m	≤ 1 km	≤ 300 m
TRR	Slower ⟷ Faster			
ARMW	Better ⟷ Worse			

FR: Frequency Range
RRP: Typical Reading Range of Passive Tags
RRA: Typical Reading Range of Active Tags
TRR: Tag Reading Rate
ARMW: Ability to Read near Metal or Water

Table 4: Overview RFID systems

Implementation

There are mainly two different ways to realize a localization system of the AGVs in the pipeless plant. Based on the fact that the plant has a size of 3 by 4 meter, the tracking can be carried out with a passive system in which a couple of passive tags on the floor can be used as landmarks. In this case the reader plus the antenna would be placed on the AGV and localize with the help of the detected tags. The other systems consists of three or four reader in each corner of the plant and an active tag on each AGV.

Pro and con

Based on the research made, two tables with advantages and disadvantages of the two RFID systems are created.

Active RFID system	
Pro	**Con**
Light independent	Prototype more expansive (3 reader + avtive tags)
Space unlimited	Datarate is related to the amount of detected tags a the same time
Localization only has to be realized in a bigger area - medium accuracy	Anticollision need, cause more AGVs are used at the same time
Wired communication between reader and computer possible	Signal strength can be influenced by envirnonment (metal or water)
Simple algorithm (Trilateration)	

Table 5: Pro and cons of active RFID system

Passive RFID system	
Pro	**Con**
Light independent	Communication between AGV and computer has to be realized
Space unlimited	Data rate is related to the amount of detected tags a the same time
Localization only has to be realized between four tags (small area) - high accuracy	Anticollision need, cause more tags are detected at the same time
Simple algorithm (Trilateration)	
Prototype cheap (1 reader + passive tags)	

Table 6: Pro and cons passive RFID system

3.4 Map-Based Localization[1]

Summary

AMCL (Adaptive Monte Carlo Localization) is a probabilistic localization system for a robot moving in 2D. It implements the adaptive (or KLD-sampling) Monte Carlo localization[5][6] approach, which uses a particle filter to track the pose of a robot against a known map.

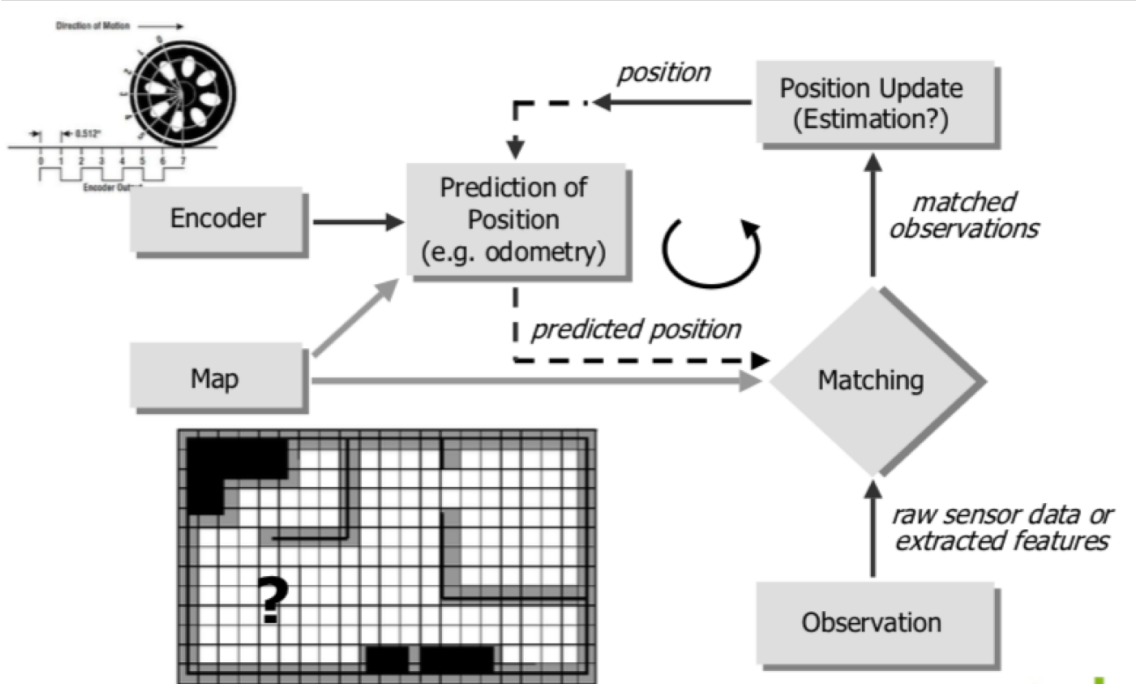


Figure 10: Adaptive Monte Carlo localization

amcl takes in a laser-based map, laser scans, and Odom scan, and outputs pose estimates (see fig.10). On startup, amcl initializes its particle filter according to the parameters provided. Note that, because of the defaults, if no parameters are set, the initial filter state will be a moderately sized particle cloud centered about (0,0,0).

Implementation

To implement such a technique a global and local map should be created as shown in fig. 11 and fig. 12.

- SLAM (Simultaneous Localization and Mapping) is a technique used in mobile robotics in which a robot builds a map of an unknown environment, keeping at the same time track of its localization in this environment.

Figure 11: Global Map

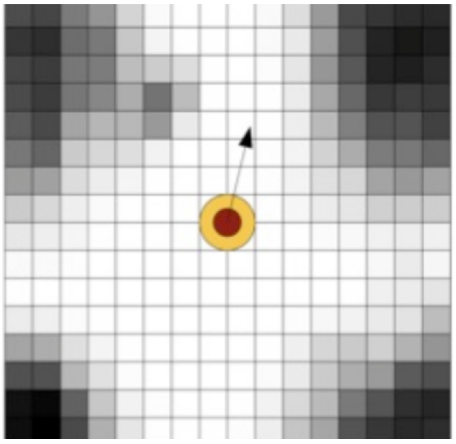


Figure 12: Local Map

- Adaptive Monte Carlo Localization
  Given a map of the environment, the goal of the algorithm is for the robot to determine its pose within the environment.
  At every time t the algorithm takes as input the previous belief $X_{t-1} = \{x_{t-1}^1, x_{t-1}^2, ..., x_{t-1}^M\}$, an actuation command u_t, and data received from sensors z_t; and the algorithm outputs the new belief X_t.

- Orientation Correction

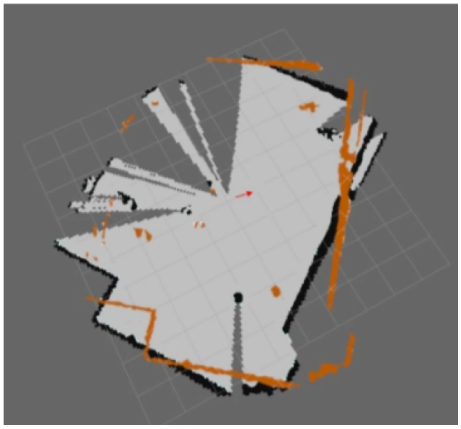


Figure 13: Robot Orientation

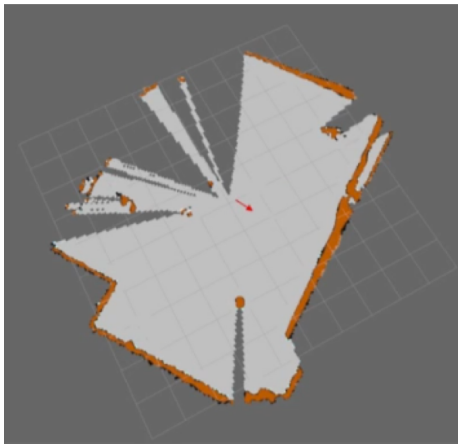


Figure 14: Correction with global map

Initially the robot assumes a position as shown in fig.13, and as it moves it begins to re-correct it's estimated orientation using the static obstacle with the global map as a reference as shown in fig. 14.

Pro and con

Based on the research made, two tables with advantages and disadvantages of the two Map-Based Localization systems are created.

Using Ultrasonic Sensor	
Pro	**Con**
Cheap €3/each	Scan angle 30°
Easy hardware Installation	Similar landmarks cause localization error
Faster feedback than the previous camera based localization system	High computational effort for large plant
Scan range 4.5 meters	Robots should start at every launch from static home position
Different map based localization algorithms are available	

Table 7: Pro and cons of Localization using Ultrasonic Sensor

4 Theoretical Background

4.1 Radio Frequency Identification

After in depth analysis of various localization methods the Radio Frequency Identification[7][8] was chosen due to its various advantages. This technology involves a reader and a tag which placed on the object needed to be tracked. The reader is continuously sending the radio waves, and when the tag is within the range of the reader, it sends a feedback signal to the reader as shown in fig. 15. The reader can track multiple tags at the same time.

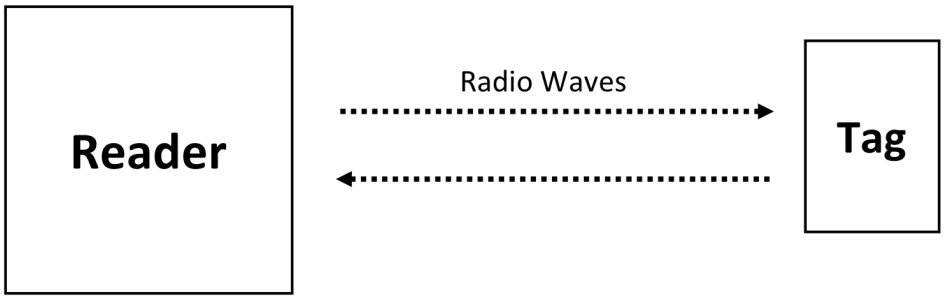


Figure 15: Reader Tag

4.1.1 RFID System

Regarding the tags as shown in fig.16, it can be either

- Active tag: which has its own power supply,
- Passive tag: which relies on the radio waves as its source of energy that come from the reader ,
- Semi-passive tag: which has power supply, but for transmitting the feedback, it relies on the signal coming from reader.

4.1.2 Working Principle

The RFID consists of three main parts:

- Generator: which generates the radio waves.
- A signal detect: which receives the feedback signal from the tag
- Micro-controller: which process the information from the generator and the detector

The tags consists of:

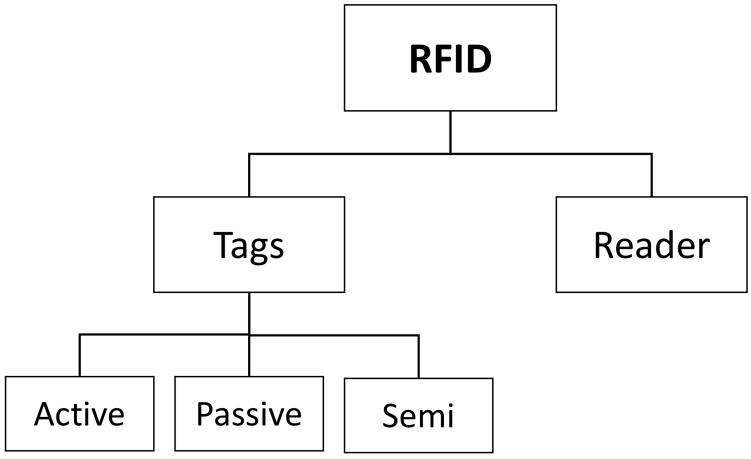


Figure 16: RFID System

- Transponder: that receives radio waves and sends the feedback
- Rectifier Circuit: which stores the energy coming from the wave across the capacitor, and this energy is used as a power supply for the controller as well as the memory

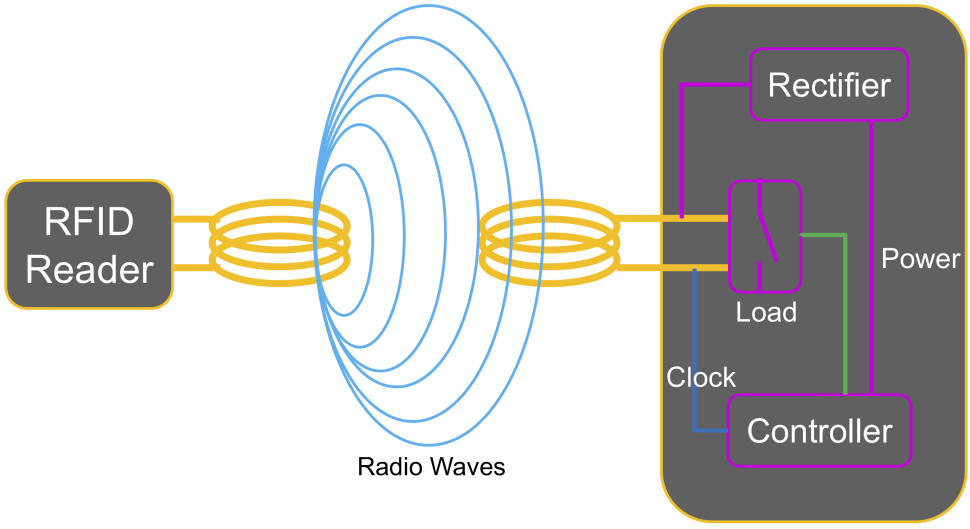


Figure 17: Inductive Coupling

The whole process of sending the information between the tag and the reader is based on principle of "Inductive coupling". The reader is continuously sending radio waves with particular frequency. In this case, the reader and the tag should be close to each other as shown

in fig. 17. The field which is generated by the reader is used to couple with antenna of the tag, and due to the mutual coupling, the voltage is induced across the coil of tag. This voltage, with a particular frequency, is rectified and used as power supply for the controller and is used to derive synchronization clock for the controller.
When the load circuit is connected to the coil, the current starts flowing through the load. Therefore, when the load is switched on and off , the current will be turned on and off respectively leading to the induction of particular voltage in the reader. This method of switching the load is called load modulation. Thus, with the help of load modulation with respect to the data stored in the tag, the value of the induced voltage can be modified, leading to the generation of modulation on carrier frequency, thereby sending the data to the reader.

4.2 Trilateration[2]

Trilateration is a method to compute the intersection point of three circles/spheres. For this, it is necessary to know the three center of the circles/spheres plus their corresponding radii. The basic idea to estimate the intersection point is to use the mathematical description of a sphere:

$$r^2 = (x - x_1)^2 + (y - y_1)^2 + (z - z_1)^2 \tag{1}$$

where ($P_n = (x_n, y_n, z_n)$) is the center of the sphere [9]. A few assumption can be made to simplify (1) for the 2D indoor localization on a floor. First of all, the z-component of all spheres can be neglected. Another assumption is that we define the origin of the first circle as the center of the coordinate system, the second along the x-axis with an distance (d) and the third shifted in x- (i) and y-direction (j), which is illustrated in following fig.

With known positions of the center of the circles d, i and j can be computed in the following way[9]:

$$d = |P_2 - P_1| \tag{2}$$

$$e_x = \frac{1}{d}(P_2 - P_1) \tag{3}$$

$$a_x = P_3 - P_1 \tag{4}$$

$$i = e_x \cdot a_x \tag{5}$$

$$a_y = (P_3 - P_1) - i * e_x \tag{6}$$

$$e_y = \frac{a_y}{|a_y|} \tag{7}$$

$$j = e_y \cdot a_x \tag{8}$$

It has to be notice that P_1,P_2 and P_3 are 2D vectors, which represents the x- and y-coordinate of the points.
After knowing these values, the relative distance from the origin of the coordinate system can

[2]Stephan

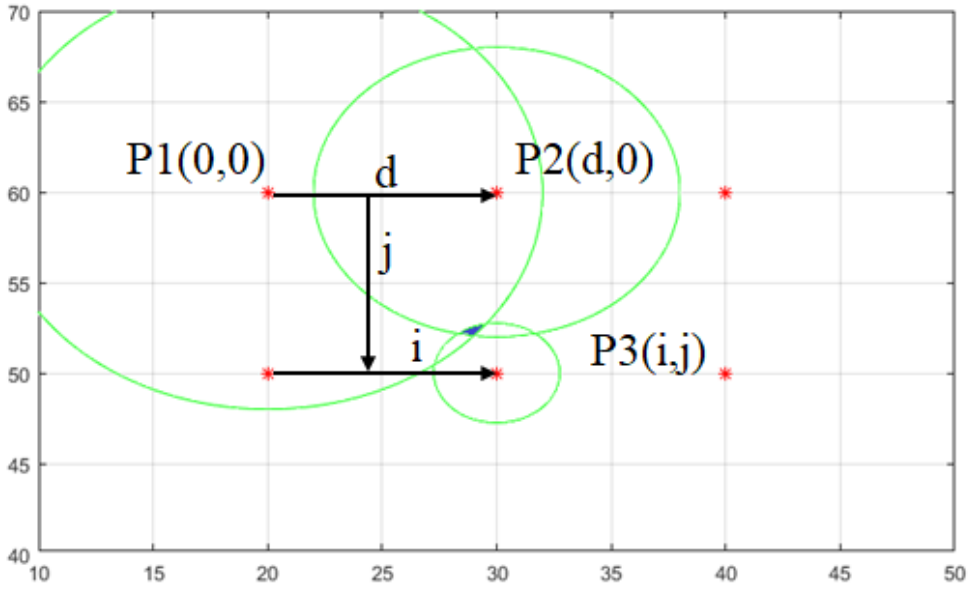


Figure 18: Overview Trilateration

be computed with the help of (1) and the center of the circles P_1(0,0), P_2(0,d) and P_3(i,j) as follows:

$$x_t = \frac{r_1^2 - r_2^2 + d^2}{2 * d} \tag{9}$$

$$y_t = \frac{r_1^2 - r_3^2 + i^2 + j^2}{2 * j} - i * \left(\frac{x_t}{j}\right) \tag{10}$$

The absolute position of the intersection point is computed in following way:

$$P = P_1 + e_x * x_t + e_y * y_t \tag{11}$$

It can be seen, that those equations are using the first two points plus radii to estimate the x-coordinate and first and third point plus the estimated x-coordinate to estimate the y-coordinate.

4.3 ...

5 Hardware[3]

5.1 RFID reader and antenna[4]

The RFID reader from KTS Systeme (see fig.19) is a HF Modul (frequency around 13.56 MHz). It contains a full-fledged microcontroller with a high-performance RFID transceiver IC. It has a 1.27 mm pitch pin-headers for THT mounting. The connection to an external antenna can be realized via a Single ended 50Ω connection or via Pin Header U.FL. jack, which was used in this project.

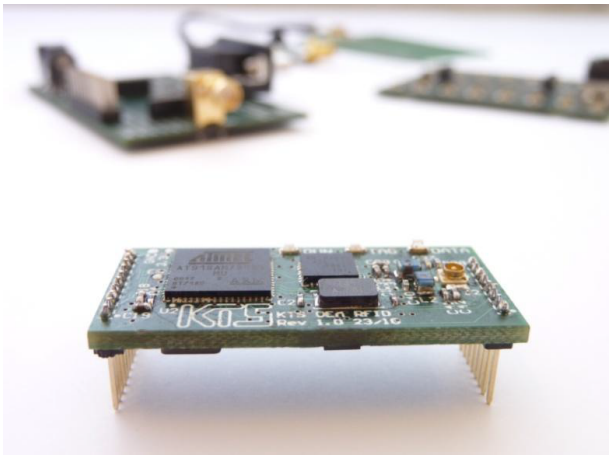


Figure 19: RFID reader KTS Systeme RFIDM1356-001

The communication to other devices is realized via a UART compatible serial interface via pin 6 (RX) and 7 (TX). The power supply is a 5 V DC connection via pin 1 (VCC) and pin 10 (GND). The reader is standardized to ISO 15693 and ISO14443A/B and has the overall dimensions 36 x 16 x 4 mm [LxWxH][10].
The reader has three LEDs:
- Green: Run - Lights when reader receives power
- Yellow: Tag - Lights when a tag is detected
- Red: Data - Lights when data transfer to or from a tag

To configure the reader, KTS Systeme also provides a software (Tag2Image) for free. The reader was configured to scan the environment in an automatic anti collision mode (AT+Scan=AC,RSSI). Anti collision means that multiple tags can be detected at the same time and is highly important in this project. The output of the scan is a continuous information of the Identification (ID) and the Received Signal Strength Indicator (RSSI) of the detected tags. For example: SCAN:+UID=E00402000018313E,+RSSI=7/6 means that the tag with

[3]Stephan
[4]Stephan

the ID (in hex) E00402000018313E was detected with a RSSI of 7/6. For the RSSI is the first number the value for the main and the second for the auxiliary receiver channel. In this project only the first number of the RSSI was used. The RSSI is an integer value from 0 to 7 and gives an information about the distance between the antenna and the detected tag. 0 stands for the maximum reading range which was mentioned to be around 15 cm. A detailed relation was figured out experimental during the project and will be explained later in this report. An AT Command Reference Guide is also available on http://rfid.kts-systeme.de/downloads/.

The antenna (fig. 20) is a HF PCB Antenne (PCBA1356_8) also from the company KTS Systeme. It has a dimension of 80 x 80 mm. The connection to the reader is realized by a SMA jack and has a self-impedance of 50Ω. The antenna is designed for passive tags in a frequency range around 13.56 MHz and has a maximum power of 1W.

Figure 20: RFID Antenna KTS Systeme PCBA1356_8

The antenna and the reader are connected with a SMA to U.FL. adapter cable.

5.2 RFID tag

The tag used in the prototype is of paper type (see fig.21) due to its added advantages as follows:

- cheap, with the cost of 18 cents per tag
- doesn't require power supply
- compact
- easier implementation in the time of plant extension.

It's working principle is based on inductive coupling with an operating frequency of 125-135 kHz and a range of 10cm. The tags are fixed on the floor at known location as shown in fig.22.

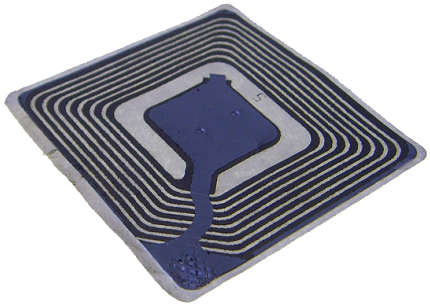


Figure 21: Paper Tag

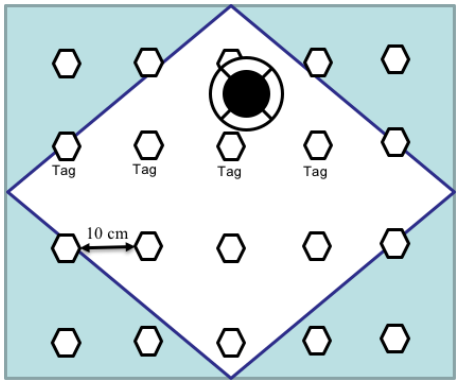


Figure 22: Tags on the plant floor

5.3 Wifi Module

The WiFi module from WEMOS Co.[11] is a mini WiFi board with 4MB flash based on ESP-8266 which is a WiFi microchip with full IP/TCP stack and micro-controller (see fig.23)

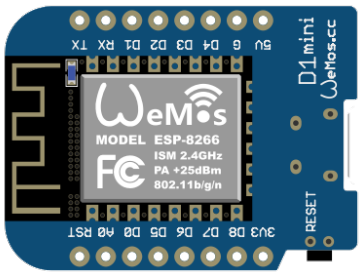


Figure 23: WeMos D1 Mini WiFi Module

- Features
  - 32-bit RISC microprocessor core running at 80 MHz
  - External QSPI flash of 4 MB
  - IEEE 802.11 b/g/n Wi-Fi
  - 16 GPIO pins
  - UART on dedicated pins, plus a transmit-only UART can be enabled on GPIO2
  - 10-bit ADC and I^2C (software implementation)

5.4 Hardware Setup

The in built Micro-controller on the robot receives the sensors data and sends commands to the actuators via serial communication using its first UART pins. (TX/RX) is the process of sending

and propagating an analogue or digital information signal over a physical point-to-point wired connection. It uses its second UART pins to communicate with the built-in WiFi Module. The built-in WiFi Module sends the received sensors data to the Computer and sends the received commands from the computer to the robot micro-controller via Wireless communication (see fig.24). Due to its less complexity, more flexibility and that the robot's in built micro-controller

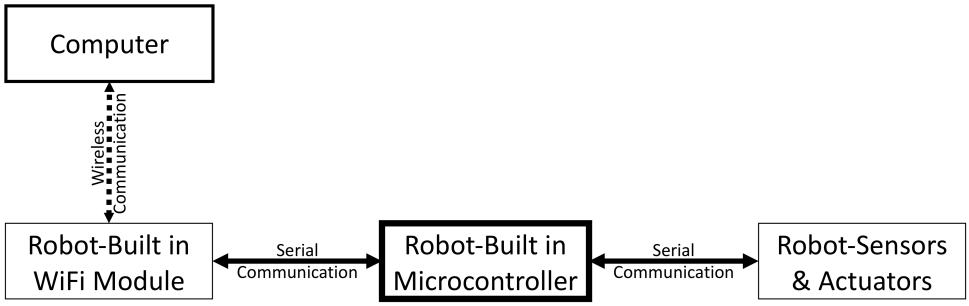


Figure 24: Robot Hardware Schematic

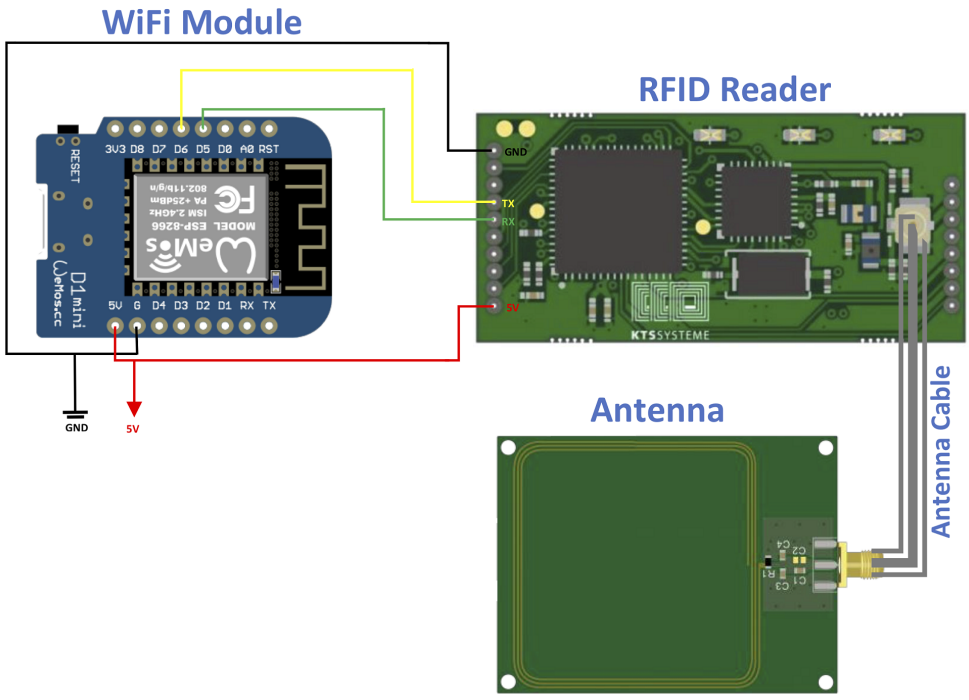


Figure 25: Hardware Schematic

UART pins are in use, a new communication setup were developed for the RFID reader to send the data from the robot to the computer in parallel to the robot hardware setup. As shown in

fig.25) the RFID reader is connected directly to the built-in micro-controller of the WeMos WiFi Module sending it the tags IDs read via serial communication. The WeMos WiFi Module sends the received data through the network. The RFID reader is connected to the antenna which sends and receives the radio waves via SMA antenna cable.
The RFID reader as well as WeMos Module are placed on the top of the robot while the Antenna is fixed to the robot base such that the radio waves would be in direct contact with the tags on the floor.

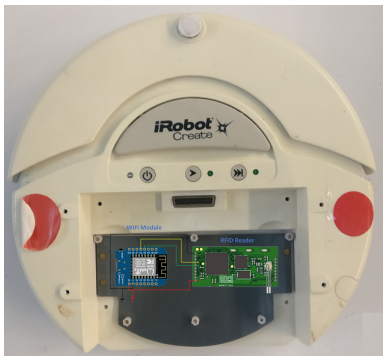


Figure 26: Top of the Robot

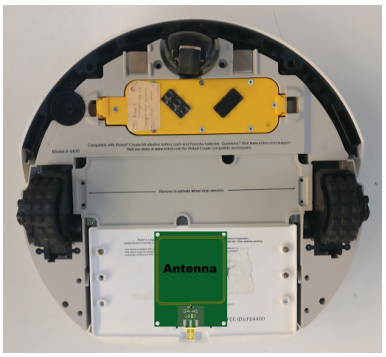


Figure 27: Base of the Robot

6 Simulation[5]

The simulation was carried out to answer important design questions before the real implementation phase. Furthermore artificial RFID reader data was created to test and simulate the algorithm, which will be explained in chapter 7.
To answer the design questions, the simulation has the following parameter (Appendix 11.1 Line 1-50):
- the size of the simulation space
- distance between the tags
- distance between the first/last row/column of tags and the boarder of the simulation space
- diameter of the robot
- position of the antenna related to the origin of the robot
- the relation between RSSI and the distance antenna and tag
- initial start position and orientation
- difference between the measurement points of the initialization procedure
- optional: cycle time and speed of the robot (for another procedure)
- logging parameter (look of the logged text file)

Foregone tests lead to a distance between the tags of 10 cm. This was founded on the fact that in this case at least 4 tags are detected at the same time (maximum reading range of 14 cm). In this case around 121 tags are needed for every square meter, which turned out to be realistic number for a small plant size.

6.1 Emulator

To create artificial RFID reader data, the emulator is able to write all detected tags together with information about the measuring point into a text file. During the initialization procedure, which was the main focus in this project, the robot turns around 360° and makes measurements every 45°.
The emulator computes the distance from the center of the antenna to the neighbouring tags at each measurement point. If a tag is closer than the maximal reading distance, the emulator writes the detected ID of the tag together with its RSSI into the text file.
The RSSI is, as expalined earlier, an integer value from 0...7. 0 defines in this case a distance from 14 to around 10 cm from the antenna to the tag. In the first version of the emulator the RSSI is based on the information from the paper [12] and mentioned a consistent increasing of the RSSI while the distance between the tags and the antenna gets smaller.
During own measurements it has been found out that this relation is inconsistent. Therefore the second version of the emulator was updated and creates more realistic data.

[5] Stephan

6.2 RSSI Measurements with real hardware

The relation of the RSSI is not just related to the distance between the antenna and the tag. It also depends on the orientation of the plain of both components. The tests with the real hardware was performed in a setup where the tags were placed on a floor and the antenna was parallel to the floor at a hight of 1.5 cm. The reason for this was the fact that the antenna should be placed directly under the robot. Tbl. 8 and fig. 28 present the results of the measurements.

RSSI (Received Signal Strength Indicator)	0/0	1/1	2/2	3/3	4/4	5/5	6/6	7/7
Maximal distance antenna to tag [cm]	14	9.8	9	8	7	6	3.5	2.8
Middle distance antenna to tag [cm]	5	5.1	5.3	5.5	5.8	4	-	-
Minimal distance antenna to tag [cm]	-	4.7	4.5	4.3	4.2	-	-	-

Table 8: Relation between RSSI and distance antenna to tag (data)

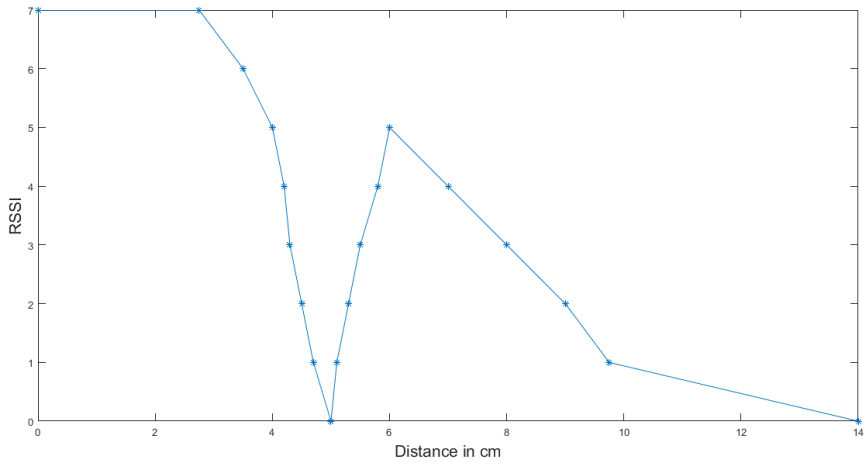


Figure 28: Relation between RSSI and distance antenna to tag

It can be seen that there exists a blind spot at a distance of 5 cm where the RSSI drops to 0. The consequence is that it is not trivial to build up a relation from the RSSI back to the correct distance.

6.3 Simulation with emulated data

The idea of the final implementation is to estimate the initial position and orientation of the robot. A first version of an algorithm to solve this problem is created in matlab. The first part of these algorithm is the emulator which simulates the 360° turn and recordes the tag information. The second part is the solver which is also explained deeper in the chapter 7.
After observing an inconsistent behaviour of the RSSI the simulation as well as the solver were updated.

6.4 Results

The application of the emulated data on the solver indicates the following results:

	Avg. accuracy position (x-, & y-direction) [mm]	Avg. Accuracy orientation [°]
Data mentioned in paper	2	<1
Own recorded data (blind spot)	10	20

Table 9: Results Simulation

As can be seen from tbl. 9, there is a sufficient good match between the estimated position and orientation of the robot for the consistent RSSI data. On the other hand the inconsistent RSSI data results in significant differences in the estimation of the position and orientation of the robot.
The reason for this is the higher complexity of the algorithm to first estimate the correct distances related to RSSI values and then start to estimate the position based on those distances.
A small error in the estimation of the position of the antenna at the first measurement point leads also to a big error in the computed orientation of the robot.

7 Implementation

7.1 Communication

The already existing software is overwritten on the built-in micro-controller of the WeMos Wifi Module which was developed to have a continuous listening to all the data sent from the RFID reader even in case of no tags within the range. The good and bad RFID data readings are as seen in the fig.29.

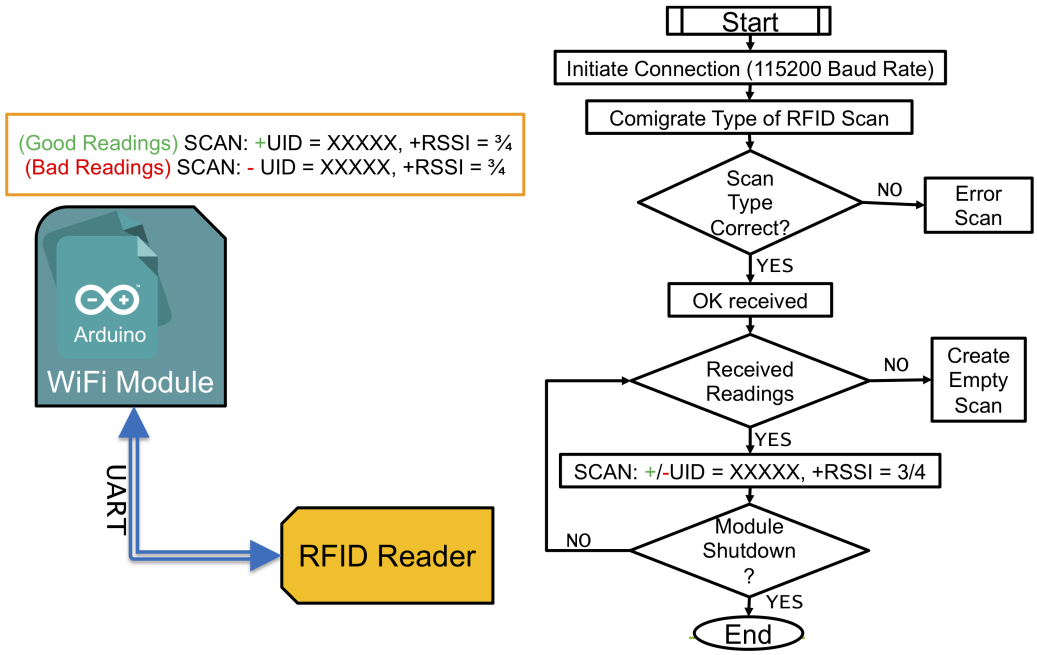


Figure 29: RFID-WeMos Module Communication

A TCP-IP communication is established within the network on the WeMos WiFi Module that start publishing the data which has been received from the reader. This communication is killed if and only if in the case of robot shutdown as shown in fig.30.

While on the other side on computer GUI (Graphical User Interface), the same type of communication is established (which should be in same network), and should grasp all the data that has been published by the WeMos WiFi module even if it no data is sent. This communication can be stopped from the GUI if needed as shown in fig.31.

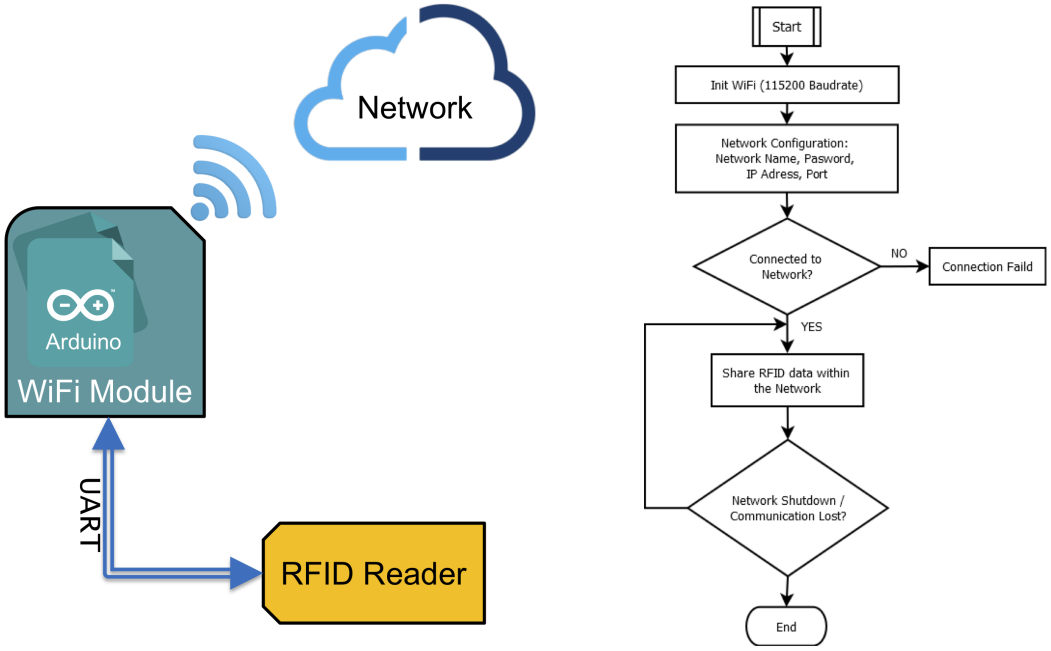


Figure 30: WeMos Module Network Communication

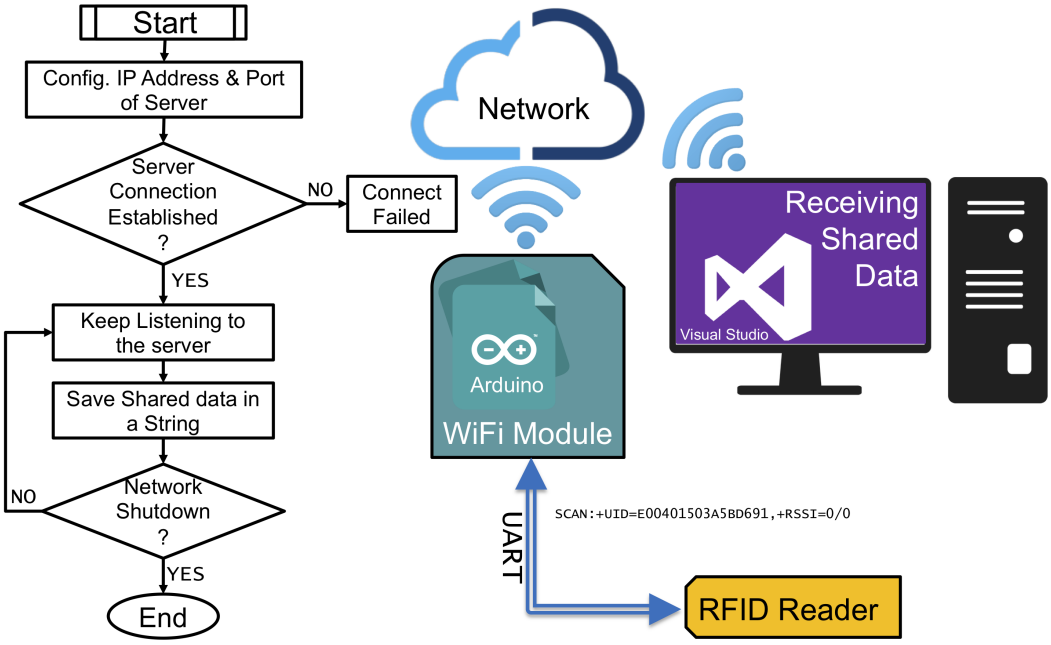


Figure 31: Computer Network Communication

7.2 Initialization procedure

In the start-up phase, before running the pipeless plant with its AGVs the correct position and orientation of each and every vehicle are not known. Even though the controller is able to compute the position of the AGVs antenna in each point of time (t=0 included), several AGV positions in the plants operation space can be discribed by one single antenna position. In fig. 32 four possible AGV positions with one common antenna position are pointed out.

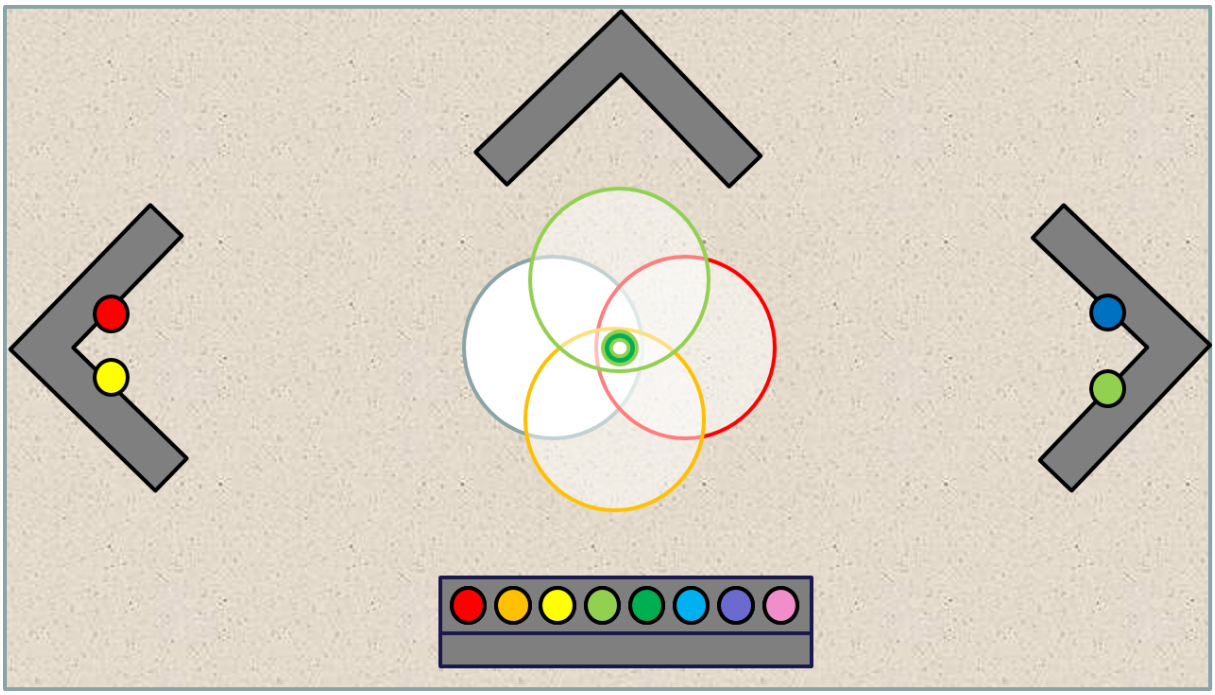


Figure 32: Different possible positions for one antenna position

Since the position information is crucial for the plant, a procedure was set up to determine the starting positions of each and every AGV. According to the fact, that the position and orientation of a single AGV is unknown at the beginning, some potential are taken into account. For instance, the plant contains several obstacles like the mixing stations, vessel storage, charging stations, plant edges and even other vehicles as represented in fig. 33.

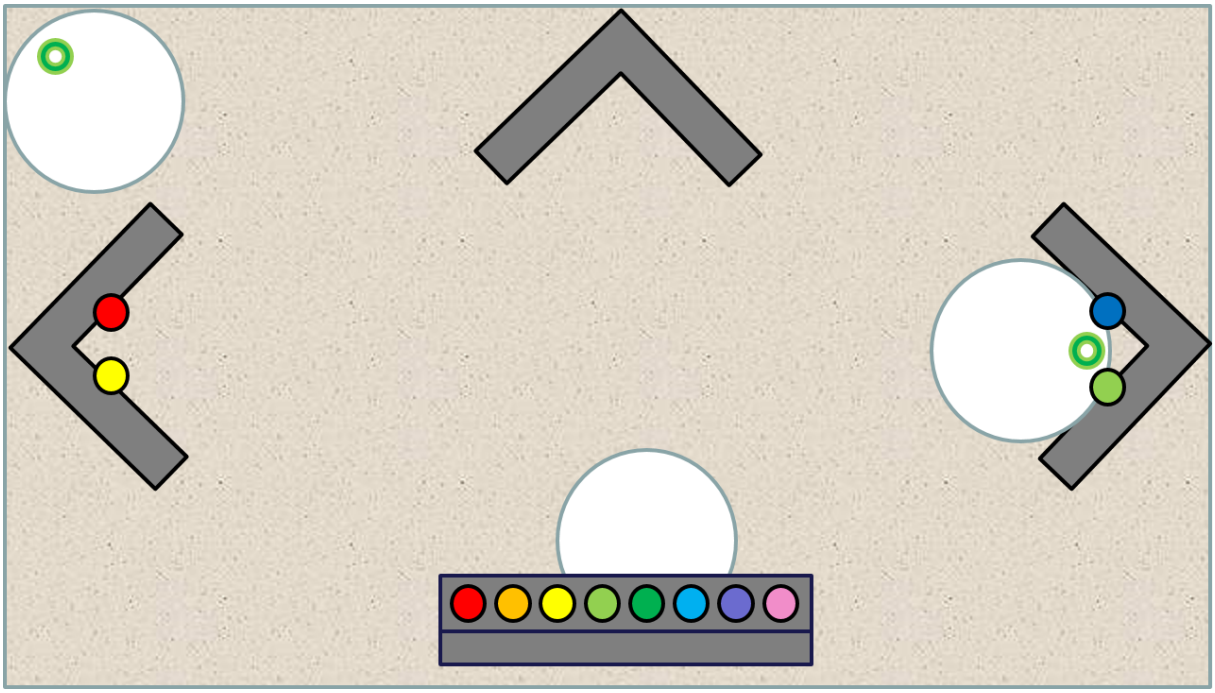


Figure 33: Possible hazards/obstacles

With respect to these potential hazards collisions during the initialization procedure have to be avoided. This is realized by taking advantage out of the AGVs ability to turn around its z axis without a change of the AGVs center point in x and y direction. This ability of the AGV leads the way that each and every robot performs an initialization turn of 360° in which measurements are taken every 45° to estimate the specific positions and orientations of the AGVs. Furthermore, the decision process of the antenna position under the robot is dominated by the fact that the position of the center point does not change during a turn around its z axis. During the 360° turn the intervals in which the measurments have to be taken need to be known by the controller. The determination of these measurement points can be computed in two different ways. On the one hand the encoders of the AGV-wheels can be used to estimate the performed rotation. On the other hand, the time of a complete turn can be measured and used as a parameter in the procedure. In terms of simplicity the second option is used in the initialization procedure. Fig. 34 illustrates a sequential flow chart which describes the movement and data processing during the initialization procedure. The part of the code which is explained and visualized by Fig. 34 is found in section 11.4

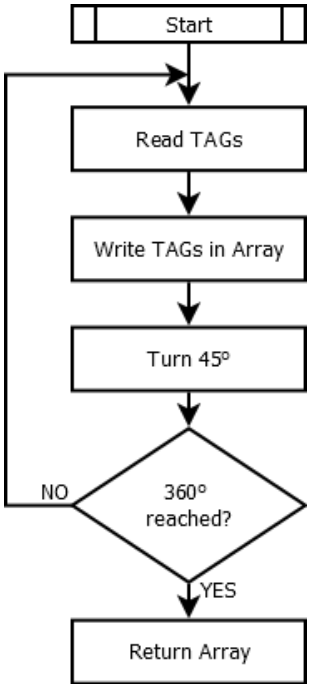


Figure 34: Flow Chart: Initial procedure 360° turn

The initialization procedure for AGV No. 1 is created and can be started in the GUI in the test environment. In the first place an integer number is given to the field called Sleeptime. This integer number is interpretated in milliseconds and describes the time of rotation. Even though a time for a complete turn of 45° has been found at around 1125ms it has to be said that this time strongly depends on the battery charge of the AGV. After the desired turning time is given to the GUI the initialization is started by pushing the button Initialization, located over the input box in fig. 35.

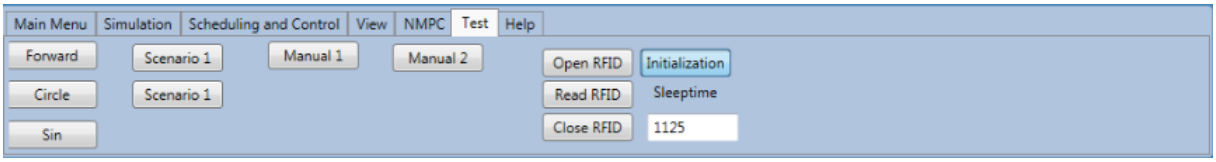


Figure 35: Test environment in GUI

In the second step, after the procedure was started, all the available IDs and their respective

RSSI in the current reading range of the RFID-Reader are read. The reading is performed in the Automatic Scan mode of the RFID reader[10]. With the included timestamp for every measurement a delay of minimum 30ms between each tag information was detected. With respect to this delay the antenna has to stop a specific period of time at each measuring point to deliver correct data of all the reachable tags. Experiences have shown, that a measuring time between one and two seconds delivers the best results. During this time every 100 ms new measurment information are taken. In order to save the single TAG information of each and every measuring point an initially empty array with 14 columns and 8 rows is created. The number of rows is derived by the fact that measurements are taken at every 45°.

$$Rows = 360°/45° = 8 \tag{12}$$

$$\tag{13}$$

The first seven columns in the array are filled by the received TAG IDs and the last seven entries are filed by the respective RSSI.
The number of columns is derived by the fact that at each and every measurement point in the used test environment, information of maximal seven TAGs can be read.

$$Columns = max.no.of tags * 2 = 7 * 2 = 14 \tag{14}$$

$$\tag{15}$$

Once the received data is saved in its corresponding row, the AGV turns around 45°to place the antenna at the next measuring point. An AGV turn is realized by setting the velocity of the right and left wheel in different directions. During the turning sections the velocity is set to 100 mm/s or rather -100 mm/s. This procedure of reading information, writing information in the initialization array and turning 45° to the next measuring point is repeating itself until a 360° turn is performed. After a successful initialization turn the corresponding array of measurement information can look like the example in table 10. The code which realizes the filling of the array can be seen in section 11.5

4	1	5	2	3			0	0	1	7	0		
5	3						2	3					
3	5						2	2					
9	8	6	5				1	1	1	2			
9	7	8	6	4	5		2	0	6	0	0	2	
4	7	5	8				0	2	3	3			
5	4	7	8	1			2	5	0	0	0		
2	4	1	5				0	2	2	0			

Table 10: Filled array after 360° turn

7.2.1 Recording and filtering data

To read the ID and RSSI of all the TAG laying in the reading range the RFID-reader is set to its Automatic mode and its Anticollision is switched on. In this mode packages of strings with a length of 35 characters are received by the plans computer. Even though these 36 character strings contain all the information of the TAG which is needed some effort has to be taken to seperate the useful parts which are processed in the localization algorithm.
With exception of the information each string contains, the structure itself is always the same. In the first five characters the substring "*SCAN:*" is detected and deleted for the further process. The first important character is found in the sixth slot of the string. Here either a "*+*" or a "-" is written. With the help of this sixth slot it is distinglished wheather the current reading is complete or not. In order to guarantee the correctness of the received information the measurments are filtered by the "*+*" and the measurments in which a "-" is included are ignored in the further processes. After the indicator for complete and in complete readings a introduction to the ID is indicated by "*UID=* " and cut out of the string. The next 16 characters defines the unique identification of the specific TAG. As a last useless string, which has to be cut out, with the structure "*.RSSI=*" is found directly after the ID. As a result the 16 character hexadecimal ID and its respective RSSI are seperated from the received string. Since the ordered tag IDs differ each other just in the last three numbers these numbers are transformed in a decimal number before UID and RSSI are used for further computations. The code which realizes the recording and filtering of the data can be seen in section 11.2

String Transformation	
Complete	**Incomplete**
SCAN:+UID=E00401503A5BD691,+RSSI=0/0	SCAN:-UID=E00401503A5BDAE4
UID=E00401503A5BD691,+RSSI=0/0	
E00401503A5BD691 0/0	
1681 0	

Table 11: String preperation

7.2.2 Analysing data

In the next step of the algorithm the previous described filled array is analyzed. To estimate the position and orientation of the AGV the array has to include two valid sets of each two valid measuring points. During this analyzation the single measurement point-sets are validated in terms of following restrictions:

1.: At the two valid measurement point each contains at least three tags
2.: The other measurement point in one set needs to have a distance of 180° to the first.

In terms to get the adequate sets of measurment points the array is analyzed row by row. The stepwise workflow is vizialized in fig. 36.
The code in which the analization is realized can be seen in section 11.6

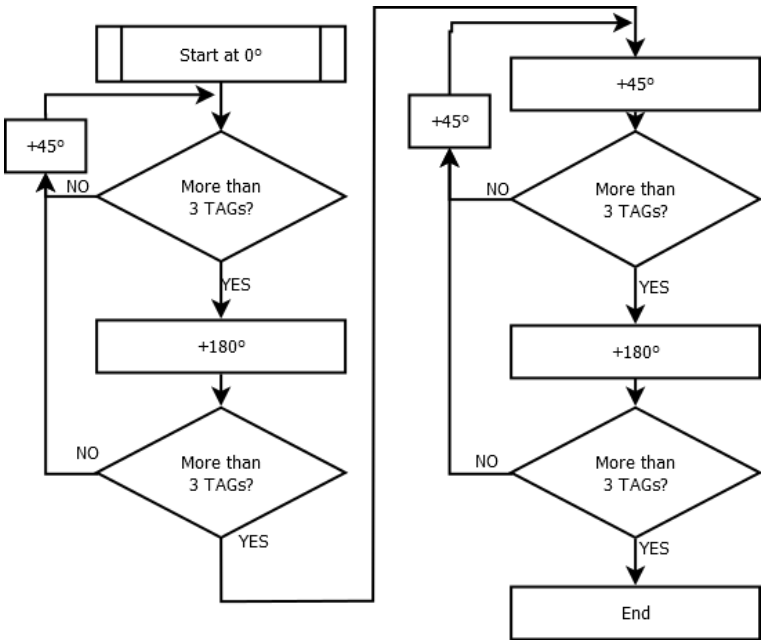


Figure 36: Flow Chart: Analizing measurment points

Initially the first row which represents the measurement at the point 0° is checked in terms of the number of readable tags. If this specific number is higher or equal three the transition is acknowledged as true and the same query will be performed at the measurement point with a distance of 180° to the former measurement point. If this next measurement point can be described as valid, the first valid set of two measurement points is found. If, on the other hand, the number of readable TAGs are less than three, which means that the triangulation algorithm cannot be performed, the current measurement point is ignored and the next measurement point is evaluated. Each of this sets of two measurement points is saved as a 1x2 array called Solution 1 and Solution 2 is used for the estimation of the position of the measurement points which is explained in the section 7.2.4 Estimation of initial position and orientation.

7.2.3 Selection of correct distance related to RSSI [6]

In a first step the multiple occurring data points (see tbl.8) are divided into three groups (max, middle and min) where max means the maximal possible distance related to one RSSI and so on.

[6] Stephan

The measurements have shown that it is not trivial to define the correct distance related to most of the RSSI. The involved algorithm selects the correct distance out of the multiple possible solutions and is shown in fig. 37:

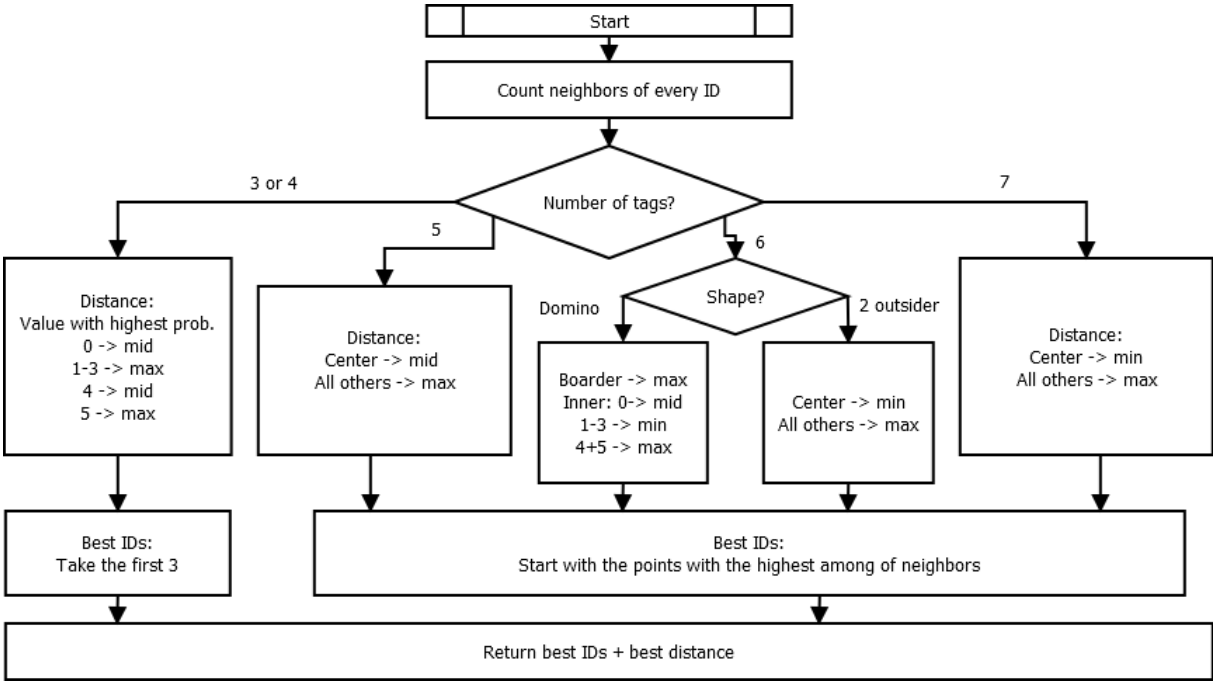


Figure 37: Flow Chart: Selection of correct distance and most proper IDs

To distinguish between the multiple possible solution for one RSSI, the algorithm defines the shape of the pattern of tags based on the number of tags at each measurement and the number of the neighbours each tag has. At each measurement point in this scenario several numbers (4-7) of detected tags are possible. The different shapes can be found in the tbl. 12.

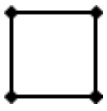
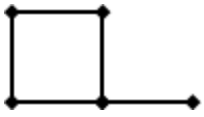
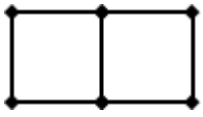
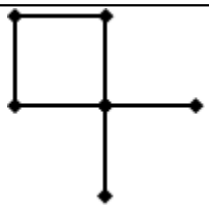
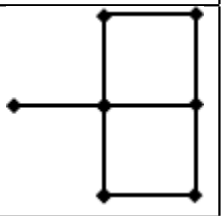
Number of detected tags	4	5	6 (Domino)	6 (2 alone)	7
Unique shapes					

Table 12: Possible shapes of pattern

Going back to the flow chart fig.37 the first step is to count the number of neighbours each tag has. With this information, the position of the tag in the pattern can be detected. For example, a tag with 3 neighbours in a pattern of 5 tags, is the center of this pattern.
After the number of tags at each measurement point and the position of each tag are defined, the selection of the correct distance will be performed based on the highest probability. To know the highest probabilities an analysis of measurements with emulated data has been done.
As an example 4 detected tags are leading to the fact that the position of the antenna should be very close to the center of this square. If in this case a RSSI of 4 is detected, the middle value (5.8 cm) will be taken.
Afterwards the most suitable three IDs will be selected, in case where more then three are detected. The algorithm takes at first the ID with the highest amount of neighbours, because these tags are close to the position of the antenna and have probably a value of 6 or 7 and are uniquely defined. In the case where several tags with the same number of neighbours, the first ID (number increasing) will be taken.
The return of the function is an array (2x3) with the indices of the chosen IDs and the correct distance. The correct distance will be indicated by the number 0,1 and 2. 0 means the maximal, 1 the middle and 2 the minimum possible value related to one RSSI. For example

$$\begin{bmatrix} 3 & 2 & 4 \\ 2 & 0 & 0 \end{bmatrix}$$

leads to the choice of the maximal value of the RSSI of the fourth detected ID and the minimum value of the RSSI of the third and the fifth ID in the recorded array at this measurement point.

7.2.4 Estimation of initial position and orientation [7]

As mentioned in chapter **??**, the main idea to estimate the initial position is to find the intersection point, which lies in the middle of the measurement points.
To compute this position, the algorithm uses trilateration at every suitable measurement point to estimate its position. For trilatertion are three defined positions plus three radii necessary, which are available after the selection of the correct distance and proper IDs.

As follows from the fig.38 shown above, the intersection point is found by computing two linear functions which go trough two corresponding points (blue lines). The center of the robot is then the intersection of those two linear functions and can be computed by the following equations:

$$x = \frac{(x_1y_2 - y_1x_2)(x_3 - x_4) - (x_1 - x_2)(x_3y_4 - y_3x_4)}{(x_1 - x_2)(y_3 - y_4) - (y_1 - y_2)(x_3 - x_4)} \tag{16}$$

$$y = \frac{(x_1y_2 - y_1x_2)(y_3 - y_4) - (y_1 - y_2)(x_3y_4 - y_3x_4)}{(x_1 - x_2)(y_3 - y_4) - (y_1 - y_2)(x_3 - x_4)} \tag{17}$$

Theoreticaly all eight measuring points are suitable points (at least four IDs found). But for the case that the real measurements differ from the theory, the algorithm just needs four suitable

[7] Stephan

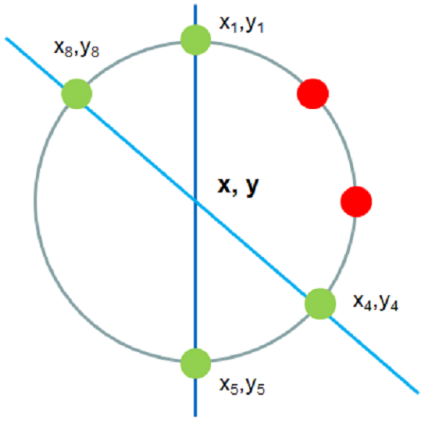


Figure 38: Computing the center of the robot

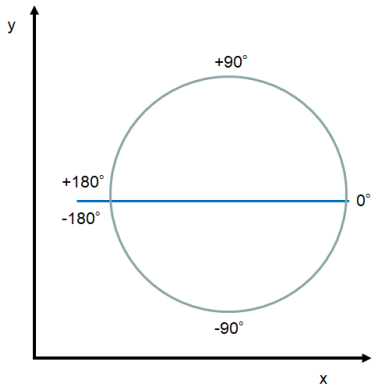


Figure 39: Orientation of robot in absolute angle

points.
After the initial position as well as the positions of 4 measurement points are known, the algorithm computes the orientation based on those information. The relative angle between the center and the first measurement point will be computed with the arctan2 function and leads to an orientation -180° $< \Theta \leq$180° as shown in fig.39.
To compute the absolute angle, the angle of the measurement point has to be subtracted and 180° has to be added. This is caused by the fact that the antenna is placed on the back of the robot and the absolute orientation should be the direction of the front. After this computation, the initial position and orientation of the robot are known.

7.3 Test setup[8]

In order to verify the validity of the initialization procedure, experiments with the components mentioned in chapter 5 were carried out. The beginning of these experiments were the reconstruction of one of the AGVs with this HW setup. After all components were added to the AGV the power supply was realized via a powerbank and the USB connection of then wifi modul. The plan is to replace this in the future with a direct connection to the battery of the AGV. Fig.40 gives an overview of the test setup and shows that also for the prototype, the reader and the wifi modul was just stuck with Sellotape on the upper layer of the AGV.

The test platform was a field of 9 tags which were stuck on a piece of carton. The IDs and its positions are shown in tbl.13.

The reason for the small setup was the fact that until the end of the project only 10 tags were available. One of the following steps should be to extend the platform with more tags.

[8]Stephan

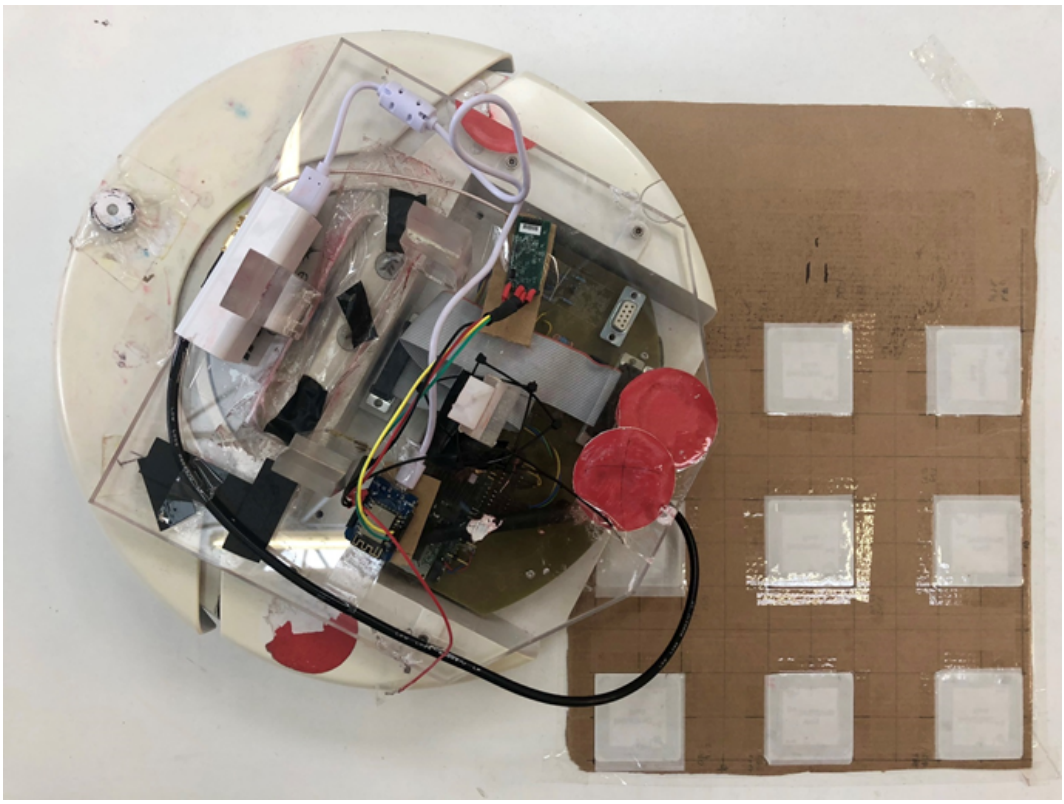


Figure 40: testing setup for initialization procedure

X-dir. [mm]	0	100	200	0	100	200	0	100	200
Y-dir. [mm]	0	0	0	100	100	100	200	200	200
ID tag [hex]	AE4	689	47A	586	785	ADC	BF4	691	78D
ID tag [dec]	2788	1673	1146	1414	1925	2780	3060	1681	1933

Table 13: Positions of the IDs in the test setup

The initialization procedure was started via the GUI. A time value was added in the GUI to perform the 45° turns. This number was around 1125 ms and is highly correlated to the battery status of the AGV.

7.4 Results[9]

A couple of tests on the test setup (previous section) were performed to compare the good results created with the simulated data with real measurements. The result of the position estimation was directly plotted in the console. The initial position was 200 mm in x- and y-direction and a varying orientation (0°, 90°, 180° and -90°). Fig.41 and fig.42 illustrate the actual measurement results and the desired position in x- and y-direction.

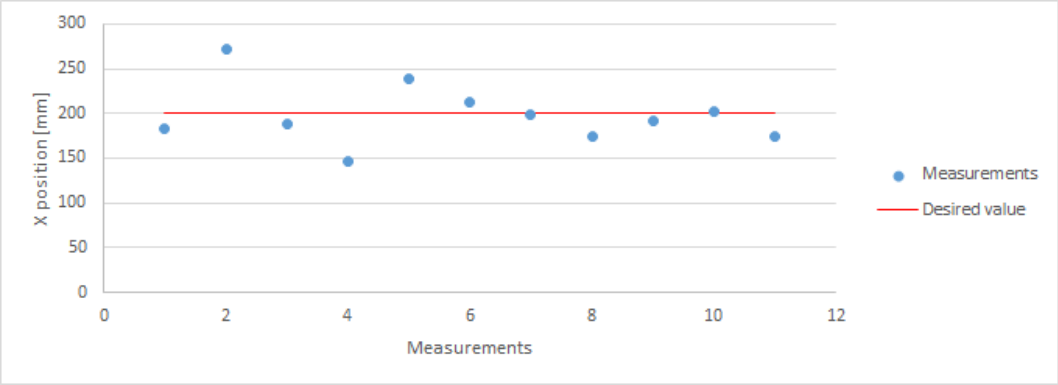


Figure 41: Estimated position in x-direction

The average of the absolute error of the position in x-direction was 24.5 mm. The minimum and maximum error were 2 mm and 72 mm.

The average of the absolute error of the estimation of the position in y-direction is with 23.3 mm, a minimum error of 3 mm and an maximum error of 77 mm very similar to the results from the estimation of the x-direction. The computation of the overall error of the position has an average derivation of 37.5 mm and a minimum and maximum error of 6.3 mm and 77 mm.

For the estimation of the orientation, the average of the absolute error was 23° with a minimum and a maximum value of 3.9° and 37.5°. The measurements also shows that an estimation of the position with a big error not necessarily leads to a big error in the estimation of the orientation (see measurement 4 in fig.41, 42 and 43).
An extension of the results could also be an analyse of the estimated positions of the antenna at the measurement points. Those points were also plotted in the console.

[9]Stephan

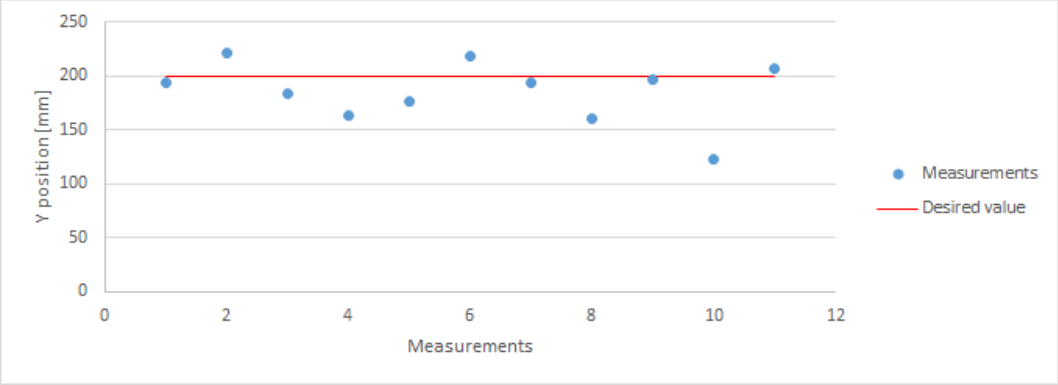


Figure 42: Estimated position in y-direction

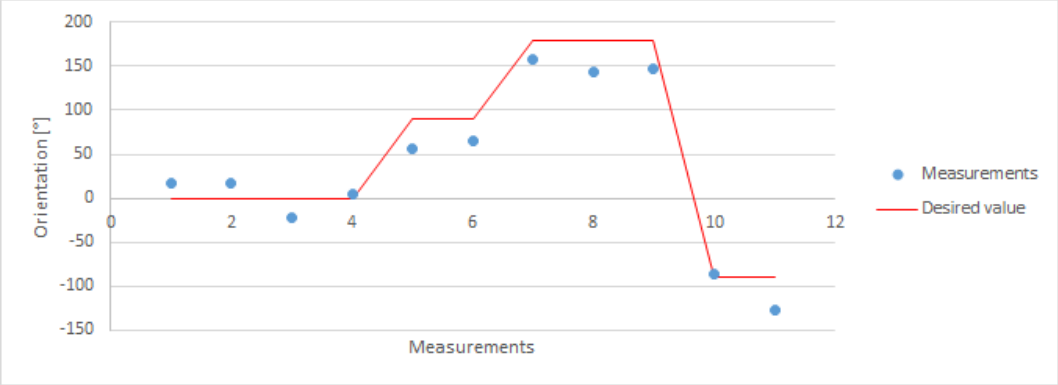


Figure 43: Estimated orientation

8 Conclusion

The developed localization solution was for the pipeless plant, a prototype of a chemical production plant which has a size of 3 by 4 meter. In this plant the vessel will be transported by AGVs from one station to another. In the actual setup only a camera, which is installed above the plant, was used to detected the AGVs and estimate their positions. The problem with this technology is the bad detection of the LED pattern from the AGVs during bright light conditions and also the space limitation. Another big disadvantage was the big computation effort which made the system also very slow. The main task of this project was to find an alternative tracking solution. During the project group phase differnt localization technologies were evaluated. With respect to the outcoming reseaches about triangulation, map-based-localizaion, pattern recognizion and localizaion via radio frequency identification the last RFID based localization of the AGV with passive tags as landmarks turned out to be the most promising among those four. With information of a similar project realized by the FH Dortmund a model to evaluate sample data and a localization algorithm was created in Matlab. This results of the simulation were promising and therefore used during the decision making process about the actual hardware setup. With an demonstration board with the size of 30 cm x 30 cm the initialization procedure algorithm was implemented in which the AGV performes and 360° turn and estimates its position and its orientation based on measuremtns during this movement. With respect to this solutions it can be said that it is possible to assemble a reader on an AGV and detect passive tags with its antenna in a range of 14 cm. It also has been found out that an inconsistent realation between the revieved signal strength (RSSI) of the detected TAGs and the distance based on the RSSI is not generally trivial and was only solved in a rather simple and unriliable way during the project. Based on the results computed by the initializaion prcedure, it can be concluded that it is possible to estimate the position of the AGV with an average accuracy of aroung 2.5 cm and an estimation error of the orientation of around 23°. Compared to the former localization set up this solutions, especially with respect to the orientation error, are not perfelty satisfying and just minimal requirements are fullfilled. The recieved data from the RFID reader have furthermore clearly shown that the anti-collision algorithm used by the reader leads to an unknown amount of time until each and every TAG in the detection area is identified. Summed up a model based demonstrator was realized which on the one hand does not improve the accuracy of the localizaion of the plant under good light conditions especially with respect to the orientation but on the other hand a promising technology for indoor localizaion with light independedcy, respectivelay cheap costs and highly scalability was found.

9 Future Work

After a proof-of-concept for an RFID based localization system has been built and a first demonstration set-up has been built the disadvantages and limitations of the prototype were evaluated. According to these results several points of improvement and extension were found and categorized into a hardware and a software section.

9.1 Hardware

- The AGVs are feed by an included 12V battery which provides the power for all included electronical devices. This 12V power supply is available on board and is suggested to be used. Currently the WiFi-Module and the RFID-Reader are fed by an external powerbank since a 5V power supply is needed. In terms of one zentralized power supply a 12 V to 5 V converter can be installed and connected to the reader and wifi module.

- As a first setup a demonstration area of 3 x 3 tags was build. In this rather small area the initializaion procedure was developed, but a real time localization while a path is followed by an AGV was not possible since the 30cm x 30cm was simply to small. For futrue research in terms of localizaion on a specified path additional TAGs can be included to the area of operation. Since the RFID concept is highly scalable the only change that needs to be made in the algorithm is the insertion of the additional TAG into the lookup table.

- Currently the Robot No. 1 is the only AGV which is equipped with the RFID technology. To run the plant with multible AGVs the remaining robots needs to be upgraded.

9.2 Software

- During the initalizatin procedure a 360° turn is performed. The desired turn around 45° is realized by a driving time of 1125 ms. But it needs to be said that this movement is highly dependend on disturbances like changing battery charge and plant underground. For the future developers it is suggested to use the encoders of the robot wheels as a determination of the orientation instead of the parameter time.

- As an alternative localization technology was found several code lines in the current code can be deleted since the camera and image processing is simply not used anymore. With a clean code an improvement of processing time will be achieved.

- As a last point it can be said that even though a localization with RFID is now possible the results are not 100 percent realiable and the accuracy especially with respect to the orientation is not satisfying so far. As an improvement the triangulation algorithm has to be optimized and or a second RFID-antenna has to be added under the AGV to reduce measurment errors.

10 References

References

[1] Yihuan Zhang Liang Wang and Jun Wang. Map-based localization method for autonomous vehicles using 3d-lidar. *IFAC (International Federation of Automatic Control) Hosting by Elsevie*, 50, 2017.

[2] Dr.-Ing. habil. Dipl.-Ing. Dipl.-Ing. Joerg Wollnack. Prinzip der dreidimensional messenden videometrischen messsysteme.

[3] Jeremie Houssineau, Daniel Clark, Spela Ivekovic, Chee Sing Lee and Jose Franco. A unified approach for multi-object triangulation, tracking and camera calibration.

[4] Matteo Munaro, Edmond Wai Yan So, Stefano Tonello, and Emanuele Menegatti. Efficient completeness inspection using real-time 3d color reconstruction with a dual-laser triangulation system. 48:201–225, 09 2015.

[5] Yuntian Brian Bai, Suqin Wu, Hongren Wu, and Kefei Zhang. Overview of rfid-based indoor positioning technology, 2012.

[6] Jun Liu Baoxian Zhang and Haoyao Chen. Amcl based map fusion for multi-robot slam with heterogenous. *IEEE International Conference on Information and Automation (ICIA)*, 6, 2013.

[7] ROS. Amcl, 2018.

[8] Hanifa SHAH Kamran AHSAN and Paul KINGSTON. Rfid applications: An introductory and exploratory study. *International Journal of Computer Science Issues*, 7(3), 2010.

[9] Vinita Sharma Ms.Neha Kamda and Sudhanshu Nayak. A survey paper on rfid technology, its applications and classification of security/privacy attacks and solutions. *IRACST - International Journal of Computer Science and Information Technology & Security*, 6(4), 2016.

[10] Pablo Cotera, Miguel Velazquez, David Cruz, Luis Medina, and Manuel Bandala. Indoor robot positioning using an enhanced trilateration algorithm. *International Journal of Advanced Robotic Systems*, 13(3):110, 2016.

[11] KTS Systeme. Rfid plug module rfidm1356, 2017.

[12] WEMOS. Wemos d1 mini, 2018.

[13] Christof Rohrig, Daniel Hess, and Frank Kunemund. Rfid-based localization of mobile robots rfid-based localization of mobile robots using the received signal strength indicator of detected tags.

11 Appendixes

11.1 Appendix A: Emulator RFID data (Matlab)

```
%% ------------------------------------------------------------------------
% Description:  Emulator, which creates txt file like the reader
%               RSSI related to the real measurements
%               For the Initialization procedure, turn around 360°
% Date:         12.06.2018
% Created by:   Stephan Vette
% ------------------------------------------------------------------------
%% RFID signal emulator
clear all
clc
close all
% Initializing
l1 = 100;   % length of the plant, x  [cm]
l2 = l1;   % width of the plant, y   [cm]
d1 = 10;    % distance between tags     [cm]
d2 = 0;     % distance last tag <-> boarder [cm]
r1 = 14;    % radius of the reading range of every tag
r2 = [r1, 9.75, 9.0, 8.0, 7.0, 6.0, 5.8, 5.5, 5.3, 5.1, 5.0, 4.7, 4.5, 4.3, 4.2, 4.0,
    3.5, 2.75, 0]; % distances at certain RSSI
r4 = [0, 1, 2, 3, 4, 5, 4, 3, 2, 1, 0, 1, 2, 3, 4, 5, 6, 7, 7]; % array with the
    different RSSI values

r3 = 33/2;  % radius of the robot
d3 = 10;    % distance between origin robot and origin antenna [cm]

angle1 = 45; % angle between the measurement points in the init procedure

gamma1 = deg2rad(22.5);  % Start orientation of robot [rad]
robStart = [22.5, 51.5]; % Start position of robot in x, y [cm]

robSpeed = 0.1;       % Speed robot [m/s]
cycleT = 100;        % Cycletime in [ms]

mode = 1;   % mode=1: tracking all available tags, which are nonzero
            % mode=2: tracking only changes in the RSSI signals
mode_hex = 0;   % activate or deactivate hex ID

% For the name of the txt file
measuementeNumber = num2str(11); % Number of measurement
% Two possibilities for the content of the txt file
% 1. Without filtering. Exactly like the reader creates data
% text0 = '<\r>';
% text1 = 'OK';
% text2 = 'SCAN:+UID=';
% text3 = '+RSSI=';

% 2. Filtered data. Without unusable information.
text0 = ' ';
text1 = ' ';
text2 = ' ';
text3 = ' ';
%% Error check
if mod(l1/d1,1)~=0
    error('Length of platform not dividable by distance between tags');
elseif mod(l2/d1,1)~=0
```

```
54      error('Length of platform not dividable by distance between tags');
55  end
56
57  %% Computing position of antenna
58  numTagsX = (l1-2*d2)/d1 +1;
59  numTagsY = (l2-2*d2)/d1 +1;
60  numTags = numTagsX * numTagsY;
61  antPos = robStart + d3 * [cos(gamma1), sin(gamma1)];
62
63  %% Display the setup, write important information into a seperate txt file
64  d1_str = num2str(d1);
65  l1_str = num2str(l1);
66  l2_str = num2str(l2);
67  numTags_str = num2str(numTags);
68
69  msg0 = ['Your plane is ',l1_str,'cm x ',l2_str,'cm.'];
70  msg1 = ['You chose a distance of ',d1_str, 'cm and need ', numTags_str,' Tags!'];
71  disp(msg0);
72  disp(msg1);
73  nameTxt = ['NumTags',measuementeNumber,'.txt'];
74  fileNumTags = fopen(nameTxt,'w');
75  fprintf(fileNumTags,'%6d\n',numTags);   % Write the number of tags in file
76  fprintf(fileNumTags,'%6d\n',l1);        % Write the size of the plant in file
77  fprintf(fileNumTags,'%6.4f\n',gamma1);      % Write the starting angle
78  fprintf(fileNumTags,'%6d\n',robStart(1));      % Write the starting pos
79  fprintf(fileNumTags,'%6d\n',robStart(2));      % Write the starting pos
80  fclose(fileNumTags);
81
82
83  %% Drawing environment
84  figure(1)
85  x1 = [0 l1 l1 0 0];
86  y1 = [0 0 l2 l2 0];
87  plot(x1, y1,'LineWidth',2)
88  xlim([-5 (l1+5)]);
89  ylim([-5 (l2+5)]);
90  hold on
91
92  % Position of the tags
93  ID = 1:numTags;
94  [Tagx,Tagy] = meshgrid(d2:d1:l1-d2,d2:d1:l2-d2);
95  plot(Tagx,Tagy,'r*')
96  % Circles
97  radiipl = ones(numTagsX,1)*r1;
98  for k=1:numTagsX
99      tempx = Tagx(1:end,k);
100     tempy = Tagy(1:end,k);
101     temppos = horzcat(tempx,tempy);
102     viscircles(temppos,radiipl,'Color','k','LineStyle',':','LineWidth',0.25);
103 end
104 robX = robStart(1);
105 robY = robStart(2);
106 plot(robX,robY,'bO','LineWidth',3);
107 plot(robX,robY,'r:');
108 viscircles([robX,robY],r3,'Color','k','LineWidth',0.25);
109 plot(antPos(1),antPos(2),'bs');
110 xlabel('Length platform in cm')
111 ylabel('Width platform in cm')
112 title({'Position and reading range of tags';'Start-, endpoint and path of the robot'});
```

```
113 hold off
114 pause(1)
115
116 %% Animation and loggin
117 xUpdateAnt = antPos(1);
118 yUpdateAnt = antPos(2);
119 deltaR = deg2rad(angle1);        % A new measurement after every XX°
120 % Txt file name
121 name = ['Meas_StartingProc_like_reader_real_data',measuementeNumber,'.txt'];
122 fileID = fopen(name,'w');
123
124 % Data stored in variables
125 dataRSSI = zeros(8,numTags);
126 streamDataRSSI = zeros(1,numTags);
127 streamDataRSSIold = zeros(1,numTags);
128 timeStep = 1;   % current measurement step
129
130 % antPos = robStart + d3 * [cos(gamma1), sin(gamma1)];
131 figure(2)
132 for l=0:360/angle1
133     deltaR_temp = deltaR * l;
134     xUpdateAnt = robStart(1) + d3 * cos(gamma1 + deltaR_temp);
135     yUpdateAnt = robStart(2) + d3 * sin(gamma1 + deltaR_temp);
136     plot(x1, y1,'LineWidth',2)
137     hold on
138     xlim([-5 (l1+5)]);
139     ylim([-5 (l2+5)]);
140     [Tagx,Tagy] = meshgrid(d2:d1:l1-d2,d2:d1:l2-d2);
141     plot(Tagx,Tagy,'r*')
142     plot(robX,robY,'bO','LineWidth',1);
143     plot(robX,robY,'r:');
144     plot(xUpdateAnt,yUpdateAnt,'bs');
145     xlim([-5 (l1+5)]);
146     ylim([-5 (l2+5)]);
147     viscircles([robX,robY],r3,'Color','b','LineWidth',0.5);
148         for k=1:numTagsX
149             tempx = Tagx(1:end,k);
150             tempy = Tagy(1:end,k);
151             temppos = horzcat(tempx,tempy);
152             viscircles(temppos,radiipl,'Color','k','LineStyle',':','LineWidth',0.25);
153         end
154     hold off
155
156     % Creating measurements
157     antPosnew=[xUpdateAnt,yUpdateAnt];
158     for m = 1:numTags                    % m = current number of tag
159         m_str = num2str(m);
160         tempTag=[Tagx(m),Tagy(m)];
161         tempD = pdist([antPosnew; tempTag] ,'euclidean');
162
163         % Display if tag is in range or not
164         if tempD > r1
165                 streamDataRSSI(m) = 0;
166                 if (streamDataRSSI(m) ~= streamDataRSSIold(m)) && mode == 2
167                     if mode_hex == 1
168                         fprintf(fileID,'%d %s%s%s%d%s\n',l*angle1,text2,dec2hex(m, 16),
                                text3,k(end),text0);
169                     elseif mode_hex == 0
170                         fprintf(fileID,'%d %s%d%s%d%s\n',l*angle1,text2,m,text3,k(end),
```

```
                                text0);
171                         end
172                          fprintf(' %d %d %1d\n',l*angle1,m,'0');
173                     end
174             elseif tempD <= r1
175                     % disp(['Label ',m_str,' in range!!!!!!!!!!!!!']);
176                     % Relation distance <-> RSSI
177                     k_temp = find(r2>=tempD);
178                     k = r4(k_temp);
179                     dataRSSI(timeStep,m) = k(end);
180                     streamDataRSSI(m) = k(end);
181                     if (streamDataRSSI(m) ~= streamDataRSSIold(m)) && mode == 2
182                         if mode_hex == 1
183                             fprintf(fileID,'%d %s%s%s%d%s\n',l*angle1,text2,dec2hex(m, 16),
                                    text3,k(end),text0);
184                         elseif mode_hex == 0
185                             fprintf(fileID,'%d %s%d%s%d%s\n',l*angle1,text2,m,text3,k(end),
                                    text0);
186                         end
187                         fprintf(' %d %d %8d,\n',l*angle1,m,k(end));
188                     elseif mode == 1
189                         if mode_hex == 1
190                             fprintf(fileID,'%d %s%s%s%d%s\n',l*angle1,text2,dec2hex(m, 16),
                                    text3,k(end),text0);
191                         elseif mode_hex == 0
192                             fprintf(fileID,'%d %s%d%s%d%s\n',l*angle1,text2,m,text3,k(end),
                                    text0);
193                         end
194                         fprintf(' %d %d %1d,\n',l*angle1,m,k(end));
195                     end
196             end
197         end
198         streamDataRSSIold = streamDataRSSI;
199         pause(cycleT/1000)
200         timeStep = timeStep + 1;
201     end
202     savefig('Figure2.fig');
203     fclose(fileID);
204
205     %% Results
206     % figure(3)   % plot for the max value of every tag
207     % dataRSSInoT = reshape(max(dataRSSI),[numTagsX,numTagsY]);
208     % plot3(Tagx,Tagy,dataRSSInoT,'*');
209     % xlabel('Length platform in cm')
210     % ylabel('Width platform in cm')
211     % title('Max RSSI signal of every tag')
212
213     figure(4)   % plot of the RSSI signal which are non zero vs. time
214     dataRSSIsum = sum(dataRSSI);
215     IDclear = find(dataRSSIsum ~= 0);
216     IDstr = string(IDclear);
217     dataRSSIclear = dataRSSI;
218     dataRSSIclear( :, all( ~any( dataRSSI ), 1 ) ) = []; % and columns
219     plot(dataRSSIclear);
220     xlabel('Measurement points')
221     ylabel('RSSI')
222     ylim([0 360/angle1])
223     legend(IDstr,'FontSize',6);
224     title('RSSI Signal of every non zero tag')
```

11.2 Appendix B: Receiving data from reader via Wifi (C#)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.IO;
using System.Threading;
using System.Net;
using System.Net.Sockets;
using MULTIFORM_PCS.ControlModules.SchedulingModule;
using MULTIFORM_PCS.ControlModules.FeedForwadModule;
using MULTIFORM_PCS.ControlModules.RoutingModule.PathAndVelocityPlanning.
    DataTypes;
using MULTIFORM_PCS.ControlModules.CameraModule.CameraForm;
using MULTIFORM_PCS.ControlModules.CameraControl.CameraControlClass;
using System.Windows.Threading;
using System.Diagnostics; // Process
using System.Globalization;
using Emgu.CV.WPF;
using System.Threading.Tasks;
using System.Collections.Concurrent;

namespace MULTIFORM_PCS.ControlModules.RFID
{
    public class receive
    {
        public string[] availablearray=new string[1];
        public void connect()
        {
            try
            {
                Console.WriteLine("Connecting");
                TcpClient tcpClient = new TcpClient("192.168.0.100", 8883);
                if (tcpClient.Connected)
                {
                    Console.WriteLine("Connected to server");
                }
            }
            catch (Exception e)
            {
                Console.WriteLine("Connection Failed");
            }
        }

        public void reading(CancellationToken ct)
        {
            if (ct.IsCancellationRequested == true)
            {
                ct.ThrowIfCancellationRequested();
```

```
49              }
50
51              Console.WriteLine("Connecting");
52              TcpClient tcpClient = new TcpClient("192.168.0.100", 8883);
53
54              if (tcpClient.Connected)
55              {
56                  Console.WriteLine("Connected to server");
57              }
58
59              using (StreamReader STR = new StreamReader(tcpClient.GetStream()))
60              {
61                  string recieve;
62                  char[] trash = new char[16];
63                  char[] UID = new char[3];
64                  char[] RSSI = new char[3];
65                  long milliseconds, seconds, minutes;
66                  string UID_, RSSI_, RSSI___;
67                  string[] array = new string[1];
68
69                  List<string> RSSI__;
70                  int UID_DEC=0;
71                  int RSSI_int = 0;
72
73                      while ((recieve = STR.ReadLine()) != null && !ct.
                            IsCancellationRequested)
74                      {
75                          if (ct.IsCancellationRequested)
76                          {
77                              try
78                              {
79                                  ct.ThrowIfCancellationRequested();
80                              }
81                              catch (AggregateException e)
82                              {
83                              }
84                          }
85
86                          if (recieve.Contains("+"))
87                          {
88
89                          List<string> Worte = recieve.Split(new string[] { "OK",
                                "<\\r>", "\n", "", "SCAN:+UID=", "+RSSI=" },
                                StringSplitOptions.RemoveEmptyEntries).ToList();
90                          string Wort = string.Join("", Worte.ToArray());
91
92                          using (StringReader sr = new StringReader(Wort))
93                          {
94                              sr.Read(trash, 0, 13);
95                              sr.Read(UID, 0, 3);
96                              UID_ = new string(UID);
```

```
97                              sr.Read(trash, 0, 1);
98                              sr.Read(RSSI, 0, 1);
99                              RSSI_ = new string(RSSI);
100                             try
101                             {
102                                 UID_DEC = Int32.Parse(UID_, System.Globalization
                                        .NumberStyles.HexNumber);
103                             }
104                             catch (Exception e)
105                             {
106                             }
107                         }
108
109                         RSSI__ = RSSI_.Split(new string[] { "," },
                                StringSplitOptions.RemoveEmptyEntries).ToList();
110                         RSSI___ = string.Join("", RSSI__.ToArray());
111                         try
112                         {
113                             RSSI_int = Int32.Parse(RSSI___);
114                         }
115                         catch (Exception e)
116                         {
117                         }
118
119                         milliseconds = DateTimeOffset.Now.Millisecond;
120                         seconds = DateTimeOffset.Now.Second;
121                         minutes = DateTimeOffset.Now.Minute;
122                         array[0] = minutes + " " + seconds + " " + milliseconds
                                + " " + UID_DEC + " " + RSSI_int;
123                         //File.AppendAllText(AppDomain.CurrentDomain.
                                BaseDirectory + "\\pythonfiles\\python_1robot\\
                                RFID_Data.log", minutes + " " + seconds + " " +
                                milliseconds + "\t UID: " + UID_ + " RSSI: " +
                                RSSI___ + "\r");
124                         //File.AppendAllText(AppDomain.CurrentDomain.
                                BaseDirectory + "\\pythonfiles\\python_1robot\\
                                RFID_Data_original.log", hour + ":" + minutes + ":"
                                + seconds + ":" + milliseconds + "\t" + recieve + "\
                                r");
125                         //Console.WriteLine(minutes + " " + seconds + " " +
                                milliseconds + "\t" + " " + UID_ + "  " + RSSI___);
126                         }
127                             this.availablearray[0] = array[0];
128                         }
129                     }
130                 }
131
132
133
134         public void disconnect()
135         {
```

```
136             TcpClient tcpClient = new TcpClient();
137             tcpClient.Connect("192.168.0.100", 8883);
138             tcpClient.Close();
139
140         }
141 }
142 }
```

11.3 Appendix C: Initialization procedure (C#)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using MULTIFORM_PCS.ControlModules.CameraModule.CameraForm;
using System.Threading;
using MULTIFORM_PCS.GUI;
using MULTIFORM_PCS.Gateway.ConnectionModule;
using MULTIFORM_PCS.ControlModules.RFID;
using System.Threading.Tasks;
using System.Collections;

namespace MULTIFORM_PCS.ControlModules.MPCModule
{
    public class Position
    {
        public int X = 0;
        public int Y = 0;
    }

    public class PositionD
    {
        public double X = 0;
        public double Y = 0;
    }

    class Init
    {
                //See Appendix D - I
        }
}
```

11.4 Appendix D: Initialization turn and recording Data(C#)

```

        public static void initialize(Int32 time)
        {
                int messungen = 100;
                //Gateway.ConnectionModule.ConnectionCTRLModule.getInstance().
                    setCTRLForRobot(0, 0.0, 100.0, 0.0, 8.0, 0.0, 0.0, 3.0);
                receive initial = new receive();    //Create a new instance of
                    class Receive
                var tokenSource = new CancellationTokenSource();
                var token = tokenSource.Token;
                Init compare = new Init();
                string[] rfid_signals = new string[messungen];
                int currentRobot = 0;
                int[] RobotAssingment = new int[] { 0, 1, 3 };
                Gateway.CTRLModule.getInstance().camCtrl.
                    processFrameAndUpdateGUI();
```

```
14                  RobotDiscription[] RobotArray = new RobotDiscription[] { Gateway
                        .CTRLModule.getInstance().camCtrl.RobotA, Gateway.CTRLModule
                        .getInstance().camCtrl.RobotB,
15                  Gateway.CTRLModule.getInstance().camCtrl.RobotC };
16                  double[][] velocity1 = new double[RobotArray.Length][];
17                  string[,] Signals = new string[messungen,8];
18                  velocity1[currentRobot] = new double[] { 0, 0 }; //Starts the
                        Robot
19                  Gateway.CTRLModule.getInstance().getRobotRemoteCTRL(
                        RobotAssingment[currentRobot]).forward(velocity1[
                        currentRobot], 0, 0, 0); //Sends velocity to Robot
20                  //Opening a new Task which works in the background to read data
                        from RFID Antenna
21                  Task t = Task.Factory.StartNew(() => initial.reading(token));
22                  Thread.Sleep(1000);
23                  for (int i = 0; i < 8; i++)      //9 Because of 8 measurements
                        every 45 degree
24                  {
25                      for (int j = 0; j < messungen; j++)     //in this for loop we
                            find all the reachable TAGs
26                      {
27                          Signals[j, i] = initial.availablearray[0];
28                          Thread.Sleep(100);
29                      }
30                      velocity1[currentRobot] = new double[] { 100, -100 }; //
                            Starts the Robot
31                      Gateway.CTRLModule.getInstance().getRobotRemoteCTRL(
                            RobotAssingment[currentRobot]).forward(velocity1[
                            currentRobot], 0, 0, 0); //Sends velocity to Robot
32                      Thread.Sleep(time); //time the robot needs for a 45 degree
                            turn
33                      velocity1[currentRobot] = new double[] { 0, 0 };  //Stops
                            the Robot
34                      Gateway.CTRLModule.getInstance().getRobotRemoteCTRL(
                            RobotAssingment[currentRobot]).forward(velocity1[
                            currentRobot], 0, 0, 0); //Sends velocity to Robot
35                  }
36
37                  tokenSource.Cancel();    //close the reading Thread
38                  try
39                  {
40                      Task.WaitAll(t);
41                  }
42                  catch (AggregateException e)
43                  {
44                  }
45                  finally
46                  {
47                      tokenSource.Dispose();
48                  }
49                      Console.WriteLine("END\r\r");
```

```
50
51              Array[] Liste = new Array[8];    //List of arrays each array in the
                    array contains the data of a special position (45Â°, 90Â°,...)
52              string[,] Init_array = new string[8, 14];    //Array filled with
                    signal strengthes and ID of every degree position
53              string temp_ID="begin", temp_RSSI;  //Substrings of Data
54              int counter;    //Counter for the row in the Init_Array
55              Console.WriteLine("");
```

11.5 Appendix E: Filling Array(C#)

```
1  for (int j = 0; j < 8; j++)//converting UID to the specific decimal numbers in
       lookup table
2               {
3                   counter = 0;
4                   for (int i = 0; i < messungen; i++)
5                   {
6                       try
7                       {
8                           temp_ID = Signals[i, j].Substring(Signals[i, j].Length -
                                6, 4);   //seperatiion of UID in the string
9                           if (temp_ID == "2788")
10                          {
11                              temp_ID = "1";
12                          }
13                          if (temp_ID == "1414")
14                          {
15                              temp_ID = "2";
16                          }
17                          if (temp_ID == "3060")
18                          {
19                              temp_ID = "3";
20                          }
21                          if (temp_ID == "1673")
22                          {
23                              temp_ID = "4";
24                          }
25                          if (temp_ID == "1925" || temp_ID == "1025")
26                          {
27                              temp_ID = "5";
28                          }
29                          if (temp_ID == "1681")
30                          {
31                              temp_ID = "6";
32                          }
33                          if (temp_ID == "1146")
34                          {
35                              temp_ID = "7";
36                          }
37                          if (temp_ID == "2780")
38                          {
39                              temp_ID = "8";
```

```
40                          }
41                          if (temp_ID == "1933")
42                          {
43                              temp_ID = "9";
44                          }
45                      }
46                      catch (AggregateException e)
47                      {
48                          Console.WriteLine("Array incomplete");
49                      }
50
51                      temp_RSSI = Signals[i, j].Substring(Signals[i, j].Length -
                            1, 1);     //seperation of RSSI in the string
52                      if (temp_ID != Init_array[j, 0] && temp_ID != Init_array[j,
                            1] && temp_ID != Init_array[j, 2] && temp_ID !=
                            Init_array[j, 3] && temp_ID != Init_array[j, 4] &&
                            temp_ID != Init_array[j, 5] && temp_ID != Init_array[j,
                            6] && temp_ID != Init_array[j, 7])  //check if the UID
                            already exists in the Init_array
53                      {
54                          //Filling Init_Array
55                          Init_array[j, counter] = temp_ID;
56                          Init_array[j, counter + 7] = temp_RSSI;
57                          counter++;
58                      }
59                  }
60              }
```

11.6 Appendix F: Checking for solutions in array(C#)

```
1   int rowLength = Init_array.GetLength(0);
2                   int colLength = Init_array.GetLength(1);
3                   string str;
4                   string headline = "|" + "ID 1" + "|" + "ID 2" + "|" + "ID 3" + "
                        |" + "ID 4" + "|" + "ID 5" + "| " + "ID 6" + "|" + "ID 7" +
                        "|" + "ST 1" + "|" + "ST 2" + "|" + "ST 3" + "|" + "ST 4" +
                        "|" + "ST 5" + "|" + "ST 6" + "|" + "ST 7" + "|";
5                   System.Console.WriteLine(headline);
6
7                   for (int k = 0; k < rowLength; k++)
8                   {
9                       str = "|" + Init_array[k, 0] + "    |" + Init_array[k, 1] + "
                                |" + Init_array[k, 2] + "    |" + Init_array[k, 3] + "
                                |" + Init_array[k, 4] + "    |" + Init_array[k, 5] + "
                                |" + Init_array[k, 6] + "    |" + Init_array[k, 7] + "
                                |" + Init_array[k, 8] + "    |" + Init_array[k, 9] + "
                                |" + Init_array[k, 10] + "    |" + Init_array[k, 11] +
                             "    |" + Init_array[k, 12] + "    |" + Init_array[k, 13]
                             + "    |";
10                      System.Console.WriteLine(str);
11                  }
12
```

```
13                  // Solver
14                  // Different Positions
15                  Position Starting = new Position();
16                  Position Antenna1 = new Position();
17                  Position Antenna2 = new Position();
18                  Position Antenna3 = new Position();
19                  Position Antenna4 = new Position();
20
21                  //Initialization for Position estimation
22                  float m1 = 0.000f;
23                  float m2 = 0.000f;
24
25                  float RobStartx_fl = 0.000f;
26                  float RobStarty_fl = 0.000f;
27
28                  double angle;
29                  double angleTemp;
30
31                  int null_counter = 0;
32                  int[] check_row = new int[8];
33                  for (int m = 0; m < rowLength; m++)
34                  {
35                      null_counter = 0;
36                      for (int n = 0; n < 7; n++)
37                      {
38                          if (Init_array[m, n] == null)
39                          {
40                              null_counter++;
41                          }
42                      }
43                      check_row[m] = 7 - null_counter;    //Array of elements with
                            the number empty places of each init_array row
44                      System.Console.WriteLine("The number of elements at " + m *
                            45 + "Â° is: \t" + check_row[m]);
45                  }
46                  bool solution_found = false;    //true if initialization process
                        is solvable
47                  bool solution1_found = false;   //true if one possible point is
                        found
48                  bool solution2_found = false;   //true if two possible points
                        are found
49                  int count = 0;
50                  int[] solution1 = new int[2];   //Array with the both degree
                        numbers of solution 1
51                  int[] solution2 = new int[2];   //Array with the both degree
                        numbers of solution 2
52                  while (solution_found == false)
53                  {
54                      while (solution1_found == false)
55                      {
56                          if (check_row[count] >= 3)
```

```
57                          {
58                              if (check_row[count + 4] >= 3)
59                              {
60                                  solution1[0] = count;
61                                  solution1[1] = count + 4;
62                                  break;
63                              }
64                              if (count >= 2)     //if we reach the 180 degree
                                    there will be no solution for this
                                    initialization turn
65                              {
66                                  System.Console.WriteLine("NO SOLUTION FOUND!!!")
                                        ;
67                                  break;
68                              }
69                          }
70                          count++;
71                      }
72                      System.Console.WriteLine(count);
73                      count = count + 1;
74                      while (solution2_found == false)
75                      {
76                          if (check_row[count] >= 3)
77                          {
78                              if (count >= 8)
79                              {
80                                  Console.WriteLine("Out of Range Exception caused
                                        in Array: count");
81                              }
82                              if (check_row[count + 4] >= 3)
83                              {
84                                  solution2[0] = count;
85                                  solution2[1] = count + 4;
86                                  solution_found = true;
87                                  break;
88                              }
89                              if (count >= 3)     //if we reach the 180 degree
                                    there will be no solution for this
                                    initialization turn
90                              {
91                                  System.Console.WriteLine("NO SOLUTION FOUND!!!")
                                        ;
92                                  break;
93                              }
94                              else
95                              {
96                                  //count = count - 1;
97                                  break;
98                              }
99                          }
100                     }
```

```
101                     System.Console.WriteLine("Solution No. 1 found at: " +
                            solution1[0] * 45 + " degree -- " + solution1[1] * 45 +
                            " degree");
102                     System.Console.WriteLine("Solution No. 2 found at: " +
                            solution2[0] * 45 + " degree -- " + solution2[1] * 45 +
                            " degree");
103                 }
```

11.7 Appendix G: Position and orientation estimation(C#)

```
1   // Providing the distance with the highest probability
2                   // Input:   # of tags, all IDs of the tags
3                   // Output:  best fitting IDs (e.g.[3 4 5] if 3rd, 4th and 5th
                        are best ones)
4                   //          the correct distance <-> RSSI signal (e.g.[2 1 3]
                        for middle, max and min)
5                   int[,] best_arr1 = new int[2, 3];
6                   int[,] best_arr2 = new int[2, 3];
7                   int[,] best_arr3 = new int[2, 3];
8                   int[,] best_arr4 = new int[2, 3];
9
10                  int[] temp_input1 = new int[7];
11                  int[] temp_input2 = new int[7];
12                  int[] temp_input3 = new int[7];
13                  int[] temp_input4 = new int[7];
14
15                  int[] temp_inputRSSI1 = new int[7];
16                  int[] temp_inputRSSI2 = new int[7];
17                  int[] temp_inputRSSI3 = new int[7];
18                  int[] temp_inputRSSI4 = new int[7];
19
20                  for (int i = 0; i < 8; i++)
21                  {
22                      for (int j = 0; j < 14; j++)
23                      {
24                          if (Init_array[i, j] == null)
25                          {
26                              Init_array[i, j] = "0";
27                          }
28                      }
29                  }
30
31                  for (int m = 0; m < 7; m++)
32                  {
33                      temp_input1[m] = Int32.Parse(Init_array[solution1[0], m]);
34                      temp_input2[m] = Int32.Parse(Init_array[solution1[1], m]);
35                      temp_input3[m] = Int32.Parse(Init_array[solution2[0], m]);
36                      temp_input4[m] = Int32.Parse(Init_array[solution2[1], m]);
37
38                      temp_inputRSSI1[m] = Int32.Parse(Init_array[solution1[0], m
                            + 7]);
```

```
39                      temp_inputRSSI2[m] = Int32.Parse(Init_array[solution1[1], m
                            + 7]);
40                      temp_inputRSSI3[m] = Int32.Parse(Init_array[solution2[0], m
                            + 7]);
41                      temp_inputRSSI4[m] = Int32.Parse(Init_array[solution2[1], m
                            + 7]);
42                  }
43
44                  best_arr1 = CorrectID_Distance(temp_input1, temp_inputRSSI1,
                        check_row[solution1[0]]);
45                  best_arr2 = CorrectID_Distance(temp_input2, temp_inputRSSI2,
                        check_row[solution1[1]]);
46                  best_arr3 = CorrectID_Distance(temp_input3, temp_inputRSSI3,
                        check_row[solution2[0]]);
47                  best_arr4 = CorrectID_Distance(temp_input4, temp_inputRSSI4,
                        check_row[solution2[1]]);
48
49                  // Position of the antennae
50                  Antenna1 = Trilateration(IDtoPOS(Int32.Parse(Init_array[
                        solution1[0], best_arr1[0, 0]])), IDtoPOS(Int32.Parse(
                        Init_array[solution1[0], best_arr1[0, 1]])),
51                                           IDtoPOS(Int32.Parse(Init_array[solution1
                                                 [0], best_arr1[0, 2]])), Int32.Parse
                                                 (Init_array[solution1[0], best_arr1
                                                 [0, 0] + 7]),
52                                           Int32.Parse(Init_array[solution1[0],
                                                 best_arr1[0, 1] + 7]), Int32.Parse(
                                                 Init_array[solution1[0], best_arr1
                                                 [0, 2] + 7]),
53                                           best_arr1[1, 0], best_arr1[1, 1],
                                                 best_arr1[1, 2]);
54
55                  Antenna2 = Trilateration(IDtoPOS(Int32.Parse(Init_array[
                        solution1[1], best_arr2[0, 0]])), IDtoPOS(Int32.Parse(
                        Init_array[solution1[1], best_arr2[0, 1]])),
56                                           IDtoPOS(Int32.Parse(Init_array[solution1
                                                 [1], best_arr2[0, 2]])), Int32.Parse
                                                 (Init_array[solution1[1], best_arr2
                                                 [0, 0] + 7]),
57                                           Int32.Parse(Init_array[solution1[1],
                                                 best_arr2[0, 1] + 7]), Int32.Parse(
                                                 Init_array[solution1[1], best_arr2
                                                 [0, 2] + 7]),
58                                           best_arr2[1, 0], best_arr2[1, 1],
                                                 best_arr2[1, 2]);
59
60                  Antenna3 = Trilateration(IDtoPOS(Int32.Parse(Init_array[
                        solution2[0], best_arr3[0, 0]])), IDtoPOS(Int32.Parse(
                        Init_array[solution2[0], best_arr3[0, 1]])),
61                                           IDtoPOS(Int32.Parse(Init_array[solution2
                                                 [0], best_arr3[0, 2]])), Int32.Parse
```

```
                                        (Init_array[solution2[0], best_arr3
                                        [0, 0] + 7]),
62                                  Int32.Parse(Init_array[solution2[0],
                                        best_arr3[0, 1] + 7]), Int32.Parse(
                                        Init_array[solution2[0], best_arr3
                                        [0, 2] + 7]),
63                                  best_arr3[1, 0], best_arr3[1, 1],
                                        best_arr3[1, 2]);
64
65              Antenna4 = Trilateration(IDtoPOS(Int32.Parse(Init_array[
                    solution2[1], best_arr4[0, 0]])), IDtoPOS(Int32.Parse(
                    Init_array[solution2[1], best_arr4[0, 1]])),
66                                  IDtoPOS(Int32.Parse(Init_array[solution2
                                        [1], best_arr4[0, 2]])), Int32.Parse
                                        (Init_array[solution2[1], best_arr4
                                        [0, 0] + 7]),
67                                  Int32.Parse(Init_array[solution2[1],
                                        best_arr4[0, 1] + 7]), Int32.Parse(
                                        Init_array[solution2[1], best_arr4
                                        [0, 2] + 7]),
68                                  best_arr4[1, 0], best_arr4[1, 1],
                                        best_arr4[1, 2]);
69
70              Console.WriteLine("1st Antenna " + Antenna1.X + " and " +
                    Antenna1.Y);
71              Console.WriteLine("2nd Antenna " + Antenna2.X + " and " +
                    Antenna2.Y);
72              Console.WriteLine("3rd Antenna " + Antenna3.X + " and " +
                    Antenna3.Y);
73              Console.WriteLine("4th Antenna " + Antenna4.X + " and " +
                    Antenna4.Y);
74
75              //Console.ReadKey();
76              // Alternative estimation of the centre of the robot + position
77              //m1 = ((float)Antenna2.Y - (float)Antenna1.Y) / ((float)
                    Antenna2.X - (float)Antenna1.X);
78              //m2 = ((float)Antenna4.Y - (float)Antenna3.Y) / ((float)
                    Antenna4.X - (float)Antenna3.X);
79              //RobStartx_fl = (1 / (m1 - m2)) * (m1 * (float)Antenna1.X - m2
                    * (float)Antenna3.X - (float)Antenna1.Y + (float)Antenna3.Y)
                    ;
80              //RobStarty_fl = m1 * (RobStartx_fl - (float)Antenna1.X) + (
                    float)Antenna1.Y;
81
82              //Starting.X = (int)RobStartx_fl;
83              //Starting.Y = (int)RobStarty_fl;
84
85              Starting.X = (((Antenna4.X-Antenna3.X)*(Antenna2.X*Antenna1.Y-
                    Antenna1.X*Antenna2.Y)-(Antenna2.X-Antenna1.X)*(Antenna4.X*
                    Antenna3.Y-Antenna3.X*Antenna4.Y)) /
86                                  ((Antenna4.Y - Antenna3.Y) * (Antenna2.X -
```

```
                                        Antenna1.X) - (Antenna2.Y - Antenna1.Y)
                                        * (Antenna4.X - Antenna3.X)));
87              Starting.Y = (((Antenna1.Y - Antenna2.Y) * (Antenna4.X *
                    Antenna3.Y - Antenna3.X * Antenna4.Y) - (Antenna3.Y -
                    Antenna4.Y) * (Antenna2.X * Antenna1.Y - Antenna1.X *
                    Antenna2.Y)) /
88                                  ((Antenna4.Y - Antenna3.Y) * (Antenna2.X -
                                        Antenna1.X) - (Antenna2.Y - Antenna1.Y)
                                        * (Antenna4.X - Antenna3.X)));
89
90              Console.WriteLine("Robotstarting Position at:" + Starting.X + "
                    mm, " + Starting.Y + "mm");
91
92              // Computing the orientation of the Robot
93              //angle = (Math.Atan2(y, x)) * (180 / Math.PI);
94              angleTemp = (Math.Atan2((Antenna1.Y - Starting.Y), (Antenna1.X -
                     Starting.X))) * (180 / Math.PI);
95              Console.WriteLine("Angle temp " + angleTemp);
96              angle = angleTemp - (double)(solution1[0]*45.0);  // in deg
97              Console.WriteLine("Angle wrong direction " + angle);
98              if (angle <= 0.0)
99              {
100                 angle = angle + 180;
101             }
102             else
103             {
104                 angle = angle - 180;
105             }
106
107            Console.WriteLine("Robotangle: " + angle + "Â°");
108
109         }
```

11.8 Appendix H: Initialization procedure 3(C#)

```
1   // Procedure and function
2   //Methode to compute the position based on the ID in [cm], Output in [mm]
3           public static Position Trilateration(Position point1, Position point2,
                Position point3, int r1t, int r2t, int r3t, int bestr1, int bestr2,
                int bestr3)
4           {
5               //double[] dist = new double[] { 10.5, 10.0, 9.5, 9.0, 8.0, 6.0,
                    5.0, 4.0 }; // FH paper
6               double[,] dist = new double[3, 8] { { 14, 9.75, 9.0, 8.0, 7.0, 6.0,
                    3.5, 2.75 },
7                                                   { 5.0, 5.1, 5.3, 5.5, 5.8, 4.0,
                                                        3.5, 2.75 },
8                                                   { 5.0, 4.7, 4.5, 4.3, 4.2, 4.0,
                                                        3.5, 2.75} }; //
                                                        Approximation of our
                                                        measurements
9
```

```
10                          Position resultPose = new Position();
11                          PositionD ex = new PositionD();
12                          PositionD ey = new PositionD();
13                          PositionD aux = new PositionD();
14                          PositionD auy = new PositionD();
15                          PositionD aux2 = new PositionD();
16                          double r1;
17                          double r2;
18                          double r3;
19                          r1 = dist[bestr1, r1t];
20                          r2 = dist[bestr2, r2t];
21                          r3 = dist[bestr3, r3t];
22
23              // For testing purpose
24              //Console.WriteLine("1st radius " + r1);
25              //Console.WriteLine("2nd radius " + r2);
26              //Console.WriteLine("3rd radius " + r3);
27
28              //Console.WriteLine("1st point " + point1.X + " " + point1.Y);
29              //Console.WriteLine("2nd point " + point2.X + " " + point2.Y);
30              //Console.WriteLine("3rd point " + point3.X + " " + point3.Y);
31
32              //unit vector in a direction from point1 to point 2
33              double p2p1Distance = Math.Pow(Math.Pow(point2.X - point1.X, 2) +
                    Math.Pow(point2.Y - point1.Y, 2), 0.5);
34              ex.X = (point2.X - point1.X) / p2p1Distance;
35              ex.Y = (point2.Y - point1.Y) / p2p1Distance;
36              aux.X = point3.X - point1.X;
37              aux.Y = point3.Y - point1.Y;
38              //signed magnitude of the x component
39              double i = ex.X * aux.X + ex.Y * aux.Y;
40              //the unit vector in the y direction.
41              aux2.X = point3.X - point1.X - i * ex.X;
42              aux2.Y = point3.Y - point1.Y - i * ex.Y;
43              ey.X = aux2.X / Norm(aux2);
44              ey.Y = aux2.Y / Norm(aux2);
45              //the signed magnitude of the y component
46              double j = ey.X * aux.X + ey.Y * aux.Y;
47              //coordinates
48              double x = (Math.Pow(r1, 2) - Math.Pow(r2, 2) + Math.Pow(
                    p2p1Distance, 2)) / (2 * p2p1Distance);
49              double y = (Math.Pow(r1, 2) - Math.Pow(r3, 2) + Math.Pow(i, 2) +
                    Math.Pow(j, 2)) / (2 * j) - i * (x / j);
50              //result coordinates
51              double finalX = 10 * (point1.X + x * ex.X + y * ey.X);
52              double finalY = 10 * (point1.Y + x * ex.Y + y * ey.Y);
53              resultPose.X = (int)(finalX);
54              resultPose.Y = (int)(finalY);
55
56              return resultPose;
57          }
```

11.9 Appendix I: Initialization procedure 4(C#)

```
1
2           // Method to compute the norm of a vector
3           public static double Norm(PositionD p) // get the norm of a vector
4           {
5               return (Math.Pow(Math.Pow(p.X, 2) + Math.Pow(p.Y, 2), 0.5));
6           }
7
8                   // Methode to compute the position based on the ID in [mm]
9           public static Position IDtoPOS(int ID)
10          {
11              Position FinalPos = new Position();
12              int[] posx = new int[9] { 10, 10, 10, 20, 20, 20, 30, 30, 30 };
13              int[] posy = new int[9] { 10, 20, 30, 10, 20, 30, 10, 20, 30 };
14              // For a 3x3 testing field
15              FinalPos.X = posx[ID - 1];
16              FinalPos.Y = posy[ID - 1];
17              return FinalPos;
18          }
```

11.10 Appendix J: Initialization procedure 5(C#)

```
1   // Find neighbours of the IDs
2           public static int[] FindNeig(int[] arrID, int numTags)
3           {
4               // Init
5               int[] neighbours = new int[numTags];
6               int[] tempNeig = new int[4];
7
8               // Find the number of neighbours
9               for (int m = 0; m < numTags; m++)
10              {
11                  // Init
12                  neighbours[m] = 0;
13
14                  // Take actual ID and compute the possible neighbours
15                  tempNeig[0] = arrID[m] - 11;
16                  tempNeig[1] = arrID[m] - 1;
17                  tempNeig[2] = arrID[m] + 1;
18                  tempNeig[3] = arrID[m] + 11;
19
20                  for (int v = 0; v < 4; v++)
21                  {
22                      foreach (int tempinput in arrID)
23                      {
24                          if (tempinput == tempNeig[v])
25                          {
26                              neighbours[m] += 1;
27                          }
28                      }
29                  }
```

```
30              }
31              return neighbours;
32          }
```

11.11 Appendix K: Initialization procedure 6(C#)

```
1   // Methode to compute the best IDs and correct distances
2           public static int[,] CorrectID_Distance(int[] arr, int[] arrRSSI, int
                numTags)
3           {
4               // Inputs
5               /*  arr = Array of all IDs
6                *  arrRSSI = Array of all RSSI
7                   numTags = Int with the num of tags found
8               */
9               // Init
10              int[,] best = new int[2, 3];
11              int[] dist = new int[numTags];          // Array which contain the
                    best distance (max(0),middle(1),min(2))
12              int[] neighbours = new int[numTags];
13              int i = 0;
14              int p = 4;
15
16              // Compute the neighbours
17              neighbours = FindNeig(arr, numTags);
18
19
20              // Switch case for the different possible shapes
21              switch (numTags)
22              {
23                  case 3:
24                      Console.WriteLine("3 Tags ----------");
25                      for (int l = 0; l < neighbours.GetLength(0); l++)
26                      {
27                          if (neighbours[l] == 0)      // Detect outlier and
                                boarder
28                          {
29                              dist[l] = 1;             // stay max
30                          }
31                          else if (neighbours[l] <= 3)     // Detect outlier and
                                boarder
32                          {
33                              dist[l] = 0;             // stay max
34                          }
35                          else if (neighbours[l] == 4)     // Detect the inner,
                                change it to min/middle
36                          {
37                              dist[l] = 1;             // change it to middle
38                          }
39                          else if (neighbours[l] > 4)      // Detect the inner,
                                change it to min/middle
40                          {
```

```
41                          dist[l] = 0;            // change it to middle
42                      }
43                      // all other numbers are at the boarder
44                  }
45                  // Select the best 3 readings
46                  i = 0;
47                  p = 4;
48                  while (i < 3)                   // Start for the first ID
49                  {
50                      for (int h = 0; h < neighbours.GetLength(0); h++)   //
                            looks for a fitting
51                      {
52                          if (neighbours[h] == p && i < 3)    // hit must be
                                same value and less then 3 hits
53                          {
54                              best[0, i] = h;                 // index of the
                                    best ID
55                              best[1, i] = dist[h];           // info about
                                    max, mid and min of this ID
56                              i += 1;
57                          }
58                          else if (i >= 3)
59                          {
60                              break;
61                          }
62                      }
63                      p -= 1;
64                  }
65                  break;
66              /*
                    ------------------------------------------------------------------------------
                    */
67              case 4:
68                  Console.WriteLine("4 Tags ----------");
69                  for (int l = 0; l < neighbours.GetLength(0); l++)
70                  {
71                      if (neighbours[l] == 0)     // Detect outlier and
                            boarder
72                      {
73                          dist[l] = 1;            // stay max
74                      }
75                      else if (neighbours[l] <= 3)     // Detect outlier and
                            boarder
76                      {
77                          dist[l] = 0;            // stay max
78                      }
79                      else if (neighbours[l] == 4)     // Detect the inner,
                            change it to min/middle
80                      {
81                          dist[l] = 1;            // change it to middle
82                      }
```

```
83                          else if (neighbours[l] > 4)      // Detect the inner,
                                change it to min/middle
84                          {
85                              dist[l] = 0;             // change it to middle
86                          }
87                          // all other numbers are at the boarder
88                      }
89                      // Select the best 3 readings
90                      i = 0;
91                      p = 4;
92                      while (i < 3)                    // Start for the first ID
93                      {
94                          for (int h = 0; h < neighbours.GetLength(0); h++)   //
                                looks for a fitting
95                          {
96                              if (neighbours[h] == p && i < 3)    // hit must be
                                    same value and less then 3 hits
97                              {
98                                  best[0, i] = h;                 // index of the
                                        best ID
99                                  best[1, i] = dist[h];           // info about
                                        max, mid and min of this ID
100                                 i += 1;
101                             }
102                             else if (i >= 3)
103                             {
104                                 break;
105                             }
106                         }
107                         p -= 1;
108                     }
109
110                     break;
111                 /*
                        ------------------------------------------------------------------------
                    */
112                 case 5:
113                     Console.WriteLine("5 Tags ----------");
114                     for (int l = 0; l < neighbours.GetLength(0); l++)
115                     {
116                         if (neighbours[l] <= 2)      // Detect outlier and
                                boarder
117                         {
118                             dist[l] = 0;             // stay max
119                         }
120                         else if (neighbours[l] == 3)     // Detect the inner,
                                change it to min/middle
121                         {
122                             dist[l] = 1;             // change it to middle
123                         }
124                         // all other numbers are at the boarder
```

```
125                     }
126                     // Select the best 3 readings
127                     i = 0;
128                     p = 4;
129                     while (i < 3)                   // Start for the first ID
130                     {
131                         for (int h = 0; h < neighbours.GetLength(0); h++)   //
                                looks for a fitting
132                         {
133                             if (neighbours[h] == p && i < 3)    // hit must be
                                    same value and less then 3 hits
134                             {
135                                 best[0, i] = h;                 // index of the
                                        best ID
136                                 best[1, i] = dist[h];           // info about
                                        max, mid and min of this ID
137                                 i += 1;
138                             }
139                             else if (i >= 3)
140                             {
141                                 break;
142                             }
143                         }
144                         p -= 1;
145                     }
146                     break;
147
148                 /*
                        ------------------------------------------------------------------------------
                        */
149                 case 6:
150                     Console.WriteLine("6 Tags ----------");
151                     switch (neighbours.Sum())
152                     {
153                         case 12: // Shape with 2 outliers
154                             Console.WriteLine("2 Outliers");
155                             for (int l = 0; l < neighbours.GetLength(0); l++)
156                             {
157                                 if (neighbours[l] <= 2)     // Detect outlier
                                        and boarder
158                                 {
159                                     dist[l] = 0;            // stay max
160                                 }
161                                 else if (neighbours[l] == 4)     // Detect the
                                        inner, change it to min/middle
162                                 {
163                                     if (arrRSSI[l] <= 3)
164                                     {
165                                         dist[l] = 2;            // change it to
                                                min
166                                     }
```

```
167                                 else if (arrRSSI[l] > 3)
168                                 {
169                                     dist[l] = 1;            // change it to
                                            middle
170                                 }
171                             }
172                             // all other numbers are at the boarder
173                         }
174                         break;
175                     case 14: // Shape like a domino
176                         Console.WriteLine("Domino");
177                         for (int l = 0; l < neighbours.GetLength(0); l++)
178                         {
179                             if (neighbours[l] <= 2)     // Detect outlier
                                    and boarder
180                             {
181                                 dist[l] = 0;            // stay max
182                             }
183                             else if (neighbours[l] == 3)     // Detect the
                                    centre, change it to min
184                             {
185                                 dist[l] = 2;            // change it to min
186                             }
187                             // all other numbers are at the boarder
188                         }
189                         break;
190                     default:
191                         Console.WriteLine("Default case");
192                         break;
193                 }
194                 // Select the best 3 readings
195                 i = 0;
196                 p = 4;
197                 while (i < 3)                   // Start for the first ID
198                 {
199                     for (int h = 0; h < neighbours.GetLength(0); h++)   //
                            looks for a fitting
200                     {
201                         if (neighbours[h] == p && i < 3)    // hit must be
                                same value and less then 3 hits
202                         {
203                             best[0, i] = h;                 // index of the
                                    best ID
204                             best[1, i] = dist[h];           // info about
                                    max, mid and min of this ID
205                             i += 1;
206                         }
207                         else if (i >= 3)
208                         {
209                             break;
210                         }
```

```
211                         }
212                         p -= 1;
213                     }
214
215                     break;
216                 /*
                    ----------------------------------------------------------------------------
                    */
217                 case 7:
218                     Console.WriteLine("7 Tags ----------");
219                     for (int l = 0; l < neighbours.GetLength(0); l++)
220                     {
221                         if (neighbours[l] <= 3)      // Detect outlier and
                            boarder
222                         {
223                             dist[l] = 0;             // stay max
224                         }
225                         else if (neighbours[l] == 4)     // Detect the centre,
                            change it to min
226                         {
227                             dist[l] = 2;             // change it to min
228                         }
229                         // all other numbers are at the boarder
230                     }
231                     // Select the best 3 readings
232                     i = 0;
233                     p = 4;
234                     while (i < 3)                    // Start for the first ID
235                     {
236                         for (int h = 0; h < neighbours.GetLength(0); h++)   //
                            looks for a fitting
237                         {
238                             if (neighbours[h] == p && i < 3)     // hit must be
                                same value and less then 3 hits
239                             {
240                                 best[0, i] = h;                  // index of the
                                    best ID
241                                 best[1, i] = dist[h];            // info about
                                    max, mid and min of this ID
242                                 i += 1;
243                             }
244                             else if (i >= 3)
245                             {
246                                 break;
247                             }
248                         }
249                         p -= 1;
250                     }
251                     break;
252                 default:
253                     Console.WriteLine("Default case");
```

```
                    break;
            }
            foreach (int ee in best)
            {
                Console.WriteLine(ee);
            }
            return best;
        }
```

11.12 Appendix L: Initialization procedure 7(C#)

```
//Methode to compute the position based on the ID in [cm], Output in [mm]
        public static Position Trilateration(Position point1, Position point2,
            Position point3, int r1t, int r2t, int r3t, int bestr1, int bestr2,
            int bestr3)
        {
            //double[] dist = new double[] { 10.5, 10.0, 9.5, 9.0, 8.0, 6.0,
                5.0, 4.0 }; // FH paper
            double[,] dist = new double[3, 8] { { 14, 9.75, 9.0, 8.0, 7.0, 6.0,
                3.5, 2.75 },
                                                { 5.0, 5.1, 5.3, 5.5, 5.8, 4.0,
                                                    3.5, 2.75 },
                                                { 5.0, 4.7, 4.5, 4.3, 4.2, 4.0,
                                                    3.5, 2.75} }; //
                                                    Approximation of our
                                                    measurements

                        Position resultPose = new Position();
                        PositionD ex = new PositionD();
                        PositionD ey = new PositionD();
                        PositionD aux = new PositionD();
                        PositionD auy = new PositionD();
                        PositionD aux2 = new PositionD();
                        double r1;
                        double r2;
                        double r3;
                        r1 = dist[bestr1, r1t];
                        r2 = dist[bestr2, r2t];
                        r3 = dist[bestr3, r3t];

            // For testing purpose
            //Console.WriteLine("1st radius " + r1);
            //Console.WriteLine("2nd radius " + r2);
            //Console.WriteLine("3rd radius " + r3);

            //Console.WriteLine("1st point " + point1.X + " " + point1.Y);
            //Console.WriteLine("2nd point " + point2.X + " " + point2.Y);
            //Console.WriteLine("3rd point " + point3.X + " " + point3.Y);

            //unit vector in a direction from point1 to point 2
            double p2p1Distance = Math.Pow(Math.Pow(point2.X - point1.X, 2) +
                Math.Pow(point2.Y - point1.Y, 2), 0.5);
```

```
33              ex.X = (point2.X - point1.X) / p2p1Distance;
34              ex.Y = (point2.Y - point1.Y) / p2p1Distance;
35              aux.X = point3.X - point1.X;
36              aux.Y = point3.Y - point1.Y;
37              //signed magnitude of the x component
38              double i = ex.X * aux.X + ex.Y * aux.Y;
39              //the unit vector in the y direction.
40              aux2.X = point3.X - point1.X - i * ex.X;
41              aux2.Y = point3.Y - point1.Y - i * ex.Y;
42              ey.X = aux2.X / Norm(aux2);
43              ey.Y = aux2.Y / Norm(aux2);
44              //the signed magnitude of the y component
45              double j = ey.X * aux.X + ey.Y * aux.Y;
46              //coordinates
47              double x = (Math.Pow(r1, 2) - Math.Pow(r2, 2) + Math.Pow(
                    p2p1Distance, 2)) / (2 * p2p1Distance);
48              double y = (Math.Pow(r1, 2) - Math.Pow(r3, 2) + Math.Pow(i, 2) +
                    Math.Pow(j, 2)) / (2 * j) - i * (x / j);
49              //result coordinates
50              double finalX = 10 * (point1.X + x * ex.X + y * ey.X);
51              double finalY = 10 * (point1.Y + x * ex.Y + y * ey.Y);
52              resultPose.X = (int)(finalX);
53              resultPose.Y = (int)(finalY);
54
55              return resultPose;
56          }
```

11.13 Appendix M: WeMos D1 Mini (Arduino)

```
#include <SoftwareSerial.h>
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>
#include <string.h>
#include <ArduinoJson.h>


//how many clients should be able to telnet to this ESP8266
#define MAX_SRV_CLIENTS 2

// Definig Wifi address, password, host and port
//const char* ssid     = "UPC113C854 A.Abouelkhair";   // write SSID between "( here )"
//const char* password = "rnU3cu6dzpkA";               // write Password between "(case sensitive)"
//const char* ssid     = "iPhone";                   // write SSID between "( here )"
//const char* password = "boody123";                 // write Password between "(case sensitive)"
const char* ssid  = "iRobot";                       // write SSID between "( here )"
const char* password = "nopipes123";                // write Password between "( case sensitive)"
//const char* ssid  = "DORTMUND-DEMO-AP";                     // write SSID between "( here )"
//const char* password = "R0b0tn1K";                // write Password between "( case sensitive)"

// Starting Wifi Server and client with Port 8883
WiFiServer server(8883);
WiFiClient serverClients[MAX_SRV_CLIENTS];

// Variables Declaration
unsigned long previousMillis = 0;
const long interval = 10;
unsigned long currentTime;

/* RFID Intialization */
SoftwareSerial RFID(14, 12, false, 256);    //RX,TX = D5,D6 (Wemos UART1)



void setup() {
  // put your setup code here, to run once:

  /* Beginning Serial Communication with RFID with baud rate 115200 */
  RFID.begin(115200);
//  delay(10);
```

```
42    // Beginning Serial Communication with baud rate 115200
43    Serial.begin(115200);
44  //  delay(10);
45    Serial.println();
46
47
48    Serial.println();
49    Serial.print("Connecting to ");
50    Serial.println(ssid);
51
52    // Time is used so the device does not stuck in
53    // connecting to Wifi forever
54    currentTime = millis();
55    unsigned long previousTime = currentTime;
56    while (WiFi.status() != WL_CONNECTED) {
57      delay(500);
58      Serial.print(".");
59      currentTime = millis();
60      if ((currentTime - previousTime) > 12000) {
61        break;
62      }
63    }
64
65
66    WiFi.mode(WIFI_STA);
67    WiFi.begin(ssid, password);
68    uint8_t i = 0;
69    while (WiFi.status() != WL_CONNECTED && i++ < 20) delay(500);
70    if (i == 21) {
71      Serial.print("Could not connect to"); Serial.println(ssid);
72      while (1) delay(500);
73    }
74    server.begin();
75    server.setNoDelay(true);
76
77
78    //   Getting the MAC address and saving it
79    if (WiFi.status() == WL_CONNECTED) {
80      Serial.println("");
81      Serial.println("WiFi connected");
82      Serial.print("IP address: ");
83      Serial.println(WiFi.localIP());
84      Serial.println("Port: 8883");
85    }
86
87    /* AUTO Send AT Command to the RFID */
88    RFID.write("AT+Scan=OFF\r");
89  //  RFID.write("AT+Scan=AC,RSSI\r");
90    RFID.write("AT+Scan=AC,RSSI\r");
91
92  //    delay(10);
```

```
93
94
95  }
96
97
98  void loop() {
99    // put your main code here, to run repeatedly
100
101 //  while(RFID.available())
102 //  {
103 //     char read= RFID.read();
104 //     Serial.print(read);
105 //     delay(10);
106 //    }
107
108   /*Send commands Through Wifi*/
109   sendCommandsWiFi();
110
111   /* send AT Commands to RFID through serial monitor */
112   //Example: AT+Scan=AC,RSSI            "Without /r"
113   // Comment the next command if you are using AUTO command send
114   sendCommandsRFID();
115
116
117 }
118
119
120
121 void sendCommandsWiFi()
122 {
123
124   unsigned long currentMillis = millis();
125   uint8_t i;
126   //check if there are any new clients
127   if (server.hasClient()) {
128     for (i = 0; i < MAX_SRV_CLIENTS; i++) {
129       //find free/disconnected spot
130       if (!serverClients[i] || !serverClients[i].connected()) {
131         if (serverClients[i]) serverClients[i].stop();
132         serverClients[i] = server.available();
133         continue;
134       }
135     }
136     //no free/disconnected spot so reject
137     WiFiClient serverClient = server.available();
138     serverClient.stop();
139   }
140
141   //do every 2ms edit from time interval in Declaration
142 //  if (currentMillis - previousMillis >= interval)    //
143 //  {  //
```

```
144 //    previousMillis = currentMillis;  //
145     for (i = 0; i < MAX_SRV_CLIENTS; i++)
146     {
147       if (serverClients[i] && serverClients[i].connected())
148       {
149 //         delay(20);
150 //         RFID.write("AT+A\r");
151         while(RFID.available()>0)
152         {
153           char read = RFID.read();
154           serverClients[i].print(read);
155 //            delay(1);   //
156           }
157       }
158     }
159 //  }  //
160 
161 }
162 
163 
164 void sendCommandsRFID()
165 {
166     /* send AT Commands to RFID through serial monitor */
167     //Example: AT+Scan=AC,RSSI             "Without /r"
168   while (Serial.available() > 0) {
169     RFID.write(Serial.read());
170   }
171 
172   }
```

11.14 Appendix N: Communication Outlay

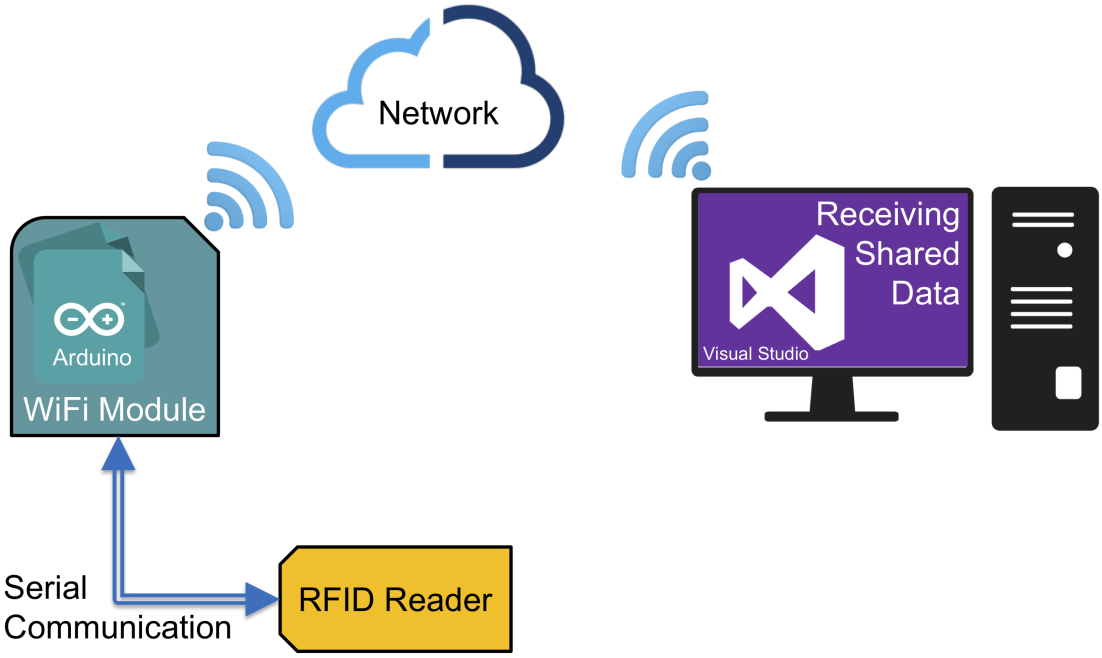


Figure 44: Communication Outlay