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(4511328)
Medhini Rajagopal Balamurugan(4511328)
Stefan Rottstegge(191455)
Stephan Vette(198907)

Supervisors:

Afaq Ahmad
Marina Rantanen-Modéer

Abstract

Summary. Note that the abstract heading is unnumbered, it should remain so. To remove heading numbering use:

```
\section*{}
```

Contents

List of Figures

List of Tables

1 Introduction

Add your name to the file name

2 Pipeless Plant

2.1 Existing setup

2.2 Problems with the Existing Setup

..
zb

- Fish eye
- Sunlight..

3 Selection Process

About the 4 techniques..

3.1 Triangulation[1]

Summary

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. 1 [1].

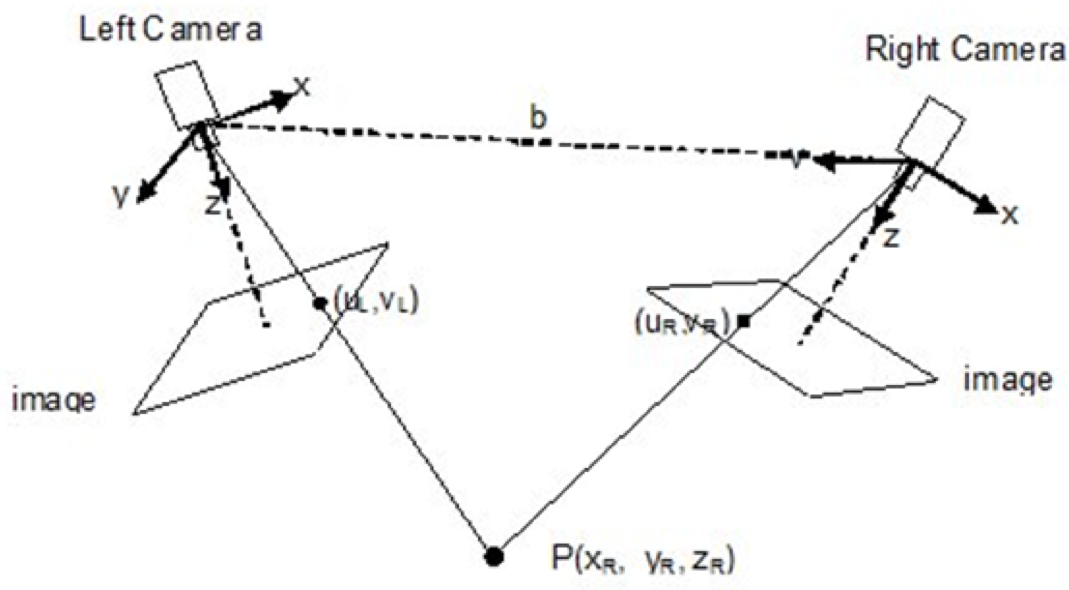


Figure 1: Passive triangulation setup with two cameras

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 pipeless plant it would be an AGV is found, the

[1]Stefan

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. 2 [2].

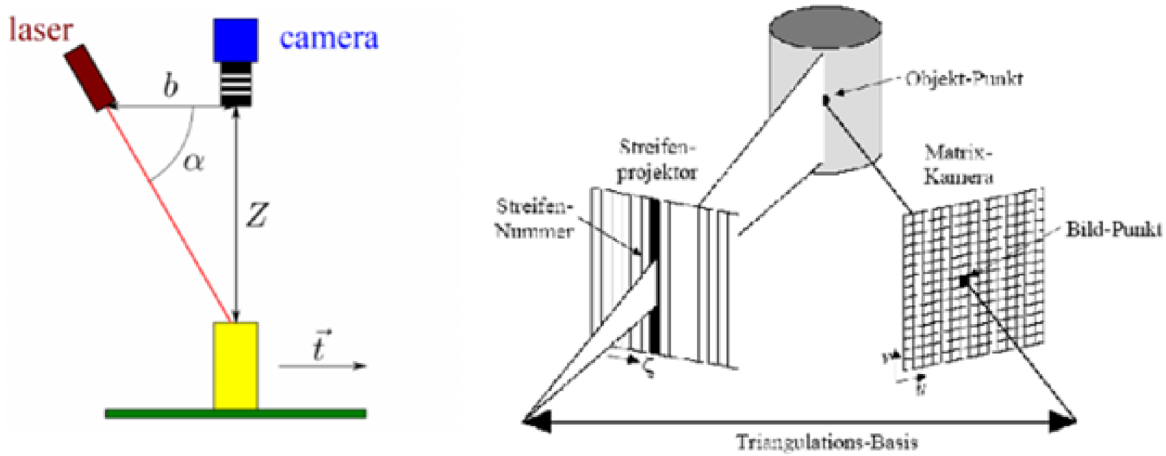


Figure 2: Active triangulation

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. 3.

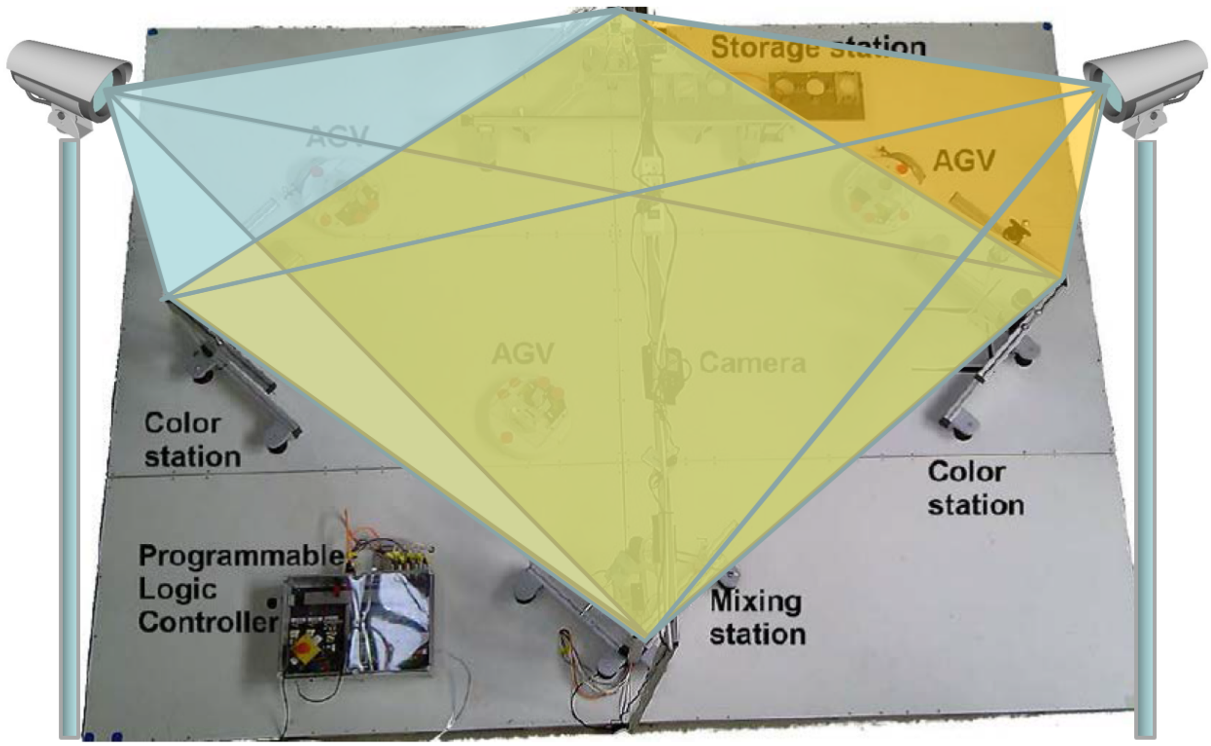


Figure 3: Implementation of passive triangulation

The left and right camera are sequentially taking pictures which are transmitted to the plants computer where the image processing takes place.

Pro and con

Based on the made research, two tables with advantages and disadvantages of the two RFID 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

Implementation

Pro and con

3.3 RFID[2]

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 [3], 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. 4, consists of a reader, which is connected to an antenna and a computer and a passive tag.

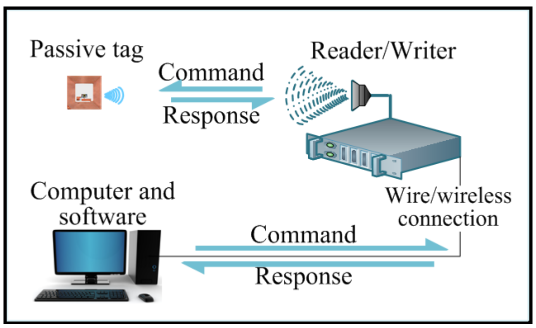


Figure 4: Passive RFID System

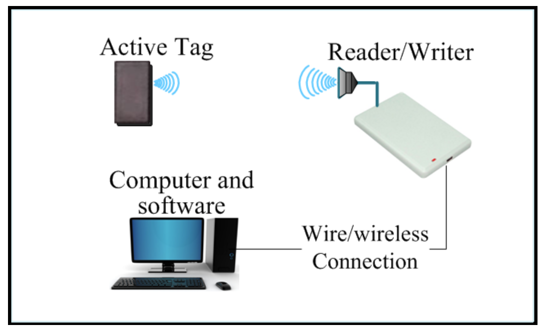


Figure 5: Active RFID System

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.5) 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 3 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.

[2] 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 3: 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 4: 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 5: Pro and cons passive RFID system

3.4 Map-Based Localization

Summary

Implementation

Pro and con

..

4 Theoretical Background

4.1 Radio Frequency Identification (Abdul)

4.2 Trilateration[3]

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 [4]. 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.

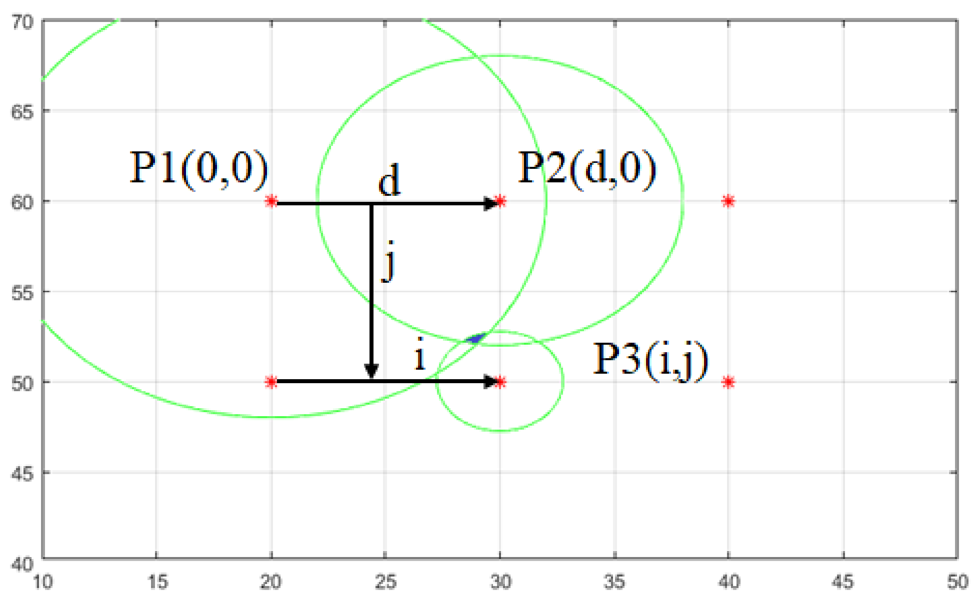


Figure 6: Overview Trilateration

With known positions of the center of the circles d, i and j can be computed in the following

[3]Stephan

way[4]:

$$d = |P_2 - P_1| \tag{2}$$
$$e_x = \frac{1}{d}(P_2 - P_1) \tag{3}$$
$$a_x = P_3 - P_1 \tag{4}$$
$$i = e_x \cdot a_x \tag{5}$$
$$a_y = (P_3 - P_1) - i * e_x \tag{6}$$
$$e_y = \frac{a_y}{|a_y|} \tag{7}$$
$$j = e_y \cdot a_x \tag{8}$$

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 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[4]

5.1 RFID reader and antenna[5]

The RFID reader from KTS Systeme (see fig.7) 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.

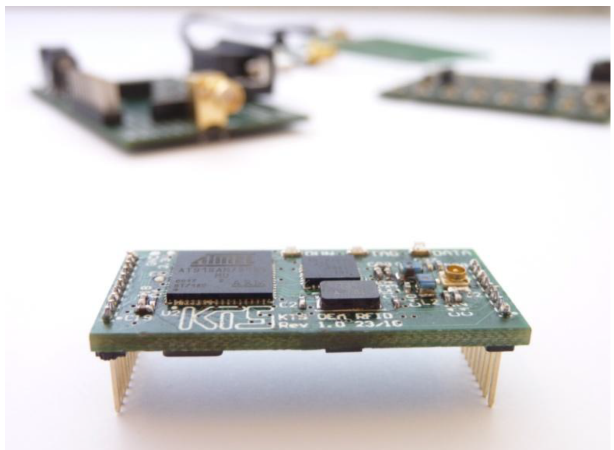


Figure 7: RFID reader KTS Systeme RFIDM1356-001

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][5].
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

[4]Stephan and Abdul
[5]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. 8) 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 8: 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 ?

5.3 Wifi modul (Abdul ?)

5.4 HW setup?!? (Abdul ?)

6 Simulation[6]

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 [6] 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.

[6] 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. 6 and fig. 9 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 6: Relation between RSSI and distance antenna to tag (data)

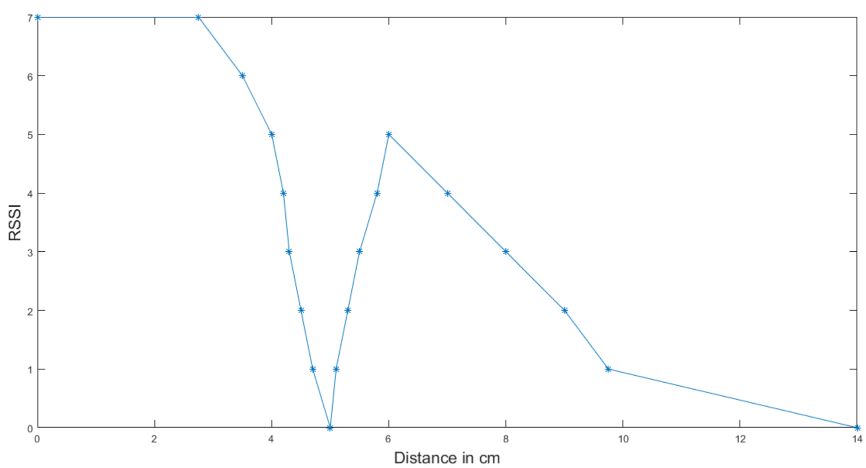


Figure 9: Relation between RSSI and distance antenna to tag

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 7: Results Simulation

As can be seen from tbl. 7, 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 (Abdul and/or Stefan)

7.2 Initialization procedure [7]

In the start-up phase, before running the pipeless plant with its AGVs the correct position and orientation of each and every vehicle is 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. 10 four possible AGV positions with one common antenna position are pointed out.

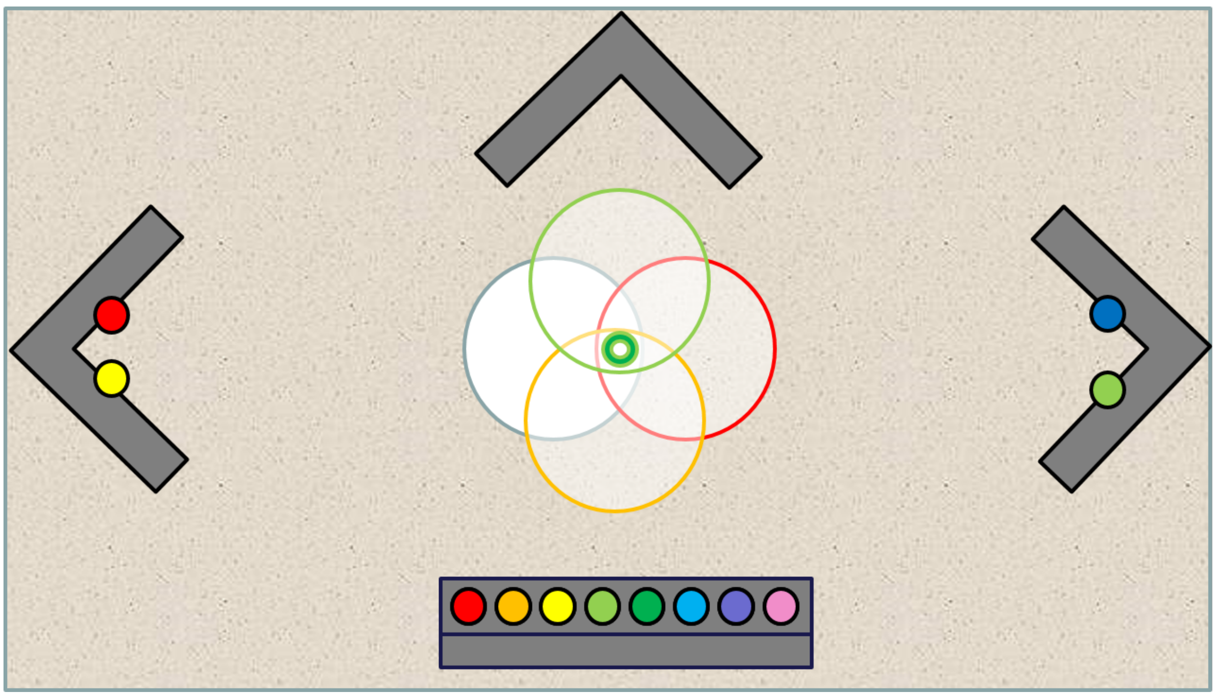


Figure 10: Different possible positions for one antenna position

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. 11.

[7] Stephan and Stefan

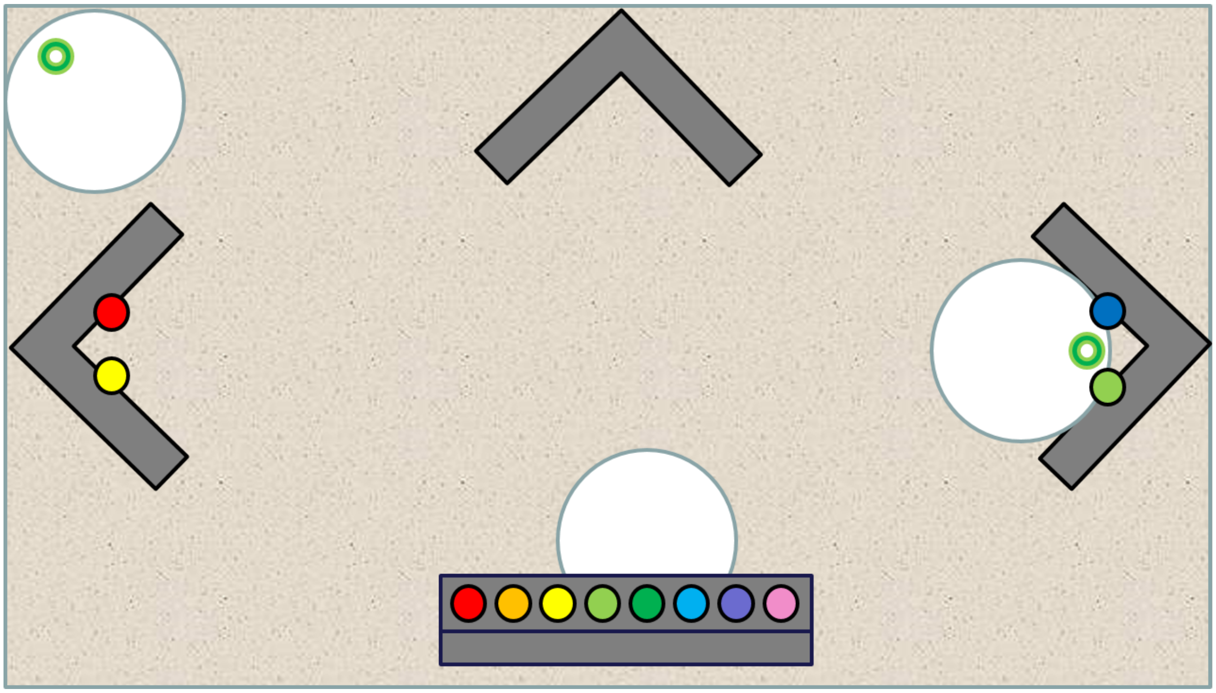


Figure 11: Possible hazards/obstacles

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. 12 illustrates a sequential flow chart which describes the movement and data processing during the initialization procedure.

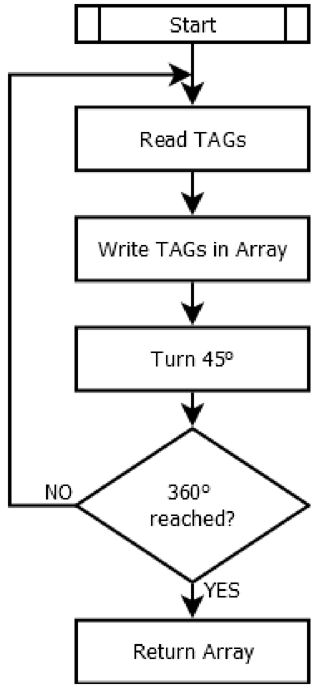


Figure 12: Flow Chart: Initial procedure 360° turn

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. 13.

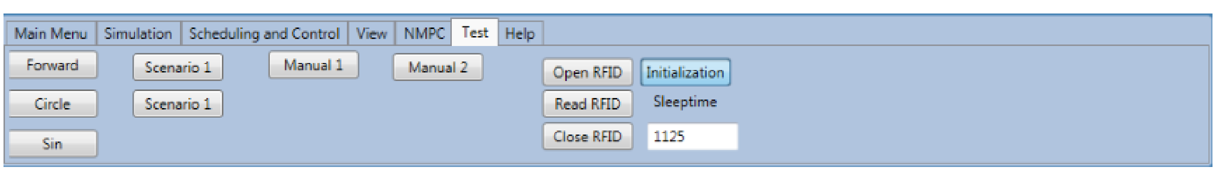


Figure 13: Test environment in GUI

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[5]. With the included timestamp for every measurement a delay of minimum 30ms between each and every 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.

$$Columns = max.no.oftags * 2 = 7 * 2 = 14 \tag{14}$$
$$\tag{15}$$

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 8.

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 8: Filled array after 360° turn

7.2.1 Recording and filtering data [8]

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 and every string contains, the structure itselve 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 if the current reading is either a complete or incomplete one. 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.

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

Table 9: String preperation

7.2.2 Analysing data[9]

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.

[8]Stefan
[9]Stefan

In terms to get the adequate sets of measurment points the array is analyzed row by row. The stepwise workflow is vizialized in fig. 14.

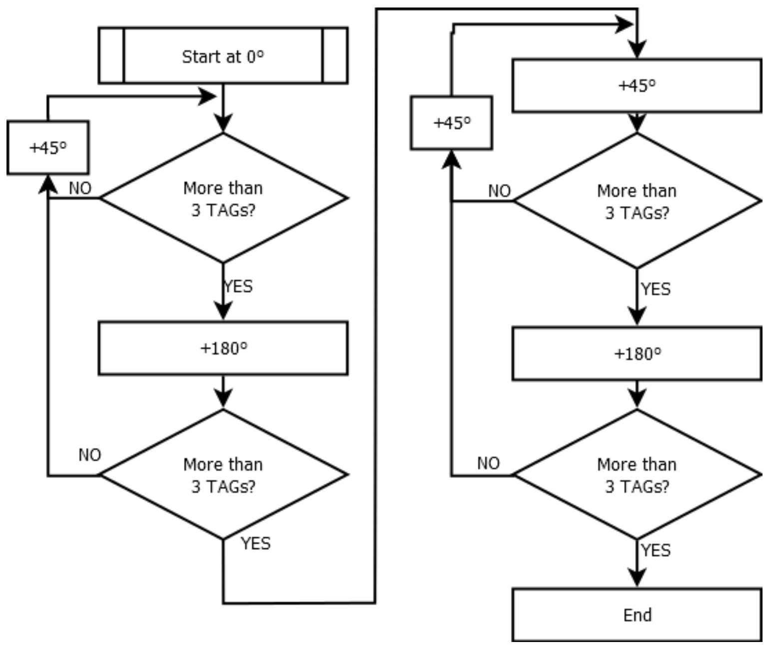


Figure 14: Flow Chart: Analizing measurment points

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 [10]

In a first step the multiple occurring data points (see tbl.6) are divided into three groups (max, middle and min) where max means the maximal possible distance related to one RSSI and so on.

[10]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. 15:

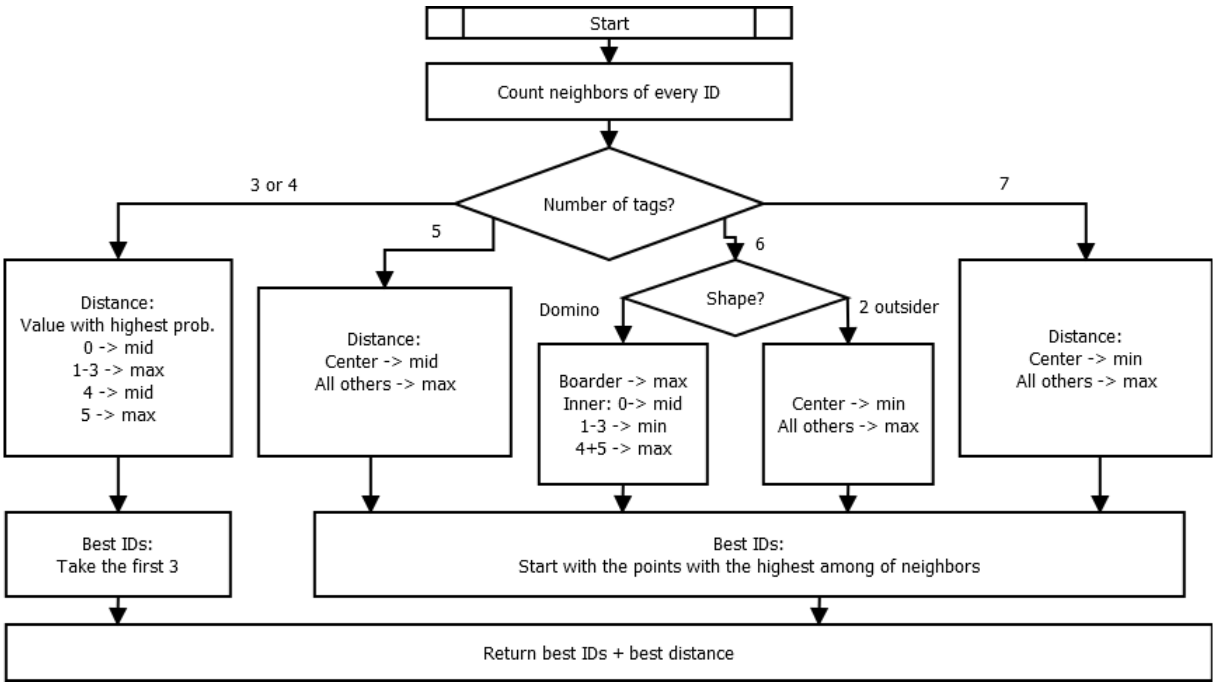


Figure 15: Flow Chart: Selection of correct distance and most proper IDs

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. 10.

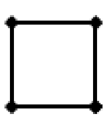
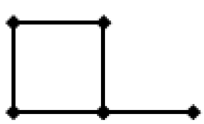
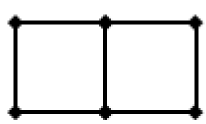
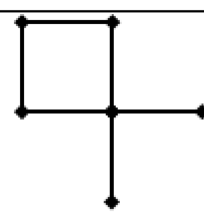
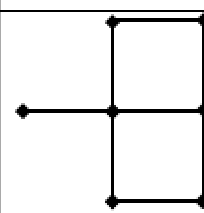
Number of detected tags	4	5	6 (Domino)	6 (2 alone)	7
Unique shapes					

Table 10: Possible shapes of pattern

Going back to the flow chart fig.15 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 [11]

As mentioned in chapter 7.2, 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.16 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

[11] Stephan

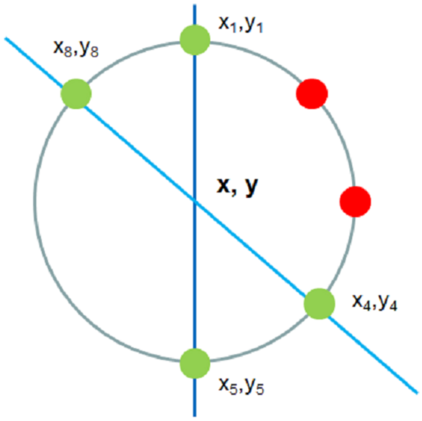


Figure 16: Computing the center of the robot

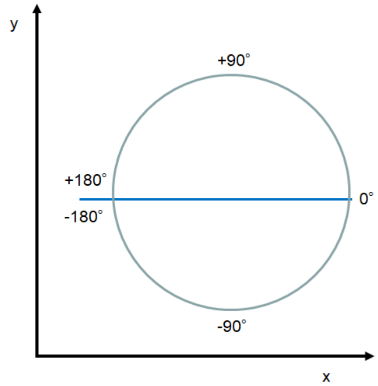


Figure 17: Orientation of robot in absolute angle

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.17.
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[12]

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.18 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.11.

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.

[12]Stephan

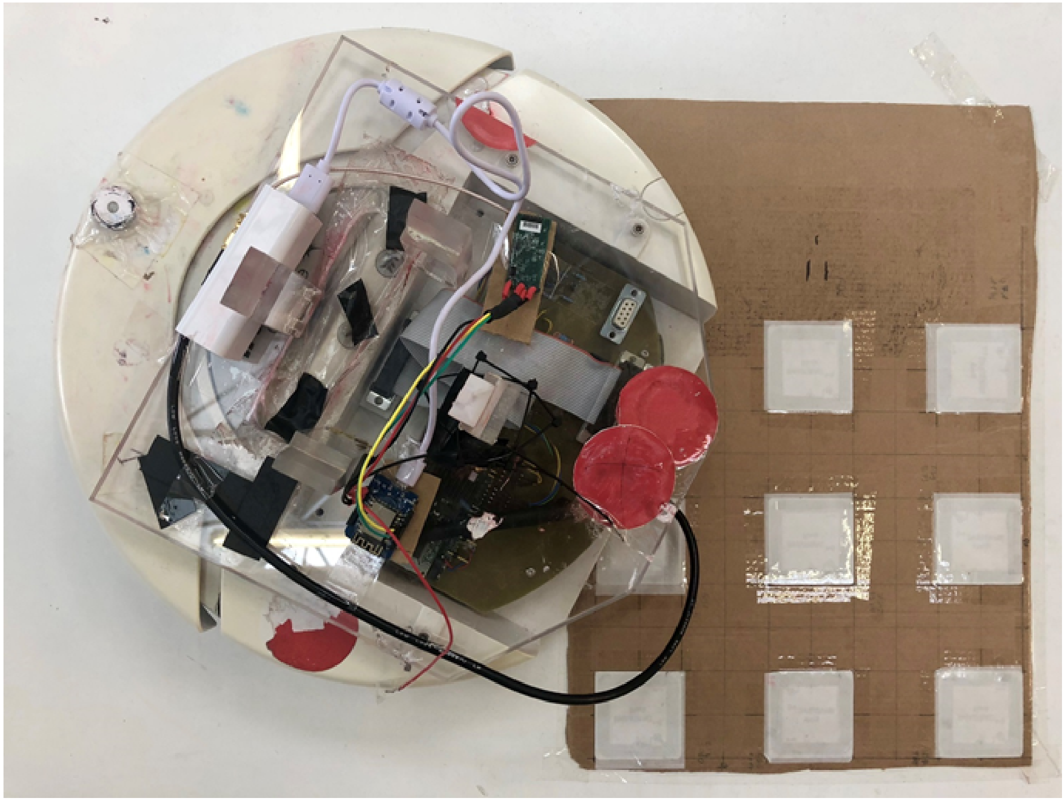


Figure 18: testing setup for initialization procedure

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 11: 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[13]

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.19 and fig.20 illustrate the actual measurement results and the desired position in x- and y-direction.

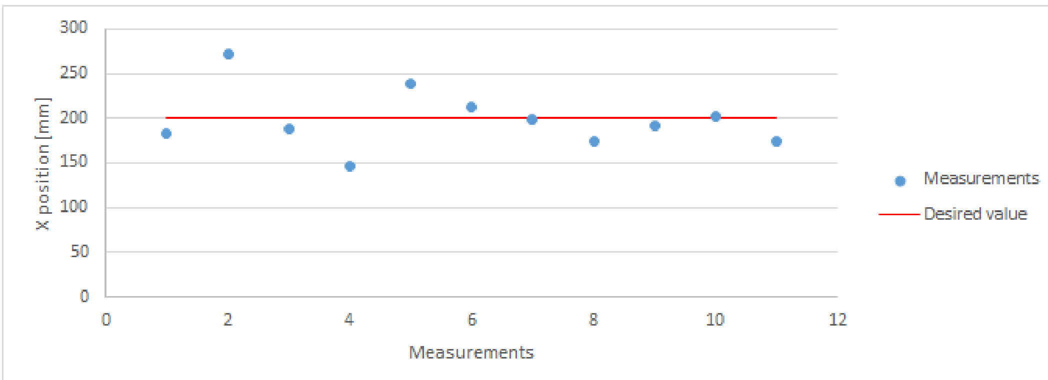


Figure 19: Estimated position in x-direction

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.19, 20 and 21).
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.

[13]Stephan

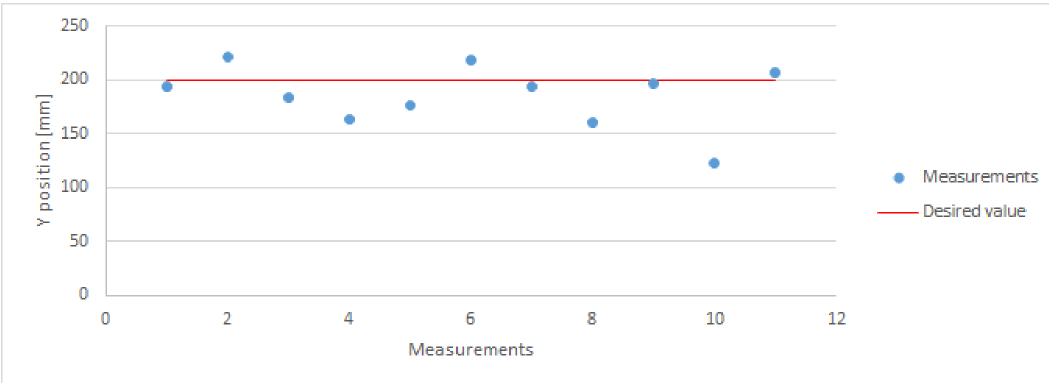


Figure 20: Estimated position in y-direction

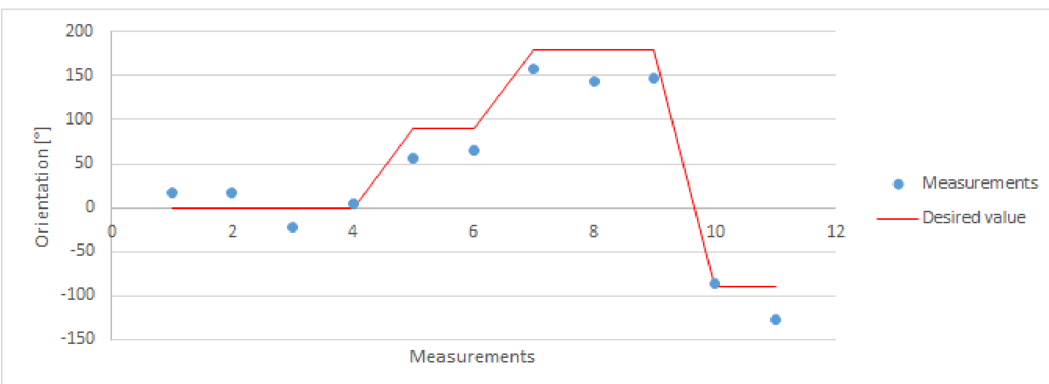


Figure 21: Estimated orientation

8 Conclusion[14]

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.

[14]Stephan and Stefan

9 Future Work[15]

After a proof-of-concept for an RFID based localizatin system has been build and a first demonstration set-up has been build 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.

[15] Stefan

10 References

References

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

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

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

[4] 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.

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

[6] 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
```

```
    error('Length of platform not dividable by distance between tags');
end

%% Computing position of antenna
numTagsX = (l1-2*d2)/d1 +1;
numTagsY = (l2-2*d2)/d1 +1;
numTags = numTagsX * numTagsY;
antPos = robStart + d3 * [cos(gamma1), sin(gamma1)];

%% Display the setup, write important information into a seperate txt file
d1_str = num2str(d1);
l1_str = num2str(l1);
l2_str = num2str(l2);
numTags_str = num2str(numTags);

msg0 = ['Your plane is ',l1_str,'cm x ',l2_str,'cm.'];
msg1 = ['You chose a distance of ',d1_str, 'cm and need ', numTags_str,' Tags!'];
disp(msg0);
disp(msg1);
nameTxt = ['NumTags',measuementeNumber,'.txt'];
fileNumTags = fopen(nameTxt,'w');
fprintf(fileNumTags,'%6d\n',numTags);   % Write the number of tags in file
fprintf(fileNumTags,'%6d\n',l1);        % Write the size of the plant in file
fprintf(fileNumTags,'%6.4f\n',gamma1);      % Write the starting angle
fprintf(fileNumTags,'%6d\n',robStart(1));      % Write the starting pos
fprintf(fileNumTags,'%6d\n',robStart(2));      % Write the starting pos
fclose(fileNumTags);


%% Drawing environment
figure(1)
x1 = [0 l1 l1 0 0];
y1 = [0 0 l2 l2 0];
plot(x1, y1,'LineWidth',2)
xlim([-5 (l1+5)]);
ylim([-5 (l2+5)]);
hold on

% Position of the tags
ID = 1:numTags;
[Tagx,Tagy] = meshgrid(d2:d1:l1-d2,d2:d1:l2-d2);
plot(Tagx,Tagy,'r*')
% Circles
radiipl = ones(numTagsX,1)*r1;
for k=1:numTagsX
    tempx = Tagx(1:end,k);
    tempy = Tagy(1:end,k);
    temppos = horzcat(tempx,tempy);
    viscircles(temppos,radiipl,'Color','k','LineStyle',':','LineWidth',0.25);
end
robX = robStart(1);
robY = robStart(2);
plot(robX,robY,'bO','LineWidth',3);
plot(robX,robY,'r:');
viscircles([robX,robY],r3,'Color','k','LineWidth',0.25);
plot(antPos(1),antPos(2),'bs');
xlabel('Length platform in cm')
ylabel('Width platform in cm')
title({'Position and reading range of tags';'Start-, endpoint and path of the robot'});
```

```
hold off
pause(1)

%% Animation and loggin
xUpdateAnt = antPos(1);
yUpdateAnt = antPos(2);
deltaR = deg2rad(angle1);        % A new measurement after every XX°
% Txt file name
name = ['Meas_StartingProc_like_reader_real_data',measuementeNumber,'.txt'];
fileID = fopen(name,'w');

% Data stored in variables
dataRSSI = zeros(8,numTags);
streamDataRSSI = zeros(1,numTags);
streamDataRSSIold = zeros(1,numTags);
timeStep = 1;    % current measurement step

% antPos = robStart + d3 * [cos(gamma1), sin(gamma1)];
figure(2)
for l=0:360/angle1
    deltaR_temp = deltaR * l;
    xUpdateAnt = robStart(1) + d3 * cos(gamma1 + deltaR_temp);
    yUpdateAnt = robStart(2) + d3 * sin(gamma1 + deltaR_temp);
    plot(x1, y1,'LineWidth',2)
    hold on
    xlim([-5 (l1+5)]);
    ylim([-5 (l2+5)]);
    [Tagx,Tagy] = meshgrid(d2:d1:l1-d2,d2:d1:l2-d2);
    plot(Tagx,Tagy,'r*')
    plot(robX,robY,'bO','LineWidth',1);
    plot(robX,robY,'r:');
    plot(xUpdateAnt,yUpdateAnt,'bs');
    xlim([-5 (l1+5)]);
    ylim([-5 (l2+5)]);
    viscircles([robX,robY],r3,'Color','b','LineWidth',0.5);
        for k=1:numTagsX
            tempx = Tagx(1:end,k);
            tempy = Tagy(1:end,k);
            temppos = horzcat(tempx,tempy);
            viscircles(temppos,radiipl,'Color','k','LineStyle',':','LineWidth',0.25);
        end
    hold off

    % Creating measurements
    antPosnew=[xUpdateAnt,yUpdateAnt];
    for m = 1:numTags                        % m = current number of tag
        m_str = num2str(m);
        tempTag=[Tagx(m),Tagy(m)];
        tempD = pdist([antPosnew; tempTag] ,'euclidean');

        % Display if tag is in range or not
        if tempD > r1
                streamDataRSSI(m) = 0;
                if (streamDataRSSI(m) ~= streamDataRSSIold(m)) && mode == 2
                    if mode_hex == 1
                        fprintf(fileID,'%d %s%s%s%d%s\n',l*angle1,text2,dec2hex(m, 16),
                            text3,k(end),text0);
                    elseif mode_hex == 0
                        fprintf(fileID,'%d %s%d%s%d%s\n',l*angle1,text2,m,text3,k(end),
```

```
                                text0);
                        end
                        fprintf(' %d %d %1d\n',l*angle1,m,'0');
                    end
            elseif tempD <= r1
                    % disp(['Label ',m_str,' in range!!!!!!!!!!!!']);
                    % Relation distance <-> RSSI
                    k_temp = find(r2>=tempD);
                    k = r4(k_temp);
                    dataRSSI(timeStep,m) = k(end);
                    streamDataRSSI(m) = k(end);
                    if (streamDataRSSI(m) ~= streamDataRSSIold(m)) && mode == 2
                        if mode_hex == 1
                            fprintf(fileID,'%d %s%s%s%d%s\n',l*angle1,text2,dec2hex(m, 16),
                                text3,k(end),text0);
                        elseif mode_hex == 0
                            fprintf(fileID,'%d %s%d%s%d%s\n',l*angle1,text2,m,text3,k(end),
                                text0);
                        end
                        fprintf(' %d %d %8d,\n',l*angle1,m,k(end));
                    elseif mode == 1
                        if mode_hex == 1
                            fprintf(fileID,'%d %s%s%s%d%s\n',l*angle1,text2,dec2hex(m, 16),
                                text3,k(end),text0);
                        elseif mode_hex == 0
                            fprintf(fileID,'%d %s%d%s%d%s\n',l*angle1,text2,m,text3,k(end),
                                text0);
                        end
                        fprintf(' %d %d %1d,\n',l*angle1,m,k(end));
                    end
            end
        end
    streamDataRSSIold = streamDataRSSI;
    pause(cycleT/1000)
    timeStep = timeStep + 1;
end
savefig('Figure2.fig');
fclose(fileID);

%% Results
% figure(3)   % plot for the max value of every tag
% dataRSSInoT = reshape(max(dataRSSI),[numTagsX,numTagsY]);
% plot3(Tagx,Tagy,dataRSSInoT,'*');
% xlabel('Length platform in cm')
% ylabel('Width platform in cm')
% title('Max RSSI signal of every tag')

figure(4)   % plot of the RSSI signal which are non zero vs. time
dataRSSIsum = sum(dataRSSI);
IDclear = find(dataRSSIsum ~= 0);
IDstr = string(IDclear);
dataRSSIclear = dataRSSI;
dataRSSIclear( :, all( ~any( dataRSSI ), 1 ) ) = []; % and columns
plot(dataRSSIclear);
xlabel('Measurement points')
ylabel('RSSI')
ylim([0 360/angle1])
legend(IDstr,'FontSize',6);
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();
```

```
            }

            Console.WriteLine("Connecting");
            TcpClient tcpClient = new TcpClient("192.168.0.100", 8883);

            if (tcpClient.Connected)
            {
                Console.WriteLine("Connected to server");
            }

            using (StreamReader STR = new StreamReader(tcpClient.GetStream()))
            {
                string recieve;
                char[] trash = new char[16];
                char[] UID = new char[3];
                char[] RSSI = new char[3];
                long milliseconds, seconds, minutes;
                string UID_, RSSI_, RSSI___;
                string[] array = new string[1];

                List<string> RSSI__;
                int UID_DEC=0;
                int RSSI_int = 0;

                    while ((recieve = STR.ReadLine()) != null && !ct.IsCancellationRequested)
                    {
                        if (ct.IsCancellationRequested)
                        {
                            try
                            {
                                ct.ThrowIfCancellationRequested();
                            }
                            catch (AggregateException e)
                            {
                            }
                        }

                        if (recieve.Contains("+"))
                        {

                        List<string> Worte = recieve.Split(new string[] { "OK", "<\\r>", "\n", "", "SCAN:+UID=", "+RSSI=" }, StringSplitOptions.RemoveEmptyEntries).ToList();
                        string Wort = string.Join("", Worte.ToArray());

                        using (StringReader sr = new StringReader(Wort))
                        {
                            sr.Read(trash, 0, 13);
                            sr.Read(UID, 0, 3);
                            UID_ = new string(UID);
```

```
                            sr.Read(trash, 0, 1);
                            sr.Read(RSSI, 0, 1);
                            RSSI_ = new string(RSSI);
                            try
                            {
                                UID_DEC = Int32.Parse(UID_, System.Globalization.NumberStyles.HexNumber);
                            }
                            catch (Exception e)
                            {
                            }
                        }

                        RSSI__ = RSSI_.Split(new string[] { "," }, StringSplitOptions.RemoveEmptyEntries).ToList();
                        RSSI___ = string.Join("", RSSI__.ToArray());
                        try
                        {
                            RSSI_int = Int32.Parse(RSSI___);
                        }
                        catch (Exception e)
                        {
                        }

                        milliseconds = DateTimeOffset.Now.Millisecond;
                        seconds = DateTimeOffset.Now.Second;
                        minutes = DateTimeOffset.Now.Minute;
                        array[0] = minutes + " " + seconds + " " + milliseconds + " " + UID_DEC + " " + RSSI_int;
                        //File.AppendAllText(AppDomain.CurrentDomain.BaseDirectory + "\\pythonfiles\\python_1robot\\RFID_Data.log", minutes + " " + seconds + " " + milliseconds + "\t UID: " + UID_ + " RSSI: " + RSSI___ + "\r");
                        //File.AppendAllText(AppDomain.CurrentDomain.BaseDirectory + "\\pythonfiles\\python_1robot\\RFID_Data_original.log", hour + ":" + minutes + ":" + seconds + ":" + milliseconds + "\t" + recieve + "\r");
                        //Console.WriteLine(minutes + " " + seconds + " " + milliseconds + "\t" + " " + UID_ + "  " + RSSI___);
                        }
                            this.availablearray[0] = array[0];
                        }
                    }
                }



        public void disconnect()
        {
```

```
            TcpClient tcpClient = new TcpClient();
            tcpClient.Connect("192.168.0.100", 8883);
            tcpClient.Close();

        }
}
}
```

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
    {
        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();
                RobotDiscription[] RobotArray = new RobotDiscription[] { Gateway.CTRLModule.getInstance().camCtrl.RobotA, Gateway.CTRLModule.getInstance().camCtrl.RobotB,
                Gateway.CTRLModule.getInstance().camCtrl.RobotC };
                double[][] velocity1 = new double[RobotArray.Length][];
                string[,] Signals = new string[messungen,8];
```

```
                velocity1[currentRobot] = new double[] { 0, 0 }; //Starts the Robot
                Gateway.CTRLModule.getInstance().getRobotRemoteCTRL(RobotAssingment[currentRobot]).forward(velocity1[currentRobot], 0, 0, 0); //Sends velocity to Robot
                //Opening a new Task which works in the background to read data from RFID Antenna
                Task t = Task.Factory.StartNew(() => initial.reading(token));
                Thread.Sleep(1000);
                for (int i = 0; i < 8; i++)      //9 Because of 8 measurements every 45 degree
                {
                    for (int j = 0; j < messungen; j++)     //in this for loop we find all the reachable TAGs
                    {
                        Signals[j, i] = initial.availablearray[0];
                        Thread.Sleep(100);
                    }
                    velocity1[currentRobot] = new double[] { 100, -100 }; //Starts the Robot
                    Gateway.CTRLModule.getInstance().getRobotRemoteCTRL(RobotAssingment[currentRobot]).forward(velocity1[currentRobot], 0, 0, 0); //Sends velocity to Robot
                    Thread.Sleep(time); //time the robot needs for a 45 degree turn
                    velocity1[currentRobot] = new double[] { 0, 0 };  //Stops the Robot
                    Gateway.CTRLModule.getInstance().getRobotRemoteCTRL(RobotAssingment[currentRobot]).forward(velocity1[currentRobot], 0, 0, 0); //Sends velocity to Robot
                }

                tokenSource.Cancel();   //close the reading Thread
                try
                {
                    Task.WaitAll(t);
                }
                catch (AggregateException e)
                {
                }
                finally
                {
                    tokenSource.Dispose();
                }
                    Console.WriteLine("END\r\r");

            Array[] Liste = new Array[8];   //List of arrays each array in the array contains the data of a special position (45Â°, 90Â°,...)
            string[,] Init_array = new string[8, 14];   //Array filled with signal strengthes and ID of every degree position
            string temp_ID="begin", temp_RSSI;  //Substrings of Data
```

```
            int counter;    //Counter for the row in the Init_Array
            Console.WriteLine("");

            for (int j = 0; j < 8; j++)
            {
                counter = 0;
                for (int i = 0; i < messungen; i++)
                {
                    try
                    {
                        temp_ID = Signals[i, j].Substring(Signals[i, j].Length -
                             6, 4);   //seperatiion of UID in the string
                        if (temp_ID == "2788")
                        {
                            temp_ID = "1";
                        }
                        if (temp_ID == "1414")
                        {
                            temp_ID = "2";
                        }
                        if (temp_ID == "3060")
                        {
                            temp_ID = "3";
                        }
                        if (temp_ID == "1673")
                        {
                            temp_ID = "4";
                        }
                        if (temp_ID == "1925" || temp_ID == "1025")
                        {
                            temp_ID = "5";
                        }
                        if (temp_ID == "1681")
                        {
                            temp_ID = "6";
                        }
                        if (temp_ID == "1146")
                        {
                            temp_ID = "7";
                        }
                        if (temp_ID == "2780")
                        {
                            temp_ID = "8";
                        }
                        if (temp_ID == "1933")
                        {
                            temp_ID = "9";
                        }
                    }
                    catch (AggregateException e)
                    {
```

```
                    Console.WriteLine("Array incomplete");
                }

                temp_RSSI = Signals[i, j].Substring(Signals[i, j].Length - 1, 1);     //seperation of RSSI in the string
                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
                {
                    //Filling Init_Array
                    Init_array[j, counter] = temp_ID;
                    Init_array[j, counter + 7] = temp_RSSI;
                    counter++;
                }
            }
        }

            int rowLength = Init_array.GetLength(0);
            int colLength = Init_array.GetLength(1);
            string str;
            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" + "|";
            System.Console.WriteLine(headline);

            for (int k = 0; k < rowLength; k++)
            {
                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] + "    |";
                System.Console.WriteLine(str);
            }

            // Solver
            // Different Positions
            Position Starting = new Position();
            Position Antenna1 = new Position();
            Position Antenna2 = new Position();
            Position Antenna3 = new Position();
            Position Antenna4 = new Position();

            //Initialization for Position estimation
```

```
float m1 = 0.000f;
float m2 = 0.000f;

float RobStartx_fl = 0.000f;
float RobStarty_fl = 0.000f;

double angle;
double angleTemp;

int null_counter = 0;
int[] check_row = new int[8];
for (int m = 0; m < rowLength; m++)
{
    null_counter = 0;
    for (int n = 0; n < 7; n++)
    {
        if (Init_array[m, n] == null)
        {
            null_counter++;
        }
    }
    check_row[m] = 7 - null_counter;    //Array of elements with the number empty places of each init_array row
    System.Console.WriteLine("The number of elements at " + m * 45 + "Â° is: \t" + check_row[m]);
}
bool solution_found = false;    //true if initialization process is solvable
bool solution1_found = false;   //true if one possible point is found
bool solution2_found = false;   //true if two possible points are found
int count = 0;
int[] solution1 = new int[2];   //Array with the both degree numbers of solution 1
int[] solution2 = new int[2];   //Array with the both degree numbers of solution 2
while (solution_found == false)
{
    while (solution1_found == false)
    {
        if (check_row[count] >= 3)
        {
            if (check_row[count + 4] >= 3)
            {
                solution1[0] = count;
                solution1[1] = count + 4;
                break;
            }
            if (count >= 2)    //if we reach the 180 degree there will be no solution for this
```

```
                                initialization turn
                            {
                                System.Console.WriteLine("NO SOLUTION FOUND!!!");
                                break;
                            }
                        }
                        count++;
                    }
                    System.Console.WriteLine(count);
                    count = count + 1;
                    while (solution2_found == false)
                    {
                        if (check_row[count] >= 3)
                        {
                            if (count >= 8)
                            {
                                Console.WriteLine("Out of Range Exception caused in Array: count");
                            }
                            if (check_row[count + 4] >= 3)
                            {
                                solution2[0] = count;
                                solution2[1] = count + 4;
                                solution_found = true;
                                break;
                            }
                            if (count >= 3)     //if we reach the 180 degree there will be no solution for this initialization turn
                            {
                                System.Console.WriteLine("NO SOLUTION FOUND!!!");
                                break;
                            }
                            else
                            {
                                //count = count - 1;
                                break;
                            }
                        }
                    }
                    System.Console.WriteLine("Solution No. 1 found at: " + solution1[0] * 45 + " degree -- " + solution1[1] * 45 + " degree");
                    System.Console.WriteLine("Solution No. 2 found at: " + solution2[0] * 45 + " degree -- " + solution2[1] * 45 + " degree");
                }

                // Providing the distance with the highest probability
```

```
        // Input:   # of tags, all IDs of the tags
        // Output:  best fitting IDs (e.g.[3 4 5] if 3rd, 4th and 5th are best ones)
        //          the correct distance <-> RSSI signal (e.g.[2 1 3] for middle, max and min)
        int[,] best_arr1 = new int[2, 3];
        int[,] best_arr2 = new int[2, 3];
        int[,] best_arr3 = new int[2, 3];
        int[,] best_arr4 = new int[2, 3];

        int[] temp_input1 = new int[7];
        int[] temp_input2 = new int[7];
        int[] temp_input3 = new int[7];
        int[] temp_input4 = new int[7];

        int[] temp_inputRSSI1 = new int[7];
        int[] temp_inputRSSI2 = new int[7];
        int[] temp_inputRSSI3 = new int[7];
        int[] temp_inputRSSI4 = new int[7];

        for (int i = 0; i < 8; i++)
        {
            for (int j = 0; j < 14; j++)
            {
                if (Init_array[i, j] == null)
                {
                    Init_array[i, j] = "0";
                }
            }
        }

        for (int m = 0; m < 7; m++)
        {
            temp_input1[m] = Int32.Parse(Init_array[solution1[0], m]);
            temp_input2[m] = Int32.Parse(Init_array[solution1[1], m]);
            temp_input3[m] = Int32.Parse(Init_array[solution2[0], m]);
            temp_input4[m] = Int32.Parse(Init_array[solution2[1], m]);

            temp_inputRSSI1[m] = Int32.Parse(Init_array[solution1[0], m + 7]);
            temp_inputRSSI2[m] = Int32.Parse(Init_array[solution1[1], m + 7]);
            temp_inputRSSI3[m] = Int32.Parse(Init_array[solution2[0], m + 7]);
            temp_inputRSSI4[m] = Int32.Parse(Init_array[solution2[1], m + 7]);
        }

        best_arr1 = CorrectID_Distance(temp_input1, temp_inputRSSI1, check_row[solution1[0]]);
        best_arr2 = CorrectID_Distance(temp_input2, temp_inputRSSI2,
```

```
            check_row[solution1[1]]);
        best_arr3 = CorrectID_Distance(temp_input3, temp_inputRSSI3,
            check_row[solution2[0]]);
        best_arr4 = CorrectID_Distance(temp_input4, temp_inputRSSI4,
            check_row[solution2[1]]);

        // Position of the antennae
        Antenna1 = Trilateration(IDtoPOS(Int32.Parse(Init_array[
            solution1[0], best_arr1[0, 0]])), IDtoPOS(Int32.Parse(
            Init_array[solution1[0], best_arr1[0, 1]])),
                                IDtoPOS(Int32.Parse(Init_array[solution1
                                    [0], best_arr1[0, 2]])), Int32.Parse
                                    (Init_array[solution1[0], best_arr1
                                    [0, 0] + 7]),
                                Int32.Parse(Init_array[solution1[0],
                                    best_arr1[0, 1] + 7]), Int32.Parse(
                                    Init_array[solution1[0], best_arr1
                                    [0, 2] + 7]),
                                best_arr1[1, 0], best_arr1[1, 1],
                                    best_arr1[1, 2]);

        Antenna2 = Trilateration(IDtoPOS(Int32.Parse(Init_array[
            solution1[1], best_arr2[0, 0]])), IDtoPOS(Int32.Parse(
            Init_array[solution1[1], best_arr2[0, 1]])),
                                IDtoPOS(Int32.Parse(Init_array[solution1
                                    [1], best_arr2[0, 2]])), Int32.Parse
                                    (Init_array[solution1[1], best_arr2
                                    [0, 0] + 7]),
                                Int32.Parse(Init_array[solution1[1],
                                    best_arr2[0, 1] + 7]), Int32.Parse(
                                    Init_array[solution1[1], best_arr2
                                    [0, 2] + 7]),
                                best_arr2[1, 0], best_arr2[1, 1],
                                    best_arr2[1, 2]);

        Antenna3 = Trilateration(IDtoPOS(Int32.Parse(Init_array[
            solution2[0], best_arr3[0, 0]])), IDtoPOS(Int32.Parse(
            Init_array[solution2[0], best_arr3[0, 1]])),
                                IDtoPOS(Int32.Parse(Init_array[solution2
                                    [0], best_arr3[0, 2]])), Int32.Parse
                                    (Init_array[solution2[0], best_arr3
                                    [0, 0] + 7]),
                                Int32.Parse(Init_array[solution2[0],
                                    best_arr3[0, 1] + 7]), Int32.Parse(
                                    Init_array[solution2[0], best_arr3
                                    [0, 2] + 7]),
                                best_arr3[1, 0], best_arr3[1, 1],
                                    best_arr3[1, 2]);

        Antenna4 = Trilateration(IDtoPOS(Int32.Parse(Init_array[
            solution2[1], best_arr4[0, 0]])), IDtoPOS(Int32.Parse(
```

```
                Init_array[solution2[1], best_arr4[0, 1]])),
                                IDtoPOS(Int32.Parse(Init_array[solution2[1], best_arr4[0, 2]])), Int32.Parse(Init_array[solution2[1], best_arr4[0, 0] + 7]),
                                Int32.Parse(Init_array[solution2[1], best_arr4[0, 1] + 7]), Int32.Parse(Init_array[solution2[1], best_arr4[0, 2] + 7]),
                                best_arr4[1, 0], best_arr4[1, 1], best_arr4[1, 2]);

            Console.WriteLine("1st Antenna " + Antenna1.X + " and " + Antenna1.Y);
            Console.WriteLine("2nd Antenna " + Antenna2.X + " and " + Antenna2.Y);
            Console.WriteLine("3rd Antenna " + Antenna3.X + " and " + Antenna3.Y);
            Console.WriteLine("4th Antenna " + Antenna4.X + " and " + Antenna4.Y);

            //Console.ReadKey();
            // Alternative estimation of the centre of the robot + position
            //m1 = ((float)Antenna2.Y - (float)Antenna1.Y) / ((float)Antenna2.X - (float)Antenna1.X);
            //m2 = ((float)Antenna4.Y - (float)Antenna3.Y) / ((float)Antenna4.X - (float)Antenna3.X);
            //RobStartx_fl = (1 / (m1 - m2)) * (m1 * (float)Antenna1.X - m2 * (float)Antenna3.X - (float)Antenna1.Y + (float)Antenna3.Y);
            //RobStarty_fl = m1 * (RobStartx_fl - (float)Antenna1.X) + (float)Antenna1.Y;

            //Starting.X = (int)RobStartx_fl;
            //Starting.Y = (int)RobStarty_fl;

            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)) /
                        ((Antenna4.Y - Antenna3.Y) * (Antenna2.X - Antenna1.X) - (Antenna2.Y - Antenna1.Y) * (Antenna4.X - Antenna3.X)));
            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)) /
                        ((Antenna4.Y - Antenna3.Y) * (Antenna2.X - Antenna1.X) - (Antenna2.Y - Antenna1.Y) * (Antenna4.X - Antenna3.X)));

            Console.WriteLine("Robotstarting Position at:" + Starting.X + "
```

```
                    mm, " + Starting.Y + "mm");

                // Computing the orientation of the Robot
                //angle = (Math.Atan2(y, x)) * (180 / Math.PI);
                angleTemp = (Math.Atan2((Antenna1.Y - Starting.Y), (Antenna1.X -
                     Starting.X))) * (180 / Math.PI);
                Console.WriteLine("Angle temp " + angleTemp);
                angle = angleTemp - (double)(solution1[0]*45.0);  // in deg
                Console.WriteLine("Angle wrong direction " + angle);
                if (angle <= 0.0)
                {
                    angle = angle + 180;
                }
                else
                {
                    angle = angle - 180;
                }

               Console.WriteLine("Robotangle: " + angle + "Â°");

            }

        // Procedure and function
        // Method to compute the norm of a vector
        public static double Norm(PositionD p) // get the norm of a vector
        {
            return (Math.Pow(Math.Pow(p.X, 2) + Math.Pow(p.Y, 2), 0.5));
        }

        //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);
            ex.X = (point2.X - point1.X) / p2p1Distance;
            ex.Y = (point2.Y - point1.Y) / p2p1Distance;
            aux.X = point3.X - point1.X;
            aux.Y = point3.Y - point1.Y;
            //signed magnitude of the x component
            double i = ex.X * aux.X + ex.Y * aux.Y;
            //the unit vector in the y direction.
            aux2.X = point3.X - point1.X - i * ex.X;
            aux2.Y = point3.Y - point1.Y - i * ex.Y;
            ey.X = aux2.X / Norm(aux2);
            ey.Y = aux2.Y / Norm(aux2);
            //the signed magnitude of the y component
            double j = ey.X * aux.X + ey.Y * aux.Y;
            //coordinates
            double x = (Math.Pow(r1, 2) - Math.Pow(r2, 2) + Math.Pow(
                p2p1Distance, 2)) / (2 * p2p1Distance);
            double y = (Math.Pow(r1, 2) - Math.Pow(r3, 2) + Math.Pow(i, 2) +
                Math.Pow(j, 2)) / (2 * j) - i * (x / j);
            //result coordinates
            double finalX = 10 * (point1.X + x * ex.X + y * ey.X);
            double finalY = 10 * (point1.Y + x * ex.Y + y * ey.Y);
            resultPose.X = (int)(finalX);
            resultPose.Y = (int)(finalY);

            return resultPose;
        }

        // Methode to compute the position based on the ID in [mm]
        public static Position IDtoPOS(int ID)
        {
            Position FinalPos = new Position();
```

```
            int[] posx = new int[9] { 10, 10, 10, 20, 20, 20, 30, 30, 30 };
            int[] posy = new int[9] { 10, 20, 30, 10, 20, 30, 10, 20, 30 };
            // For a 3x3 testing field
            FinalPos.X = posx[ID - 1];
            FinalPos.Y = posy[ID - 1];
            return FinalPos;
        }

        // Find neighbours of the IDs
        public static int[] FindNeig(int[] arrID, int numTags)
        {
            // Init
            int[] neighbours = new int[numTags];
            int[] tempNeig = new int[4];

            // Find the number of neighbours
            for (int m = 0; m < numTags; m++)
            {
                // Init
                neighbours[m] = 0;

                // Take actual ID and compute the possible neighbours
                tempNeig[0] = arrID[m] - 11;
                tempNeig[1] = arrID[m] - 1;
                tempNeig[2] = arrID[m] + 1;
                tempNeig[3] = arrID[m] + 11;

                for (int v = 0; v < 4; v++)
                {
                    foreach (int tempinput in arrID)
                    {
                        if (tempinput == tempNeig[v])
                        {
                            neighbours[m] += 1;
                        }
                    }
                }
            }
            return neighbours;
        }

        // Methode to compute the best IDs and correct distances
        public static int[,] CorrectID_Distance(int[] arr, int[] arrRSSI, int
            numTags)
        {
            // Inputs
            /*  arr = Array of all IDs
             *  arrRSSI = Array of all RSSI
                numTags = Int with the num of tags found
            */
            // Init
```

```
            int[,] best = new int[2, 3];
            int[] dist = new int[numTags];          // Array which contain the best distance (max(0),middle(1),min(2))
            int[] neighbours = new int[numTags];
            int i = 0;
            int p = 4;

            // Compute the neighbours
            neighbours = FindNeig(arr, numTags);

            // Switch case for the different possible shapes
            switch (numTags)
            {
                case 3:
                    Console.WriteLine("3 Tags ----------");
                    for (int l = 0; l < neighbours.GetLength(0); l++)
                    {
                        if (neighbours[l] == 0)     // Detect outlier and boarder
                        {
                            dist[l] = 1;            // stay max
                        }
                        else if (neighbours[l] <= 3)     // Detect outlier and boarder
                        {
                            dist[l] = 0;            // stay max
                        }
                        else if (neighbours[l] == 4)     // Detect the inner, change it to min/middle
                        {
                            dist[l] = 1;            // change it to middle
                        }
                        else if (neighbours[l] > 4)     // Detect the inner, change it to min/middle
                        {
                            dist[l] = 0;            // change it to middle
                        }
                        // all other numbers are at the boarder
                    }
                    // Select the best 3 readings
                    i = 0;
                    p = 4;
                    while (i < 3)                   // Start for the first ID
                    {
                        for (int h = 0; h < neighbours.GetLength(0); h++)    // looks for a fitting
                        {
                            if (neighbours[h] == p && i < 3)    // hit must be same value and less then 3 hits
                            {
                                best[0, i] = h;                 // index of the
```

```
                                        best ID
                                    best[1, i] = dist[h];            // info about max, mid and min of this ID
                                    i += 1;
                                }
                                else if (i >= 3)
                                {
                                    break;
                                }
                            }
                            p -= 1;
                        }
                        break;
                /* ------------------------------------------------------------------------------ */
                case 4:
                    Console.WriteLine("4 Tags ----------");
                    for (int l = 0; l < neighbours.GetLength(0); l++)
                    {
                        if (neighbours[l] == 0)      // Detect outlier and boarder
                        {
                            dist[l] = 1;             // stay max
                        }
                        else if (neighbours[l] <= 3)      // Detect outlier and boarder
                        {
                            dist[l] = 0;             // stay max
                        }
                        else if (neighbours[l] == 4)      // Detect the inner, change it to min/middle
                        {
                            dist[l] = 1;             // change it to middle
                        }
                        else if (neighbours[l] > 4)      // Detect the inner, change it to min/middle
                        {
                            dist[l] = 0;             // change it to middle
                        }
                        // all other numbers are at the boarder
                    }
                    // Select the best 3 readings
                    i = 0;
                    p = 4;
                    while (i < 3)                    // Start for the first ID
                    {
                        for (int h = 0; h < neighbours.GetLength(0); h++)    // looks for a fitting
                        {
                            if (neighbours[h] == p && i < 3)     // hit must be
```

```
                                        same value and less then 3 hits
                                    {
                                        best[0, i] = h;                 // index of the best ID
                                        best[1, i] = dist[h];           // info about max, mid and min of this ID
                                        i += 1;
                                    }
                                    else if (i >= 3)
                                    {
                                        break;
                                    }
                                }
                                p -= 1;
                            }

                            break;
                    /* ------------------------------------------------------------------------------------------ */
                    case 5:
                        Console.WriteLine("5 Tags ----------");
                        for (int l = 0; l < neighbours.GetLength(0); l++)
                        {
                            if (neighbours[l] <= 2)      // Detect outlier and boarder
                            {
                                dist[l] = 0;             // stay max
                            }
                            else if (neighbours[l] == 3)      // Detect the inner, change it to min/middle
                            {
                                dist[l] = 1;             // change it to middle
                            }
                            // all other numbers are at the boarder
                        }
                        // Select the best 3 readings
                        i = 0;
                        p = 4;
                        while (i < 3)                    // Start for the first ID
                        {
                            for (int h = 0; h < neighbours.GetLength(0); h++)    // looks for a fitting
                            {
                                if (neighbours[h] == p && i < 3)     // hit must be same value and less then 3 hits
                                {
                                    best[0, i] = h;                 // index of the best ID
                                    best[1, i] = dist[h];           // info about max, mid and min of this ID
```

```
                i += 1;
            }
            else if (i >= 3)
            {
                break;
            }
        }
        p -= 1;
    }
    break;

/* ---------------------------------------------------------------------------- */
case 6:
    Console.WriteLine("6 Tags ----------");
    switch (neighbours.Sum())
    {
        case 12: // Shape with 2 outliers
            Console.WriteLine("2 Outliers");
            for (int l = 0; l < neighbours.GetLength(0); l++)
            {
                if (neighbours[l] <= 2)     // Detect outlier and boarder
                {
                    dist[l] = 0;            // stay max
                }
                else if (neighbours[l] == 4)     // Detect the inner, change it to min/middle
                {
                    if (arrRSSI[l] <= 3)
                    {
                        dist[l] = 2;            // change it to min
                    }
                    else if (arrRSSI[l] > 3)
                    {
                        dist[l] = 1;            // change it to middle
                    }
                }
                // all other numbers are at the boarder
            }
            break;
        case 14: // Shape like a domino
            Console.WriteLine("Domino");
            for (int l = 0; l < neighbours.GetLength(0); l++)
            {
                if (neighbours[l] <= 2)     // Detect outlier and boarder
                {
```

```
                                dist[l] = 0;            // stay max
                            }
                            else if (neighbours[l] == 3)     // Detect the centre, change it to min
                            {
                                dist[l] = 2;            // change it to min
                            }
                            // all other numbers are at the boarder
                        }
                        break;
                    default:
                        Console.WriteLine("Default case");
                        break;
                }
                // Select the best 3 readings
                i = 0;
                p = 4;
                while (i < 3)                       // Start for the first ID
                {
                    for (int h = 0; h < neighbours.GetLength(0); h++)   // looks for a fitting
                    {
                        if (neighbours[h] == p && i < 3)    // hit must be same value and less then 3 hits
                        {
                            best[0, i] = h;                 // index of the best ID
                            best[1, i] = dist[h];           // info about max, mid and min of this ID
                            i += 1;
                        }
                        else if (i >= 3)
                        {
                            break;
                        }
                    }
                    p -= 1;
                }

                break;
            /* ---------------------------------------------------------------------------- */
            case 7:
                Console.WriteLine("7 Tags ----------");
                for (int l = 0; l < neighbours.GetLength(0); l++)
                {
                    if (neighbours[l] <= 3)     // Detect outlier and boarder
                    {
                        dist[l] = 0;            // stay max
```

```
                        }
                        else if (neighbours[l] == 4)      // Detect the centre, change it to min
                        {
                            dist[l] = 2;              // change it to min
                        }
                        // all other numbers are at the boarder
                    }
                    // Select the best 3 readings
                    i = 0;
                    p = 4;
                    while (i < 3)                    // Start for the first ID
                    {
                        for (int h = 0; h < neighbours.GetLength(0); h++)    // looks for a fitting
                        {
                            if (neighbours[h] == p && i < 3)     // hit must be same value and less then 3 hits
                            {
                                best[0, i] = h;                   // index of the best ID
                                best[1, i] = dist[h];             // info about max, mid and min of this ID
                                i += 1;
                            }
                            else if (i >= 3)
                            {
                                break;
                            }
                        }
                        p -= 1;
                    }
                    break;
                default:
                    Console.WriteLine("Default case");
                    break;
            }
            foreach (int ee in best)
            {
                Console.WriteLine(ee);
            }
            return best;
        }
    }
}
```