technische universität dortmund

Fakultät Bio- und Chemieingenieurwesen

A group project final report on:

Advanced AGV Drive Control in a Pipeless Plant

Group members:

Angel Garza

Awjoykanti Bonik

Lusan Maharjan

Mauricio Martinez Severich

Shehabeldin Abdelgawad

Seyed Ali Baradaran Birjandi

Supervised by:

Goran Stojanovski

Sankaranarayanan Subramanian

Shaghayegh Nazari

April 7, 2015

Table of Contents

List of figures

List of tables

1. Introduction

Angel Garza, Mauricio Martinez, Shehabeldin Abdelgawad, Seyedali Baradaranbirjandi

1.1 Motivation (why a pipeless plant)

Batch plants are of great importance in the chemical industry. In order to increase the flexibility and efficiency of batch plants, replacing the maze of pipes with one or multiple mobile vehicles that transport the material in the plant has been proposed. The pipes can be replaced with multiple automated guided vehicles, also referred to as AGVs. Pipe cleaning times which are needed after every changeover would be saved in a pipeless plant. This flexibility, however, comes at a cost. The problem of task/resource allocation as well as collision-free movements of the mobile vehicles introduces new challenges in a pipeless plant [16].

1.1.1 The miniature pipeless plant

To simulate a pipeless plant a miniature plant was built at the Process Dynamics and Operations Group (DYN) in the scope of the European project MULTIFORM. The plant aims to simulate the production of chemicals by producing multi-layered and multi-colored plaster arts. The details of producing the plasters are provided to the plant as recipes. The plant comprises AGVs, four different stations controlled by a PLC, each station is responsible for accomplishing a particular task in producing a layer of plaster art:

- **AGVs:** used to transport the vessels between stations.
- **Storage station:** vessels (either empty or full) are stored in this station, including vessel exchanges.
- **Mixing station:** adds and mixes compounds that are needed for plaster hardening.
- **Color stations:** supply the different colored plasters. Two colors are available per station.

There is also a camera that can be used to obtain visual feedback of the plant. An overview of the plant can be seen in Figure 1.

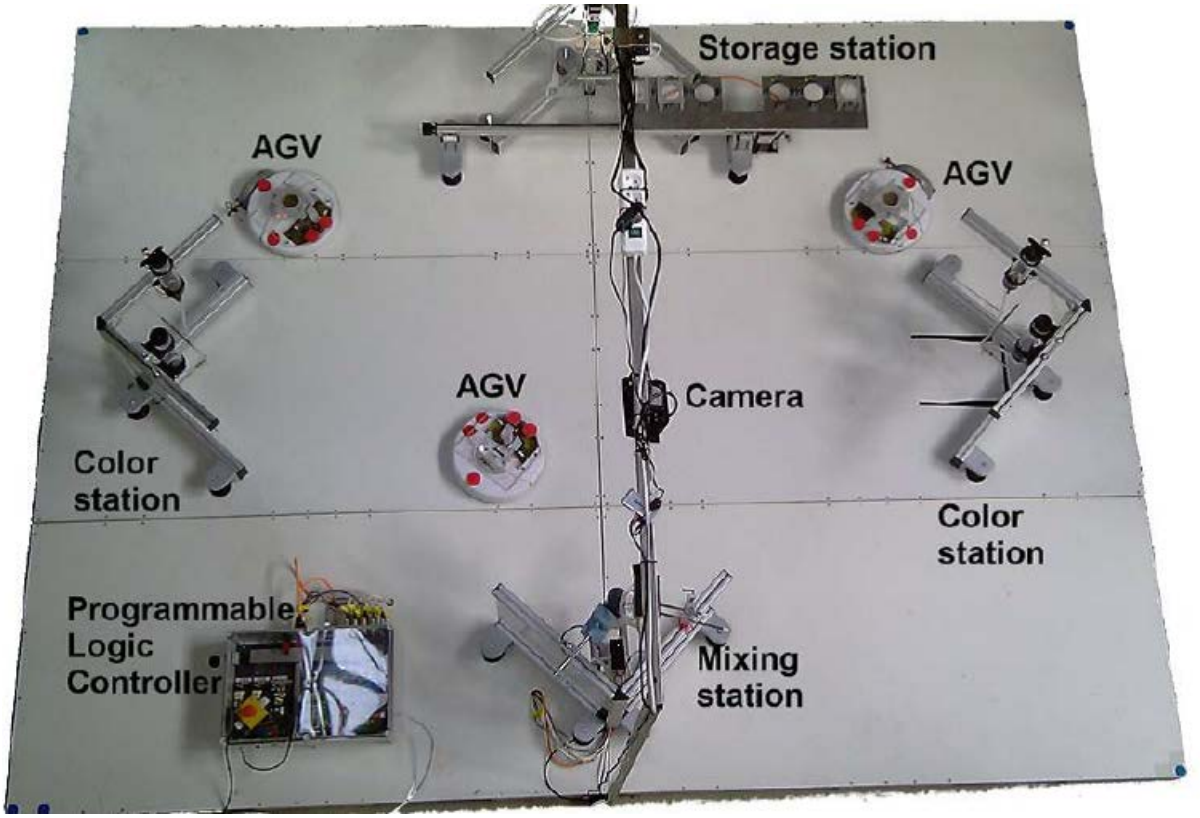


Figure 1: Plant overview

1.2 Plant module layout (previous implementation)

The plant is implemented in a modular fashion to enable easy upgrading and swapping of components thus making prototyping and new component integration easy and simple.

Scheduler: The operation starts with a recipe selection phase, then it is passed to the scheduler module. The scheduler module is then responsible for assigning the recipes to the list of available AGVs. The Scheduler deals with allocating the plant's resources such that the plaster art recipes are executed.

Router: The router gets the generated schedule as an input and it is responsible for generating paths for the movement of the AGVs which participate in executing the given schedule. The router ensures collision free movements of the AGVs by generating a velocity profile for each path.

Controller: The controller module is responsible for supplying the inputs (left and right wheel velocities) to the AGVs, such that the AGVs reach their desired

destinations with the correct orientations, the location and orientation are determined by the camera.

Vision system: The vision module is responsible for detecting the plant's state; this is done by detecting the positions and orientations of the AGVs relative to the plant's coordinate system. This is intended to be used as the controller's feedback. The project had an image processing algorithm, with an image processing time of 250ms [1] and upon inspection needed to be faster.

1.3 Objective of the group project

The plant was operating on a modelica model, to simulate the real plant's operation. The objective of the group project is to provide the framework with necessary controllers to first test and then carry out an execution of a set of configurable recipes autonomously on the physical plant.

In order to achieve the main goal, the following improvements and developments must first take place as part of the group project:

- Proper communication with AGVs must be established
    - Reading data
    - Sending information and commands
- A reliable control module must be developed
    - Orientation reliability and correction
    - Position reliability and correction
- A reliable low delay method of feedback for the controller module must be implemented
    - Use of encoders
    - Use of camera feedback (image processing)

2. Hardware

Angel Garza Palomares

2.1 iRobot Create

The plant is equipped with Create robots from the company iRobot. As mentioned before, these robots (so called Automated Guided Vehicles (AGV)) will perform the movement tasks inside the plant carrying the vessels to the corresponding stations.

2.1.1 iRobot Create Open Interface

The Create Open Interface (OI) is a software interface for controlling Create's behavior and reading its sensors, this is done through a series of commands called *opcodes*. To achieve this purpose, a processor capable of generating serial commands should be connected to one of the two ports on the Create providing serial communication [11].

In this project, the robots are already equipped with an Arduino board connected to the Cargo Bay Connector. Therefore, it is possible to communicate with the robots through the Arduino. Further, a communication between the Arduino and the user (PC) is needed, for this purpose an Xbee communication is already implemented.

The Arduino is the microcontroller which sends the corresponding opcodes to the robot and the sensors' information to the PC. As already mentioned, the Arduino is connected through Cargo Bay Connector which provides a 5V 100mA supply when the robot is switched on; it also contains 4 digital inputs and 4 digital outputs; the board with the proper connections was already constructed in the previous work. For more information on the hardware development, please see [1]. Finally the Xbee is used for the communication between the PC and the microcontroller, the code on the PC determines the speed for each wheel in order to carry out the tasks. In Figure2 an overview of the communication is presented.

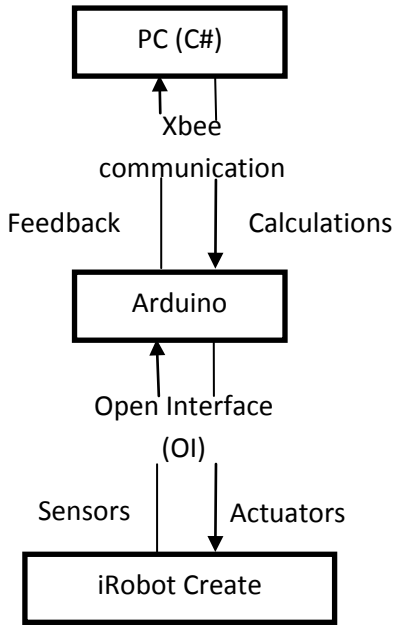


Figure 2: Overview of communication

2.2 Sensors

The Create contains 43[1] different data packets from which sensory information can be read. There are two supported communication modes for the transfer of data packages, continuous and interrupt based. In continuous communication mode that can be activated using *opcode 148*, there is a continuous stream of sensory information every 15ms.On the other hand, the interrupt based communication that can be activated using command *opcode 142*, means that the information is retrieved only when a certain event happens (sampling time, button, etc.). Previously the *opcode 148* was implemented getting the information from all the sensors even though not all data is useful for the project. However, some problems (explained in the following chapters of this document) were encountered, not only of the amount of obtained data, also the sampling time used by the implemented opcode.

[1]*There are 34 different data that can be obtained from the sensors, each having a specific packet ID; there are 2 more packet IDs that are unused; and finally 7 more packet IDs that contain a stream with more than one type of data (refer to the Create Open Interface manual), in total 43 packet IDs .*

One of the goals of this project is to make a sensor fusion between the encoders, which are integrated in the wheels of the Create, and the camera vision system (the camera system is explained in chapter 5.) to get an accurate location of each robot at the plant. In this section, the properties and limitations of the encoders are discussed, as well as the modifications which were done in this part during the project.

According to the manual of the robot, the linear and angular resolutions of the encoders are:

Linear: (68 mm wheel diameter × π) / (32 teeth × 25 gearing reduction) = 0.27mm
Angular: 360 degrees / (32 teeth × 25 gearing reduction) = 0.45 degrees

However, the value is given as a signed integer value, therefore the exact number in millimeters or degrees could not be obtained unless the value is an exact integer.

2.2.1 Properties of the encoders

It is possible to read distance and angle information from the encoders (in mm/s).Data is given as signed 16 bit integer, and the range is: -32768 – 32767. It is mentioned in the manual of the Create that, for better accuracy, it is better to keep the speed between -200 – 200 mm/s, please see [14], even though the speed range is -500 – 500 mm/s. Distance and angle are obtained using the packet ID 0, 2 or 6 as part of a group of sensors; or individually, the packet ID 19 for distance and 20 for angle. Note that both data packets have a size of 2 bytes.

2.2.2 Limitations of the encoders

Some limitations of the encoders have to be considered; first of all the fact that every time the distance or angle is read from them the data gets cleared. After every update the encoders start counting from zero, this is an issue when the sampling time and the velocity are too small since it could lead to a distance smaller than 1 millimeter which would always result in 0 millimeters since the value is truncated and given as an integer. Given that the retrieved data is given as an integer value a loss of information is present, i.e. 3.4 mm would be

read as 3 mm, so the error depends on how often the encoders are read and also on the length of the path. The best way for having the most accurate values would be to get the information as often as possible, but at the cost of loss of information. Originally *opcode 148* was implemented and the encoders were being read every 15ms, this would mean if the robot moves at a speed of 50 mm/s, 66 updates occur in one second (note that after each update the distance and angle values are cleared). So at every update, the value is less than one (0.75mm). The value read is 0 at all time. If the entire movement is at a low speed, the distance travelled from the computer's perspective will be zero. It has to be noticed that the error is much more pronounced if the angle is read, because a bad orientation could lead to a totally different spot than the desired one with the further movements.

In order to overcome this problem, a possible solution is to read the number of encoder impulses, but due to software limitations this was not possible.

2.3 Modifications

Considering that reading the information of the sensors at a sampling time of 15ms is fast for the purposes in this project, which the idea is to use this information as a source of feedback, it was decided to change it for a sampling time of 200ms.This would reduce the error and the information loss between two consequent samples would not affect the accuracy too much.

Thus, *opcode 142* has to be used instead of *opcode 148*. This implementation is done in the microcontroller's code and opposed to the *opcode 148* which is called only once at the beginning starting the continuous streaming, *opcode 142* was implemented in a way that it is called every 200ms. This change has improved the accuracy but the error is still considerable specially while reading the angle.

2.4 Testing control with encoder feedback (opcode 142)

Awjoykanti Bonik

To test the encoder accuracy an interface called *TestRobot* was developed. This interface was created in order to be able to specify a desired distance or angle and speed to the robot, the interface can be seen in Figure 3.

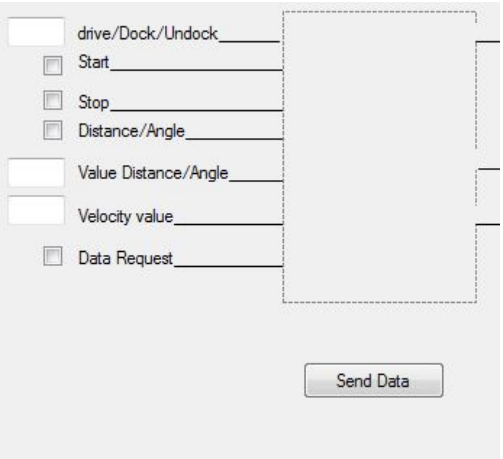


Figure 3: The TestRobot interface

The important input data fields for test purposes are:

- Distance/Angle: If checked, the value set in Value Distance/Angle is the desired angle (positive means counter clockwise); if the box is not checked, then the value set in Value Distance/Angle is the desired distance (negative means backward).
- Value Distance/Angle: the desired distance or angle.
- Velocity value: the speed at which the robot will do the movement. The sign determines the direction of the linear or angular movement (this value can be changed while the robot is running)

While the robot is running, it will continuously update the distance by using the encoder feedback. The frequency of the update depends on the sampling time which is set in the microcontroller. Whenever the accumulated distance or angle is greater than or equal to the desired value, a “0” velocity command will be sent to the robot to stop. Linear distance was measured manually.

2.4.1 Results of linear movement of the AGVs with encoder feedback

A test was carried out by running the robot for specific linear distance (forward and backward) of 500 mm. Ten samples for each velocity that is presented in Table 1 were taken.

Samples / Velocity (mm/s)	1	2	3	4	5	6	7	8	9	10	Avg. (mm)	Error (%)
50	509	501	509	498	493	515	510	492	495	503	502.5	0.5%
-50	515	503	502	503	512	497	498	495	515	500	504	0.8%
100	512	532	510	512	533	518	535	506	524	535	521.7	4.2%
-100	522	516	544	530	555	526	545	536	536	541	535.1	6.5%
150	511	563	524	512	557	556	574	558	523	538	541.6	7.7%
-150	563	516	530	543	553	529	513	565	570	536	541.8	7.7%
200	556	596	518	559	518	521	515	515	555	575	542.8	7.9%
-200	543	550	570	542	518	573	560	577	564	578	548.5	8.8%

Table 1: Linear movement error

The following figures show the percentage of error at some specific velocities for both cases (forward and backward):

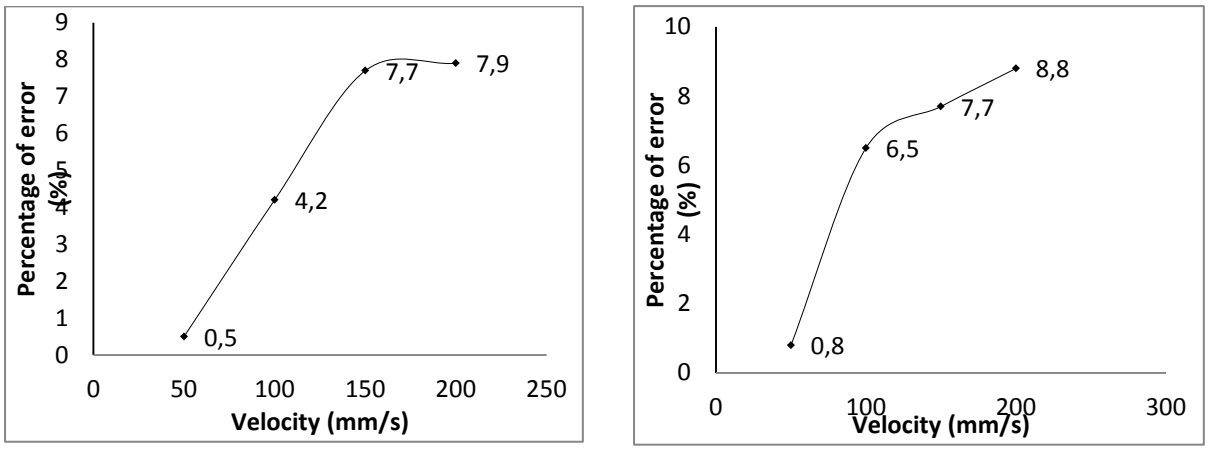


Figure 4: Percentage of error for different velocities during linear movement (forward and backward)

It can be observed that the error increases in fast movements.

2.4.2 Results angular movement with encoders feedback

The same test was performed for the angular movement. The robot was tested for a rotation (clockwise) of 360 degrees and 5 samples for each velocity were taken. The test results are shown in Table 2:

Samples / Velocity (mm/s)	1	2	3	4	5	Avg. (degree)	Error (%)
50	382	380	370	377	385	378	4.8
100	376	380	377	382	388	380.6	5.4
150	388	395	373	382	390	385.6	6.6
200	420	373	390	385	390	391.6	8.1

Table 2: Angular movement error

The following graph shows the behavior of the error percentage for different velocity values:

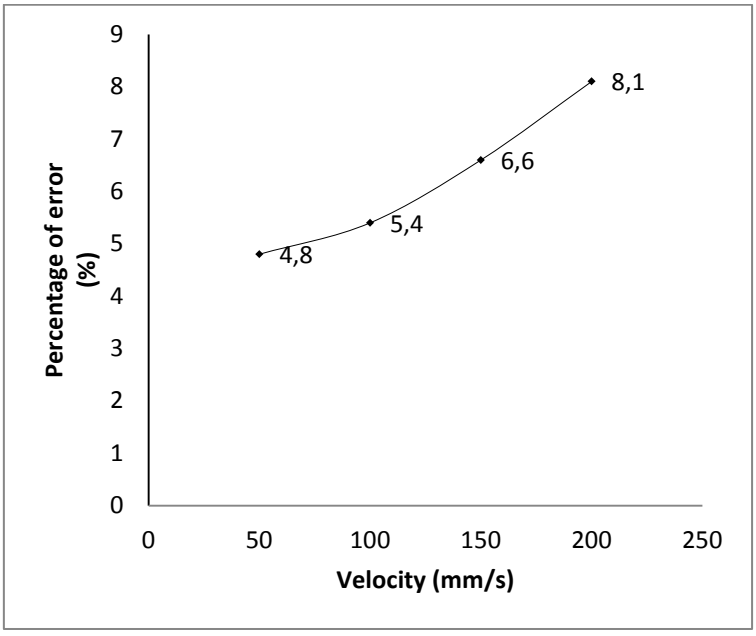


Figure 5: Percentage of error for different velocities angular movement

Considering the angular movement, the orientation error can lead the robot to a different location.

2.5 Conclusion

As mentioned in chapter 1, one of the purposes of this project is the use of the odometry information in conjunction with the camera vision system as a source of feedback for the controller (chapter 5), to avoid noisy measurements or processing time from the camera. As it was discussed in this chapter, there is a compromise between how often the sensors are read and the accuracy of the information. The loss of information cannot be avoided or reduced by the use of the encoder's impulses; however, since detecting the impulses is not possible for the encoders on the Create's wheels, that option was discarded.

The controller works well for linear movements because the error introduced by the odometry does not cause a big difference between the reached point and the desired one. However, when the controller (using encoder's feedback) is used to achieve a desired orientation, the error becomes much more considerable and a good result cannot be achieved.

After trying different options (opcodes, sampling times), the performance of the controller for angular movements was not good enough, because unlike linear movement, the error presented in angular movement could lead to a large error; thus it was decided not to use the encoders as a source of feedback, instead the camera vision system is used (discussed in chapter 5).

3. Routing Module

Shehabeldin Abdelgawad

3.1 Overview of operation

The plant as discussed in chapter 1 starts with a recipe selection phase which when complete is passed to the scheduler module. The scheduler module is then responsible for assigning the recipes to the list of available AGVs. The schedule is then passed to the router module, where the movements of the AGVs are calculated and collision is avoided using an A* algorithm. The routing module's job is to execute the schedule given by the scheduler while avoiding collisions between the AGVs.

3.2 Previous state of routing module

The routing module's output is a list of groups. Where each group corresponds to a period of time in the schedule in which the AGVs move. It is important to note that the movement times of the routing module is always shorter than that given to the schedule to allow for correction and provide a safety margin. For example the schedule is programmed such that each movement takes 15 seconds, then the routing module is set such that the longest movement takes 10 seconds for instance to provide time for the AGV's pose correction. Each group in the list contains a list of a class called the "CollisionInfoForPath", which corresponds to the robots that will move in that group. The "CollisionInfoForPath" class contains the array of Path points for which the AGV should pass by, the velocity array that the AGV should follow along its path, as well as its name, Array of orientations and task ID, see Figure 6.

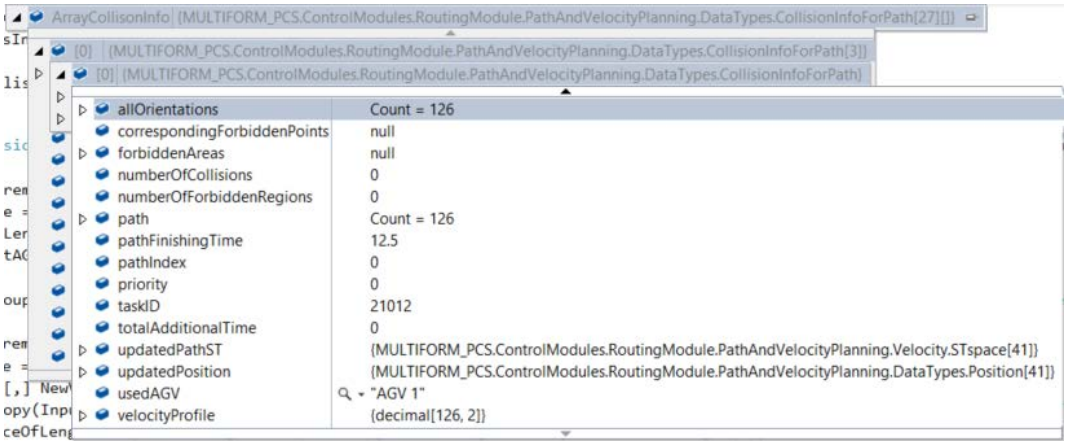


Figure 6: Array generated by the "CollisionInfoForPath" class

In the previous figure we can see the array of the first AGV of the group number 0, it is important to notice that for each AGV that has to complete a movement task, the amount of values for the path has to be the same than the amount of values for the allOrientations and velocityProfile arrays; this means that the AGV has to fulfill an specific orientation, position and velocity at each sampling time. Since the orientation is considered to take zero time on the path it was decided to stop every AGV running simultaneously (including the global time) and run the controller to change the orientation of the AGV which need it, for this purpose the allOrientations array is used, the current index is compared with the previous index and if the values are different, the controller is executed until the new orientation is reached, then the global time continues from where it was stopped as well as the AGVs that were stopped (in case any of them were stopped), see Figure 7.

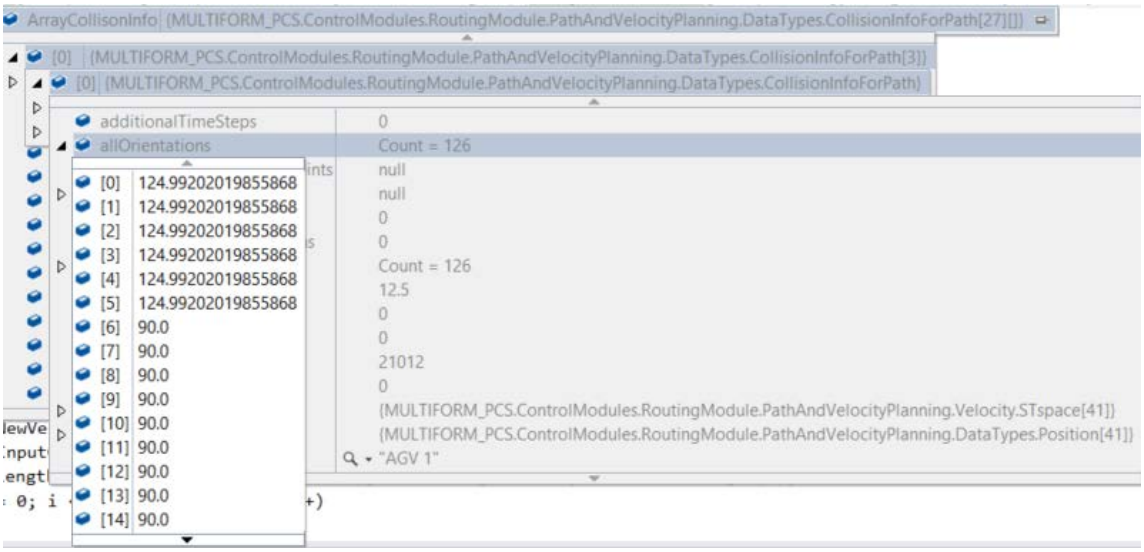


Figure 7: Orientation along the sampling times

This is the resulted array for group number 0, in this group only AGV1 needs to moved but the arrays for the other two AGVs have to exist, otherwise an “out of range” exception will be shown, that’s why the arrays for AGV2 and AGV3 are created with null values, see Figure 8.

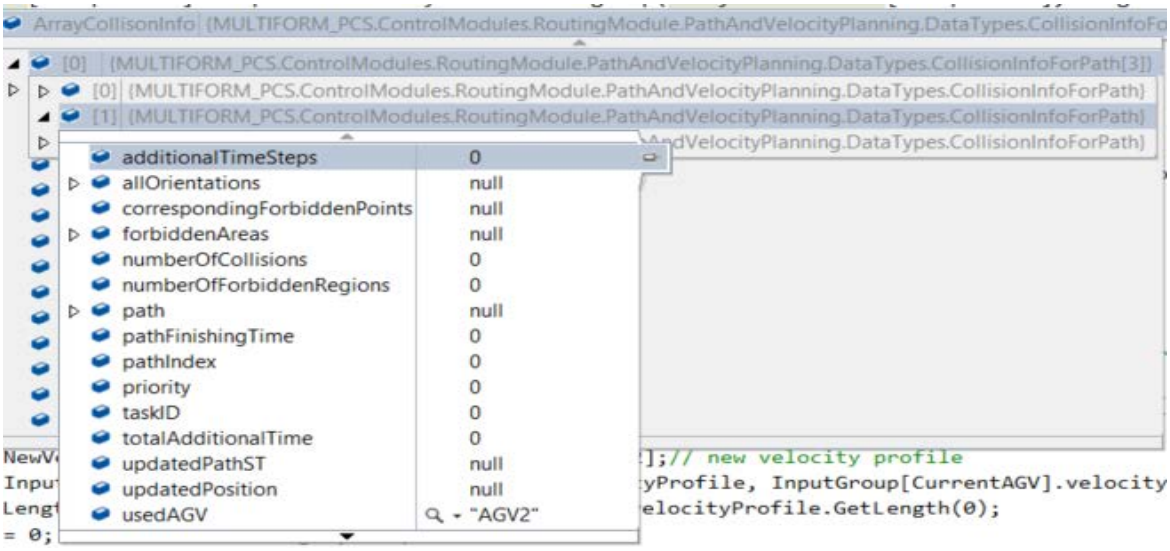


Figure 8: Arrays for AGV2 in group 0.

As it can be seen in Figure 8, the array for AGV2 is created even though it doesn't have to perform any movement, the same occurs for AGV3 at group 0.

3.3 Limitations of the current implementation

The current implementation of the routing module is not loop friendly since the array is jagged, meaning that not all groups have the same number of AGVs and the array has no structure. The first AGV in the group can be any AGV that has been scheduled to move in the group. With this in mind, it is not possible by only looking at the current group to identify which robot should move without checking a string at every iteration. The major problem however is that not all the AGVs have the same velocity profile length, as some may finish earlier than others. To make the router module's output more loop friendly and easier to visualize and debug, some modifications were made.

It is important to note that the routing module assumes rotations to occur instantaneously (rotations take 0 time). This is a negative aspect of the current implementation of the routing module. Due to this assumption, when an AGV rotation is detected during the following of the router modules velocity profile, the global time is stopped as well as all other AGVs which are following velocity profiles. This is done to simulate the instantaneous rotation of the AGV and ensure collision free operation of the plant.

3.4 Modifications to router module's output

Each group was modified to include all AGVs being considered on the plant at the point of the route creation, this means that even if the robot does not move in the current group it will have an entry which has a velocity profile/array of zero and a null point array.

The velocity profiles are all of the same length corresponding to the longest velocity profile of the group. If a velocity profile is extended it is padded with zeros.

The orientations of each "CollisionInfoForPath" were of the same number as the number of points which were 8-10 times less than the number of entries in the velocity profiles. The orientations have been extended according to the number of velocity entries such that the initial and final entries remain the same. This will later provide an easily accessible method to obtain the orientation reference for feedback.

3.5 Feed-Forward module's principal of operation

The feed forward module obtains the router module's output and works to implement it on the AGVs. It is important to note that the plant operates in two phases. The first is the "Feed-forward" phase in which the AGVs receive their velocities directly from the Feed-Forward module without using any feedback. The second phase is the "correction phase" in which the AGVs are controlled using camera feedback to the final destination points of their path.

The "Feed-Forward" phase feeds the velocity profiles calculated by the routing module to the robots at the intervals specified by the velocity profiles (100ms). It is assumed by the routing module that the rotations happen instantaneously, to simulate this instantaneous rotation the schedule's time is stopped. The rotations are done not using feed-forward but by using camera feedback.

The "correction phase" is executed at the beginning of each group to ensure that the initial positions of the robots are correct before the start of the feed-forward phase, and at the end of each phase to ensure that docking is possible. For the docking procedure to be started, the AGV has to be approximately 50cm away from the station where it has to dock (straight line), there are two

black lines which the AGV has to follow with the help of two (left and right) of the four cliff[2] sensors on the Create robot in order to reach the exact position where the action of the station takes place, finally with the help of the bumper[3] sensor (front part of the robot) the AGV detects when the position has been reached. One must mention that the error incurred by the feed-forward is not constant and changes with every run of the plant (random error). During the correction phase the AGVs are controlled using the controller which obtains feedback through the camera. The system is sampled during this time as fast as possible to provide the best possible accuracy.

It is important to also note that the correction time is not constant and depends on the amount of error introduced during the feed-forward phase. For the plant to have enough time in all cases to correct the poses of the AGVs, an upper bound for correction time is always added at the end of a group, if for some reason the time is not enough for correction the plant would no longer function correctly for the next group.

This procedure is repeated until the schedule is complete.

3.6 Improvement of AGV movements (Ramp Velocity Profile)

Lusan Maharjan

The main task of an AGV is to transport the fluid from one station to another station for various processes such as mixing, filling, hardening and so on. The fact that the fluid carried by the AGV reaches the destination safely should be considered. Apart from these, it should be taken into consideration that, the AGV should move to the desired destination accurately. Using the rectangular velocity profile leads to sudden movements and stopping of the AGV to the maximum velocity. Thus, the moment of inertia is highly observable in rectangular velocity profile which introduces more error to the AGV movement as well as it increases the chance of spilling the fluids from the vessel carried by the AGV. But this is not the case with a ramp velocity profile.

---

[2] The Create robot has 4 cliff signal sensors (left signal, front left signal, right signal and front right signal. Packet ID 28, 29, 30 and 31 respectively). The signal of these sensors is returned as an unsigned 16-bit value (Range: 0 – 4095)

[3] The bumper sensor returns a value of either 1 or 0; where 1 means that the bumper is being pushed (Packet ID: 7).

In a ramp velocity profile, the velocity of the AGV gradually increases with a constant rate until it attains the maximum velocity, then it travels with this "peak" velocity for certain interval of time, after which the velocity of the system again decreases at the same constant rate until the AGV comes to rest. As there is no such sudden change in the velocity of the AGV, it is less affected by inertia due to which it has less chance of spilling the fluid from the vessel. It also moves more accurately to the destination. Thus, a ramp velocity profile was chosen over a rectangular velocity profile for the movement of the AGV.

3.6.1 Conversion of rectangular velocity profile to ramp velocity profile

The routing module provides the rectangular velocity profiles of the AGVs. Converting the rectangular velocity profiles to ramp velocity profiles leads to lower errors during movements.

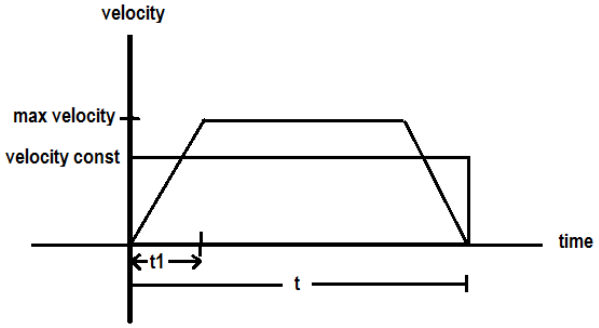


Figure 9: Ramp and rectangular velocity profiles

In order to convert the rectangular profile into a ramp profile, a maximum velocity of 200mm/s is considered. A previous investigation showed if the robot moves with a velocity higher than 200mm/s, then the risk of spilling the fluid from the vessel is present. The following procedure is suggested.

1. Calculate the total ramp time (t1) of the ramp velocity profile.

$$totalramptime(t1) = \frac{(\max velocity - velocityconst)}{velocityconst} * totaltime$$

2. Calculate the next velocity for the next time slot considering an interval time of 100ms. A ramp time of 100ms is assumed for simplicity such that the ramp velocity profile can be calculated

$$rampvelocity = \frac{\max velocity * ramptime}{totaltime}$$

3. Calculate the time interval for which the robot moves with the maximum velocity.

$$maximumvelocitytime = totaltime - 2 * totalramptime$$

4. Increase the velocity of the robot by calculated ramp velocity at every ramp time until it reaches the maximum.
5. Move with the maximum velocity for the calculated time.
6. Decrease the velocity of the robot by the calculated ramp velocity at every 100ms until the robot stops.

The procedure results in a ramp velocity profile for moving equal amount of distance in the same interval of time.

3.6.2 Results and Conclusion

The explained procedure has been tested by moving the robot with a specific velocity for a specific interval of time as shown in the table below. For this experiment, the robot is moved with the constant velocity for certain interval of time. The travelled distance by the AGV is then recorded. After that, the error is calculated using the rectangular profile. The same experiment is performed, this time with ramp velocity profile also recording the measurements. After performing the experiments and calculating the error, the data are compared and it is observed that the final accuracy of the robot is improved when using ramp profiles. The observed distances for both the velocity profiles are listed below.

Sample	Velocity(mm/s)	Time(s)	Distance Travelled(mm)	Estimated Distance(mm)	Error (%)

1	50	10	501	500	0.2
2	100	10	1003	1000	0.3
3	150	10	1480	1500	1.3
4	190	10	1843	1900	3

Table 3: Robot moving with ramp velocity for 10 second

Sample	Velocity(mm/s)	Time(s)	Distance Travelled(mm)	Estimated Distance(mm)	Error (%)
1	50	10	504	500	0.8
2	100	10	980	1000	2
3	150	10	1440	1500	4
4	190	10	1780	1900	6.3

Table 4: Robot moving with different rectangular velocity for 10 second

As mentioned, the ramp velocity profile shows improvement. But despite being the better one, we are not able to use this profile in the existing plant. As mentioned earlier, the rotation of the robot is considered to happen instantaneously. This means the robot has to take zero time for performing the rotation which is possible when the plant is paused while rotation takes place. Thus, the moving robot stops suddenly if any one of the robots needs to rotate. After rotation of the robot is finished this moving robot will resume its motion but this time it will start to move with the same velocity instantly at which it was paused just before the rotation of other robot takes place. Thus, the velocity profile acts as rectangular profile at this point. This leads to violation of the ramp velocity profile.

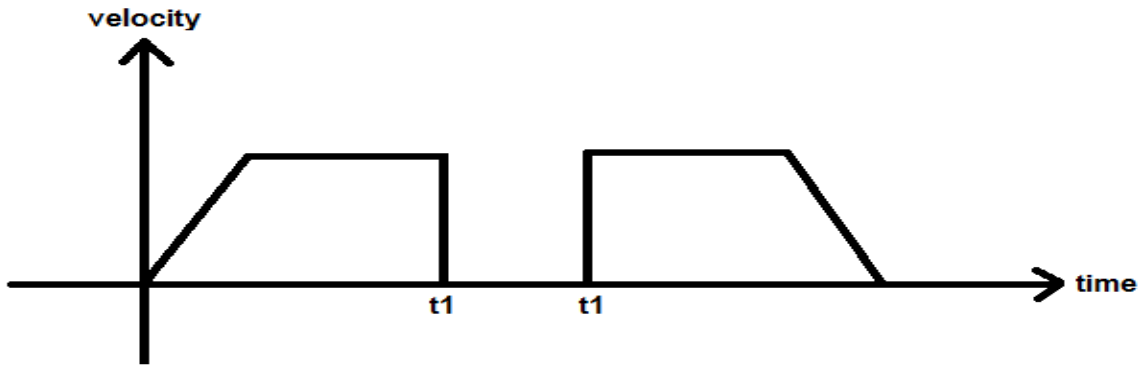


Figure 10: Example to illustrate the violation of ramp velocity profile

In addition to this, there is another problem concerning the speed limit of the robot. If the constant velocity provided by the routing module is more than 200mm/s then the maximum attainable velocity for the ramp profile should be beyond 200mm/s. This leads to violation of our assumption.

If the robot moves with very low speed and the time interval for the movement is low, then in some cases it is unable to attain the top speed of 200mm/s as the maximum velocity for ramp profile. Even if it is possible the ramp velocity will be so high that it would actually behave like constant velocity profile. To avoid this problem, the maximum attainable velocity for the ramp profile is reduced until the desirable result is achieved. The minimum value is set to maximum attainable velocity for ramp profile as 25mm/s.

4. Controller module

Seyedali Baradaranbirjandi

4.1 Robot kinematics

Kinematics is the science of studying motions of bodies regardless of the forces or moments that cause the motion. Choosing the proper formulation of kinematics model for a robot or a manipulator is vital. Basically there are two different spaces used in kinematics modeling, Cartesian space and Quaternion space [10]. According to the purpose of modeling and the type of motion, one will decide to select the proper space. It is more convenient to represent the kinematic model of AGVs in Cartesian space, though. In addition forces and constraints of each wheel in two-wheeled mobile robots must be shown with respect to a clear reference frame. This is important in mobile robots because a clear relation between the moving and fixed parts of this particular plant is required [10]. The position/state of a two-wheeled robot in Cartesian coordinate system can be represented by 3 variables, namely x, y and θ. Figure 7 shows the position of an AGV.

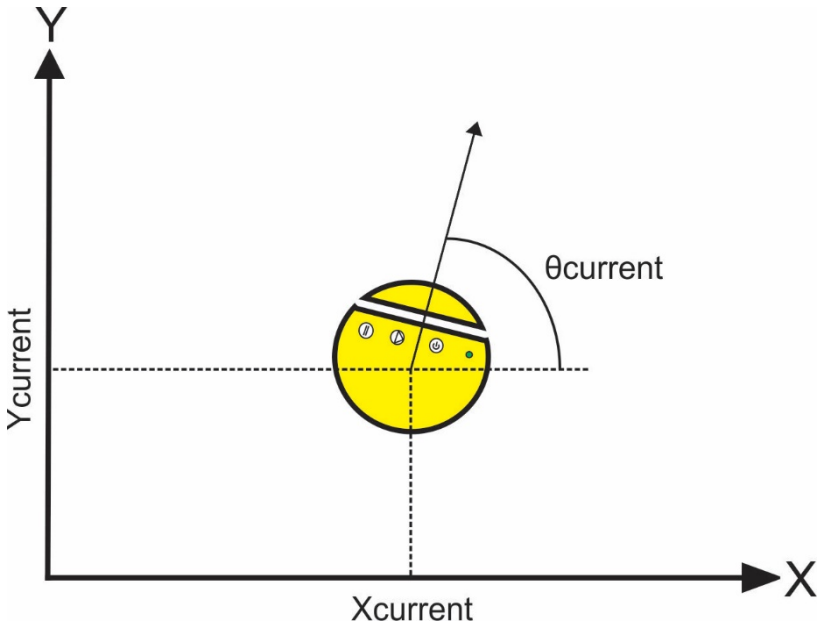


Figure 11: position representation of an AGV

According to the position representation, the kinematic model of a two-wheeled mobile robot can be described by the following equation:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} cos(\theta) & 0 \\ sin(\theta) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \quad (4.1)$$

Where *v* is the linear velocity and *ω* is the angular velocity of the robot.

4.2 Controller

As it was discussed in the introduction, there are different strategies for wheeled mobile robot locomotion. The goal is to drive each AGV to a specified desired position. Nevertheless, there are some assumptions about the robots and the plant.

4.3.1.1. It is assumed that the robot moves only in one direction. In other words, there is no backward motion. Thus the distance between current position and destination is always positive.

4.3.1.2. The route of each AGV is provided by the routing module which was explained in section 3.

4.3.1.3. Each AGV moves with a command which it receives from the main control routine.

Problem description: As it is mentioned in Chapter 3, the route is determined by a higher level function. The path includes some intermediate points. These intermediate points are passed to the controller sequentially. In other words, each intermediate point is a reference for the controller, until the robot reaches the next intermediate point. According to the reference the controller determines the angular and linear velocity of the robot, computes each wheel's velocity and sends the proper command to the robot. Thus the problem is to find each wheel's angular velocity according to the current position of the robot and coordination of the next intermediate point.

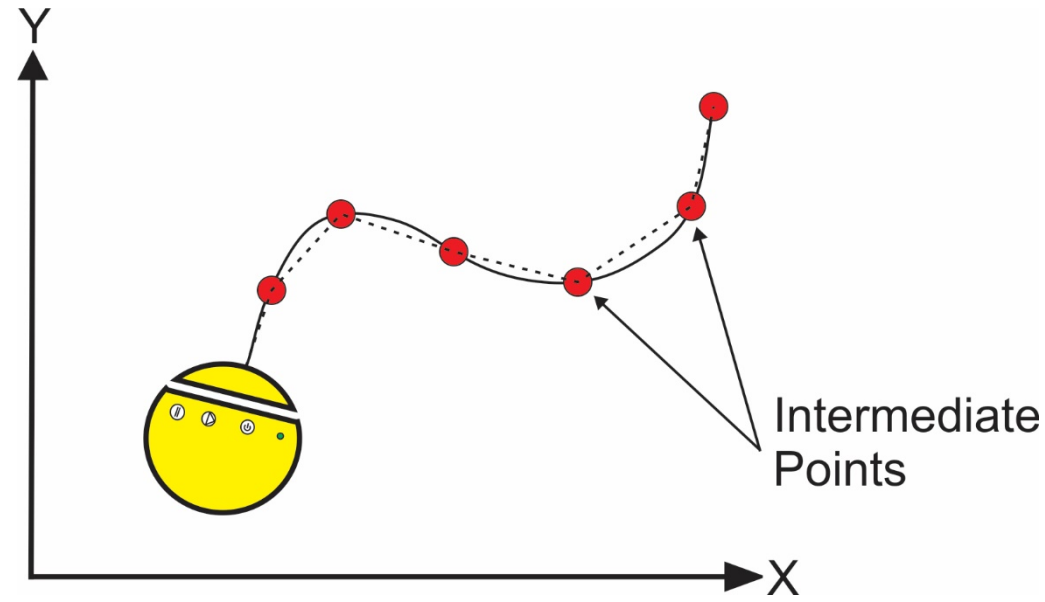


Figure 12: shows the AGV route and respective intermediate points

The control methods examined for AGVs in this project are categorized into 2 groups:

4.3.2.1. Open loop control

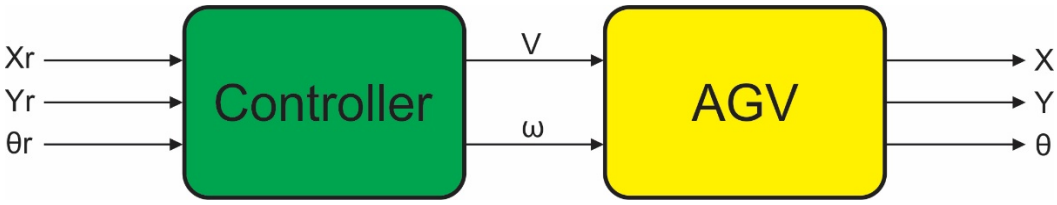


Figure 13: open loop control block diagram

Consider a free body which intends to move linearly from point A to point B. There are two factors that determine the quality of the motion, namely time and velocity. Regardless of the distance between A and B, if one says that this body must reach the destination in 5 seconds for example, then the velocity is calculated accordingly and given to the body. On the other hand velocity can be kept constant instead. Let us say that the body has to move with velocity 5 m/s. Accordingly the time of motion between A and B will be the division of distance over velocity. With this brief explanation, the strategy used in open loop control will be clear. In this approach either velocity or time need to be kept constant and accordingly the other one is calculated. Strictly speaking, it can be said that the robot has to reach the destination with a constant speed.

Thus the time will vary for different destinations. However, it is possible to keep the time constant. As a result, velocity will change for different destinations. The time is considered constant in this project, though. $t_{fixed,1}$ and $t_{fixed,2}$ are the time for linear and angular motion respectively. These parameters are chosen 10 seconds in this project. This means that no matter if robot intends to rotate 10 degrees or 350 degrees, the rotation will last 10 seconds. However, the velocity of rotation for 10 and 350 degrees will be different. A similar example for linear motion can be cited as well.

Two strategies were deployed for open loop control:

a) Linear and angular velocities are fed to the robot separately.

The motion of the robot between 2 intermediate points is divided into 3 phases:

1. Rotate toward the next intermediate point. As it is mentioned above, it is considered that the time is constant (instead of velocity), angular velocity of the robot and each wheel's velocity have to be calculated.

$$\Delta\varphi = \varphi_{destination} - \varphi_{current} \tag{4.2}$$

$$\omega = \frac{\Delta\varphi}{t_{fixed,1}} \tag{4.3}$$

$$\omega_L = \frac{\omega \times L}{2R} \tag{4.4}$$

$$\omega_R = -\omega_L = -\frac{\omega \times L}{2R} \tag{4.5}$$

$$V = 0 \tag{4.6}$$

where $t_{fixed,1}$ is the fixed time of rotation,
ω_L and ω_R are the left and right wheels' angular velocity, respectively,
L is the distance between the two wheels and R is the radius of each wheel.

2. Translate toward the destination.

$$\rho = \sqrt{(X_{destination} - X)^2 + (Y_{destination} - Y)^2} \tag{4.7}$$

$$V = \frac{\rho}{t_{fixed,2}} \tag{4.8}$$

$$\omega_R = \omega_L = \frac{V}{R} \tag{4.9}$$

where ρ is the distance between the current AGV position and the destination and $t_{fixed,2}$ is the fixed time of linear translation.
$t_{fixed,1}$ as well as $t_{fixed,2}$ are chosen arbitrarily.

3. Rotate to the desired angle. This phase is executed when the robot reaches the final intermediate point (final destination). The procedure is the same as the first phase.

b) Linear and angular velocities are fed to the robot simultaneously.

Given that the position and rotation of current and next intermediate points of an AGV are known, the path between these 2 intermediate points can be described with an arc of a circle. Accordingly the problem will be to determine the center of this circle.

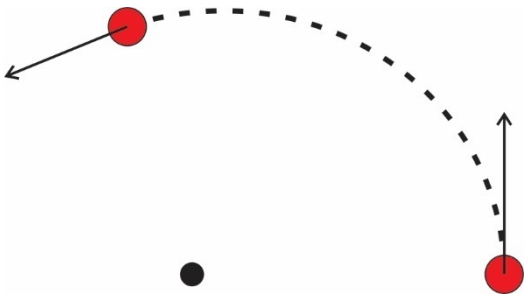


Figure 14: the curved path between two intermediate points

Since the time of the motion is considered to be fixed, once the center of this circle is calculated, linear and angular velocities of the robot can be determined. It is assumed that V and ω are constant along the path. The angular velocity of each wheel is computed by the following formulas:

$$\omega_R = \frac{(2\times V + \omega\times L)}{2R} \tag{4.10}$$

$$\omega_L = \frac{(2\times V - \omega\times L)}{2R} \tag{4.11}$$

However this approach has a lot of drawbacks one of which is neglect of obstacles on the arc of the circle. The routing module which is responsible for determining the intermediate points and avoiding obstacles, assumes that AGV moves straight between the intermediate points. Therefore in this curved path there may be an obstacle which was not expected by the upper level.

For both the above-mentioned approaches, it is considered that the time is a reference for the AGV's motion. When time reaches the desired value, the current motion/phase (rotation or translation phase) needs to be stopped.

4.3.2.2. Closed loop control:

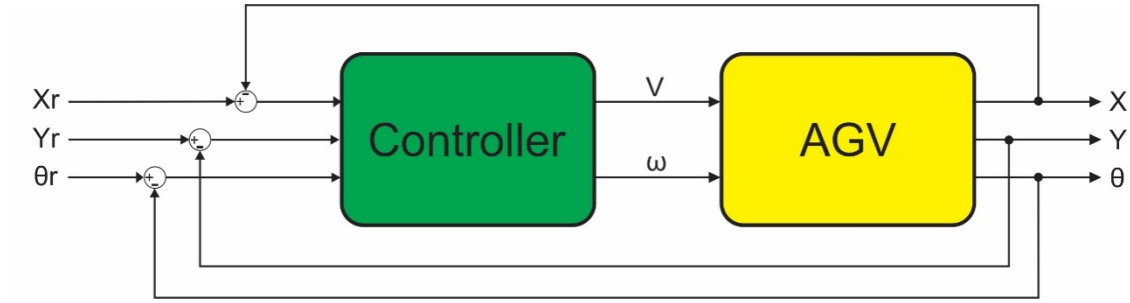


Figure 15: closed-loop system

The difference between open and closed loop control is that the error between the desired and the current position is taken into account, in order to determine the velocity of an AGV in closed loop system. A P controller is employed in this project. (Equations 4.13 and 4.16). In other words, the angular and linear velocities of an AGV are proportional to the size of the error. Bigger error results into bigger velocity.

There are two ways to measure the current position of the robots: Using the encoders of the robot's motors or using the data from the camera installed above the plant. Since the encoders in rotation phase are not accurate enough, it was decided to use the camera information to locate the AGV. This data is compared with the reference value and the error is obtained accordingly. The motion of the robot is again divided into 3 phases. In each phase the AGV is either fed with angular or linear velocity. Consequently there will be two gains, one for rotation and one for translation.

In the rotation phase the rotational error is,

$$\Delta\varphi = \varphi_{destination} - \varphi_{current}\ , \tag{4.12}$$

$$\omega = P_1 \times \Delta\varphi\ , \tag{4.13}$$

$$\omega_R = -\omega_L = -\frac{\omega \times L}{2R} \tag{4.14}$$

where P_1 is the rotation gain. The value for rotation gain is $4\text{x}10^{-4}$ which is chosen empirically for this project.

And for the translation phase, the translational error is,

$$\rho = \sqrt{(X_{destination} - X)^2 + (Y_{destination} - Y)^2} \tag{4.15}$$

$$V = P_2 \times \rho \tag{4.16}$$

$$\omega_R = \omega_L = \frac{V}{R} \tag{4.17}$$

where P_2 is the translation gain. The value for rotation gain is $15\text{x}10^{-3}$ which is also chosen empirically for this project.

Due to the sensor limitations and the robot's inaccuracies, the error will not become exactly 0. Thus there will be some acceptable margins of error for each phase. Once the error is less than a specified threshold (2 degrees for rotation and 7 cm for translation), the phase will change and the AGV will enter the next stage.

4.3 Conclusion

Even though the open loop control system is remarkably simple to implement, it has many disadvantages. The environment of the robot, specifically the field in that the robot moves, is not fully known. If there are uneven ground conditions for example, the robot will end up not of course in the desired destination. Therefore it is more reasonable to control the robot in a closed loop system. As a result, in the presence of disturbance such as uneven ground condition or bad calibrated wheels, the robot can still converge to the desired

destination. However, maintaining the stability of an AGV in a closed-loop system is vital. To exemplify the importance of stability, consider a situation in which the robot is considerably far away from its destination. The translational error and the linear velocity will become big respectively. Conversely the velocity of wheels cannot increase arbitrarily. Thus there need to be a threshold for the maximum speed of the wheels in order to maintain the stability of the robot.

5. Image Processing and Localization

Mauricio Martinez Severich

5.1 Camera vision

One element in the current project is a fisheye camera positioned above the plant. The camera is from IDS (Imaging Developing System GmbH). The image acquired using this camera covers all the plant, and will be used to calculate the necessary feedback - the location information (position and orientation) of the robots – to the controller.

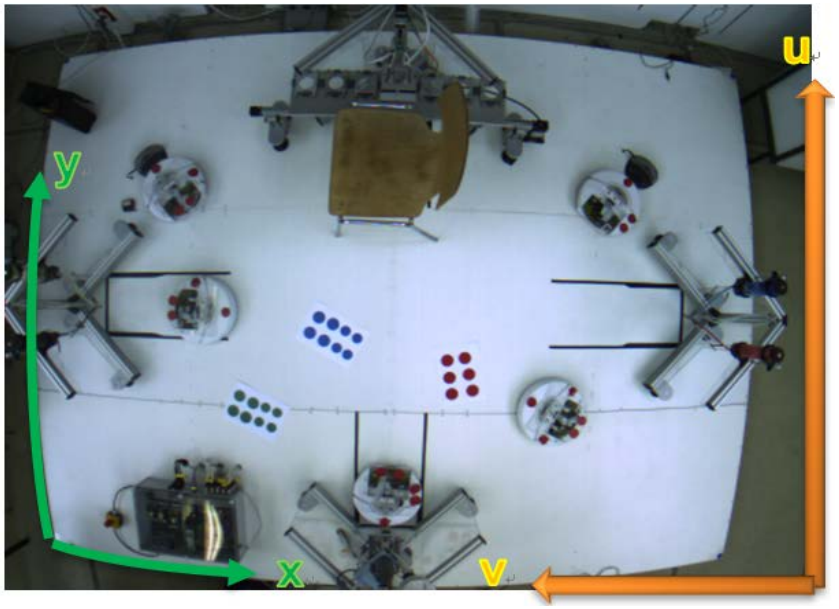


Figure 16: Plant seen by fisheye camera

Using the current system, there are two main issues regarding the camera vision system.

- The image processing time.
- Dewarping the image and transforming from the pixel coordinate system (u, v) to the world coordinate system (x, y).

5.2 Image Processing

During an earlier group project, an image processing algorithm was implemented. According to the project report, the image processing time was 250ms (from camera capture until calculating the pose information in the

World coordinate system) [1].In order to improve the image processing time, a new algorithm is implemented in this project.

5.2.1 Introduction

The current project is developed in C# platform with WPF (Windows Presentation Foundation) format, image processing tools will be necessary to include to the project solution.

OpenCV was designed for computational efficiency with a strong focus on real-time applications. It contains library that can take advantage of multi-core processing [2].

Emgu is a cross platform, that allows OpenCV functions to be called from .NET compatible languages such as C#, VB, VC++, etc.[3].

The procedure to develop the image processing algorithm will be explained in the following sections. It follows the idea, to capture the image, segment the image, cluster the segmented tokens according to the number of robots present on the image and finally calculate the position and orientation.

5.2.2 Segmentation

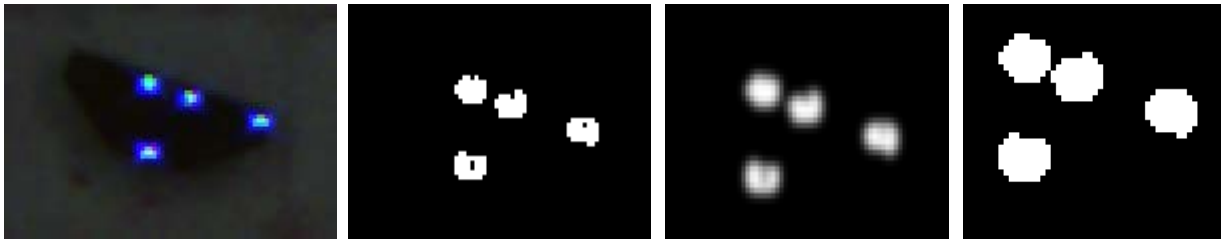
The PCBs with different LED configurations that were designed in the previous report [1] are plugged onto the robots. These LEDs will be the points of interest for the segmentation.

Figure 17 : Right image: Robot with the PCB. Left Image: segmentation of the LED

An appropriate threshold has to be selected to obtain a segmented image. Since the points of interest for the algorithm is the brightness of the LEDs, a non-linear colour space called HSV (hue, saturation, value) will be used.

Hue: is the property of a color that varies in passing from red to green. **Saturation:** is the property of the color that varies in passing from red to pink. **Brightness** (sometimes called **lightness** or **value**): is the property that varies in passing from black to white [4].

To improve the segmentation, the algorithm blurs the image using a Smooth Gaussian Filter and then applies again a threshold in the grayscale space. This improvement is necessary to approximate the segmented region of interest to circles. It later compares the image using the Hough Circles detector algorithm, concluding with the information of the circles detected in the image coordinate system(u, v).



a) Original b) First threshold c) Smooth Gaussian Filter d) Final Threshold

Figure 18 : Segmentation steps

5.2.3 Clustering

The clustering algorithm needs an input of the current number of Robots present on the image. This can be solved by applying a threshold of the number of tokens (detected LEDs) in the image.

The algorithm to be used for clustering is the K-means clustering [5]. The procedure is the following:

1. Choose a fixed k number of cluster (Number of Robots).
2. Select k random tokens as initial cluster centers.
3. Assign tokens to the nearest cluster center using the Euclidean distance criteria.
4. Compute the centroid as the mean of all the tokens belonging to the cluster.
5. Repeat until there is no change in the number of clusters.

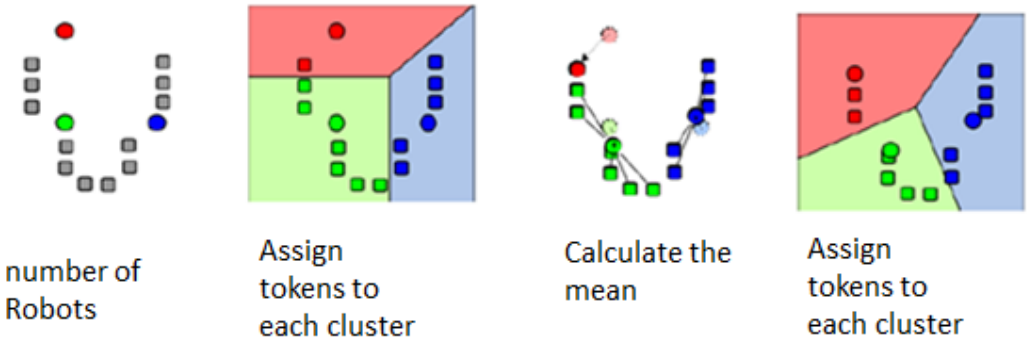


Figure 19: k means clustering

The result of the algorithm is shown in the following figure. Note that the Robots are distinguished by the color of the point of interest.

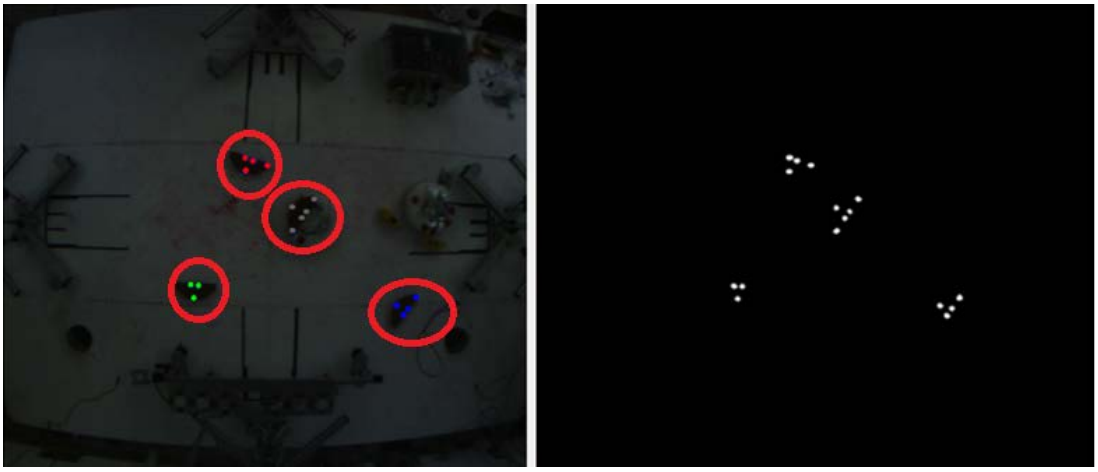


Figure 20 : Clustering of 4 different LEDs Configuration

5.2.4 Position and orientation

Now that all the tokens are clustered, each cluster can be analyzed in terms of position and orientation.

Figure 18 illustrates the LEDs configuration of the Robots, this configuration is arranged in a way to identify the Robot, localize the position and calculate the orientation. This configuration was designed in the previous Project implementation [1].

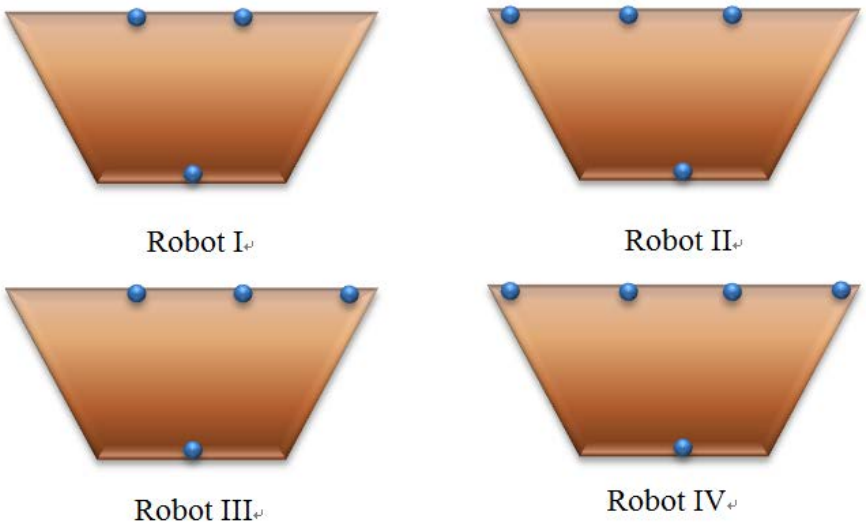


Figure 21: Identity markers of Robot I, II, III and IV

In order to find the apex of each Robot, the following procedure is applied:

1. Select randomly 3 tokens.
2. Check if the tokens do not belong to a line, using triangle areas criteria. If yes proceed with step 3 otherwise repeat step 1.
   Use the Heron Formula to obtain the area of the triangle.

$$Area = \sqrt{s(s-a)(s-b)(s-c)} \tag{5.1}$$

Where:

$$s = \frac{a+b+c}{2} \tag{5.2}$$

$$a, b, c \text{ are the triangle sides}$$

To verify whether the tokens belong to a line, the area is compared to a threshold.

Figure 22 : Determine the main triangle

3. Find the Apex in the triangle using Euclidean distance criteria with the vertices.

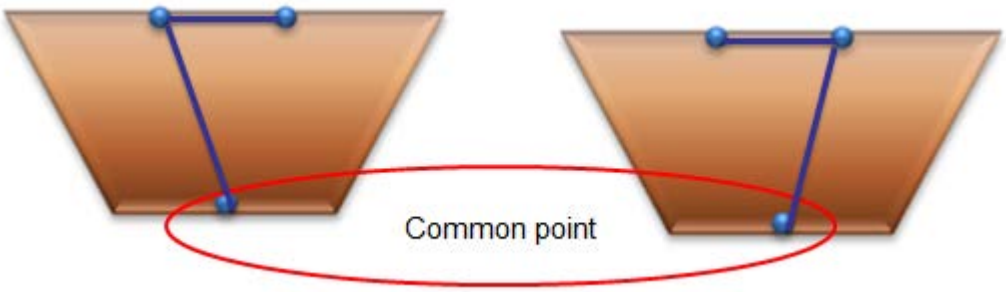


Figure 23: Find the apex

Now that the main triangle has been detected, we calculate the midpoint and the orientation using the following formula.

$$\overrightarrow{p_{12}} = \left(\frac{u_1 + u_2}{2}, \frac{v_1 + v_2}{2}\right) \tag{5.3}$$

where:

$$\overrightarrow{p_1} = (u_1, v_1);\ \overrightarrow{p_2} = (u_2, v_2)\overrightarrow{p_1}, \overrightarrow{p_2} \in P \tag{5.4}$$

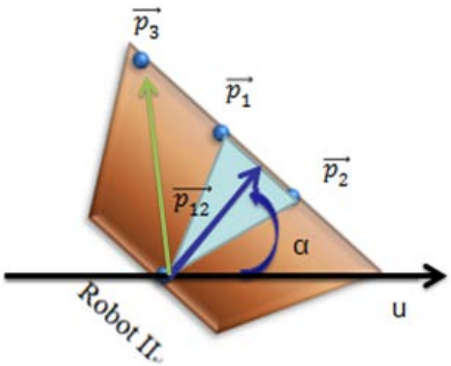


Figure 24: Orientation and midpoint

A simple way to identify the Robot is by the number of detected LEDs. This method leads to a problem with the Robot II and Robot III. To solve this issue, the cross product of the two vectors p3 and p12 is calculated: $\overrightarrow{p_3} \times \overrightarrow{p_{12}} = (u_3 v_{12}) - (v_3 u_{12})$

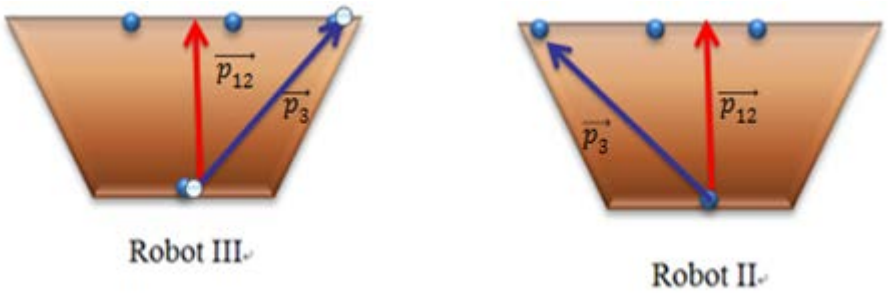


Figure 25: Vectors for the cross product

The result of the cross product for Robot II will be positive and for Robot III will be negative.

5.3 Camera Calibration and Mapping

Shehabeldin Abdelgawad

As previously discussed in chapter 2, the encoders on the AGVs do not provide sufficient accuracy for rotation, they are also prone to errors that accumulate and needs to be corrected by a world reference frame from time to time regardless of their accuracy. Therefore the use of camera feedback is necessary for the correction of the accumulated errors of the encoders and for all rotation movements of the AGVs.

5.3.1 Advantages of using fisheye lenses

Fisheye lenses have a wide view angel this makes it possible to capture the entire plant from a relatively low height. The camera is fixed such that it obtains a top view of the plant. The fixation and captured image can be seen from Figure 1and Figure 13respectively.

5.3.2 Disadvantages of using fisheye lenses

Fisheye lenses introduce heavy distortion which can easily be seen from Figure 13, if one looks to the edges of the plant which should appear to be straight, in the image appear to be curved. This is caused mainly by the radial and

tangential distortion introduced by the fisheye lens. Fisheye lenses suffer mainly from radial distortion and mildly from tangential distortion. As we move away from the center of the image, the radius and consequently the amount of radial distortion increase. The optical center of the image is where the radius is considered to be zero.

5.4 Factors affecting the reliability of the camera feedback

The plant is present within an environment whose lighting conditions are affected by the sun. This causes changes to the light intensity falling on the plant and thus being reflected to the sensor. This causes problems with the segmentation process which depends partly on the intensity values for segmentation.

Another factor that decreases the reliability of the camera feedback is the noise created by bright or highly reflective objects on the plant such as metal bolts, Xbee LEDs and reflections on the clear acrylic fixed to the sides of the stations. The last factor influencing the reliability of the camera feedback negatively is the saturation caused by the high light intensity of the LEDs.

All these factors affect the segmentation process which in turn affects the clustering process leading to inaccurate results, since the robots are not correctly identified. These factors are introduced through the design and location of the plant, and can be seen in Figure 23.

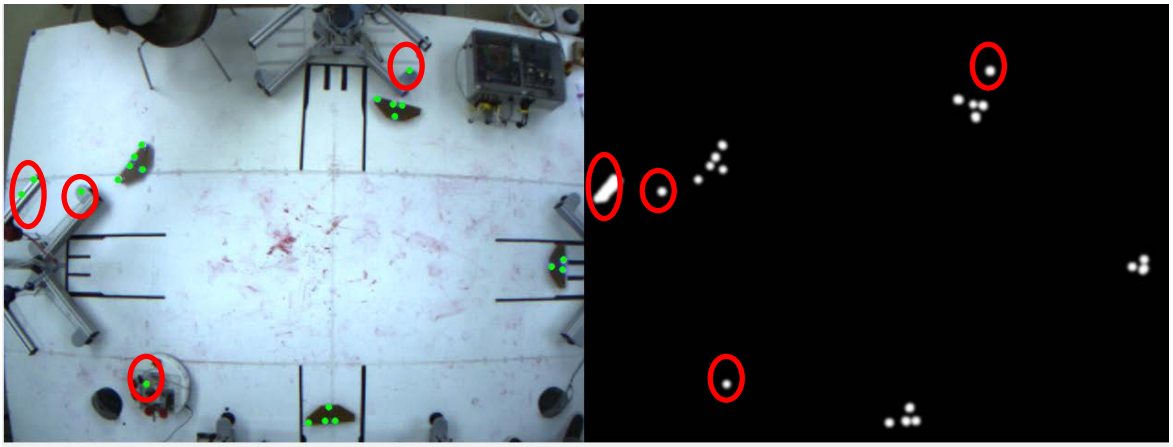


Figure 26: Noise due to reflections

5.4.1 Solution to improve reliability of camera feedback

The previous factors affecting the reliability of camera feedback are all caused by the high light intensity in the same HSV range as the LEDs used to detect the robots. The reliability can be improved by reducing the exposure time of the camera sensor to the plant when taking images or by reducing the aperture size of the camera, both methods result in the sensor being exposed to lower light intensity values. This mainly reduces the noise introduced by changing light conditions and helps with metallic and glossy reflections on the plant. The Xbee LEDs have been coved from the top to prevent the camera from detecting them as they are also of the same color of the LEDs used to define to robots. An example image is show in Figure 24.

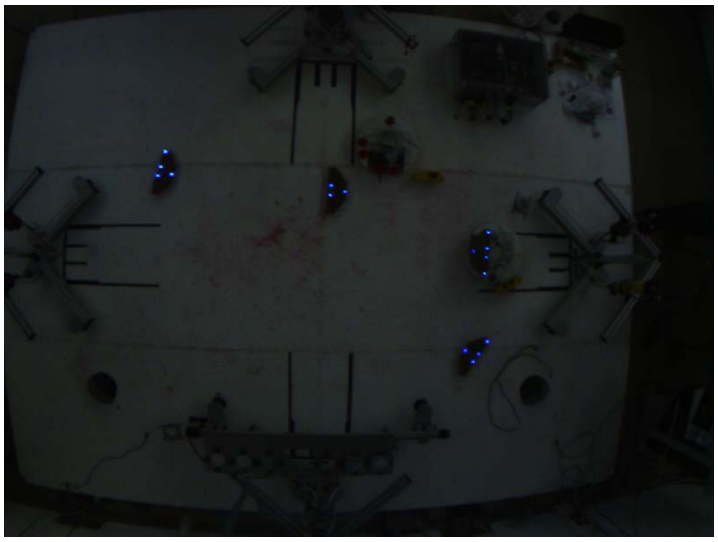


Figure 27: Image after aperture size reduction

5.5 Problems associated with distortion introduced by the Fisheye lens

Due to the Tangential and Radial Distortion introduced by the fisheye lens, the mapping from Plant coordinates to World coordinates becomes nonlinear. This results in different pixel areas for a constant plant area, due to the stretching effect introduced by the lenses' distortion. The difference depends on the location of the detected area in the image. One can see a "stretch" effect in figure27, which increases with radial distance from the optical center. This change in the area due to distortion makes the detection of the apex difficult as the detection of the apex depends on the area of the detected triangles in the images. Without distortion correction which requires calibration, the areas

of the triangles vary along the plant significantly and the control of the AGVs in plant coordinates using camera feedback would become inaccurate.

5.5.1 Compensating distortion introduced by the Fisheye lens

As previously discussed, Fisheye lenses introduce heavy distortions which must be compensated to provide adequate localization accuracy for feedback control.

Calibration is the mapping of a device's output to a set of reference values such that a measurement of the output of the device can be mapped/converted to a meaningful value in some reference.

In the current project context the camera pixels are referred to as (u, v). These coordinates must be mapped to the plant coordinates (x, y, z).However, since the plant is a plane we only consider (x, y). This is achieved by using calibration images, Images which have an easy to detect set of points in both world/plant and pixel coordinates, which correspond to one another. This was done using a checkerboard pattern. The calibration process uses the following functions from OpenCV: FindChessboardCorners, CalibrateCamera and InitUndistortMap.

A sample image before and after the detection can be seen in Figure 25andFigure 26.

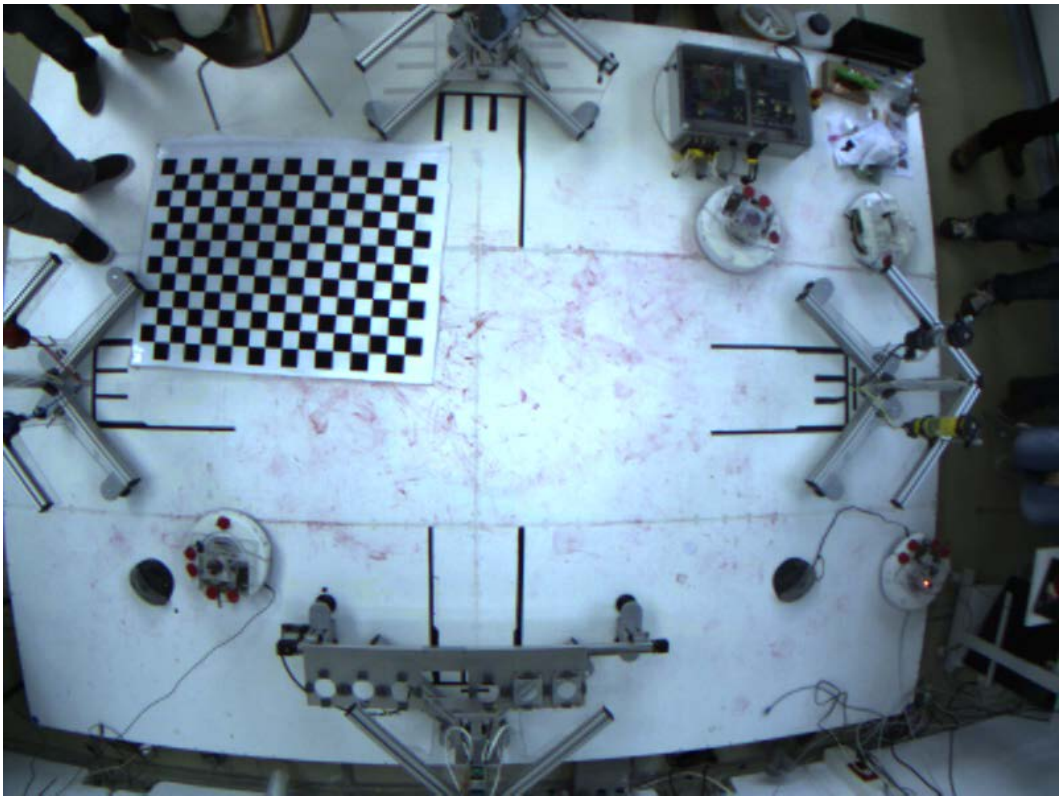


Figure 28: Before detection

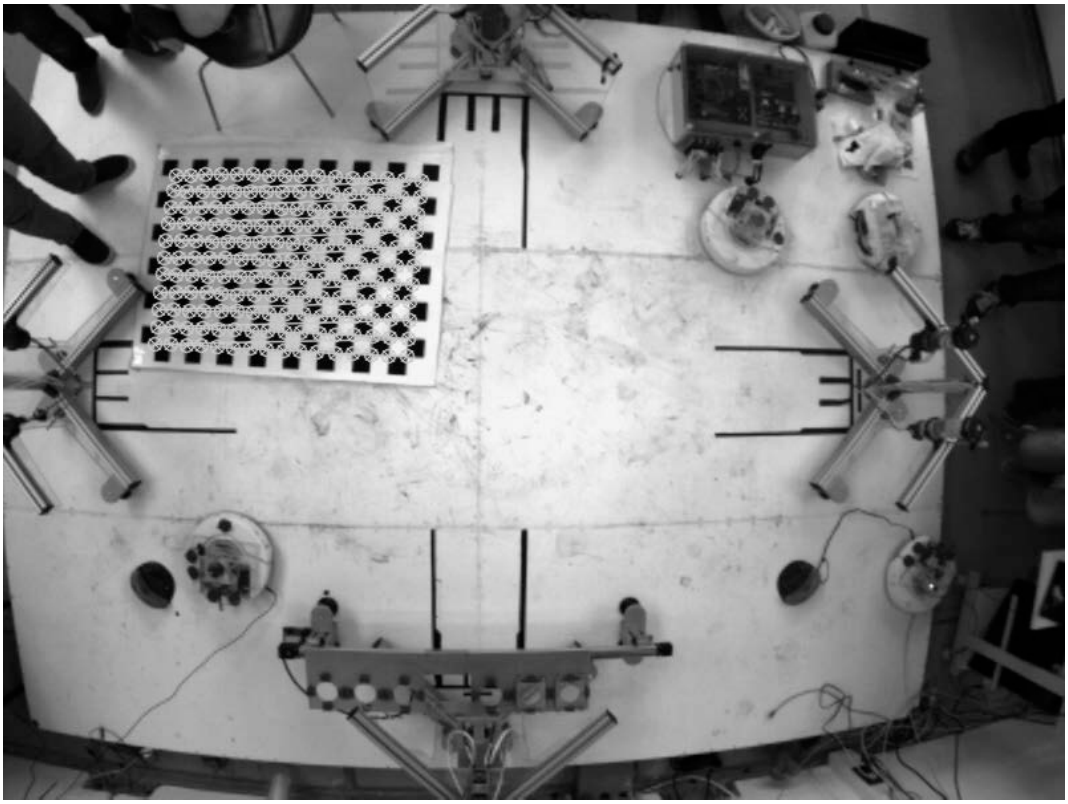


Figure 29: After detection

The sequence for the camera calibration is as follows:

- Images with the checkerboard pattern in a known position relative to the plant's coordinate system are taken.
- The images are passed to the calibration class in the main program, which dose the following.
    - The plant coordinates of the upper left corner of the pattern are passed to the calibration class.
    - The corners of the checkerboard pattern in the images are detected and mapped to the corresponding plant coordinates.
    - The calibration algorithm (OpenCV) runs to determine the intrinsic and extrinsic camera parameters using the corresponding sets.
    - The camera parameters are used to create a map which is saved for the current program run and is used to dewarp the images later obtained during operation of the plant also known as distortion compensation.

The camera calibration process is formulated as an optimization problem to minimize the root mean square error between the projected world coordinates to the sensor coordinates and the given sensor coordinates.

Each calibration image gives 165 points, since the checkerboard pattern is uniform, the locations of the corners can be easily computed if one knows the location of one corner on the board relative to the plant's coordinate system. In this work it was picked to be the top left corner.

Camera parameters are classified as either intrinsic parameters or extrinsic parameters.

Intrinsic parameters which are independent of the scene being viewed are also known as camera constants e.g.

- Focal length
- Pixels per mm (sensor)
- Distortion parameters
- Image center

Extrinsic parameters are dependent on the scene

- Rotational Matrix
- Translation between coordinate systems

5.6 Mathematical Background

The compact equation governing the transformation from world to pixel coordinates is as follows:

$$s\, m' = A[R|t]M' \tag{5.5}$$

Where 's' is the scaling, 'm' are the pixel coordinates, 'A' is the matrix of intrinsic camera parameters, 'R|t' is the rotational | translational matrix and 'M'is the world coordinate to be transformed.

The Detailed equation for transforming from world to pixel coordinates is as follows:

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{5.6}$$

$f_x$and $f_y$are the focal lengths expressed in pixel units, c_x and c_y is the principal point at the image center, u and v are the coordinates of the projection point in pixels and r_{xx} and t_x are the Rotational elements and translational elements respectively.

If no distortion is present, the rotation and translation would be expressed as follows with the division of z to return from the projective geometry's space.

$$\begin{bmatrix} x \\ y \\ x \end{bmatrix} = R \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + t$$

$$x' = \frac{x}{z}$$

$$y' = \frac{y}{z}$$

$$u = f_x * x' + c_x$$
$$v = f_y * y' + c_y \tag{5.7}$$

If distortion is present the transformation would be extended as follows:

$$\begin{bmatrix} x \\ y \\ x \end{bmatrix} = R \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + t$$
$$x' = \frac{x}{z}$$
$$y' = \frac{y}{z}$$
$$x'' = x' \frac{1 + k_1 r^2 + k_2 r^4 + k_3 r^6}{1 + k_4 r^2 + k_5 r^4 + k_6 r^6} + 2p_1 x'y' + p_2(r^2 + 2x'^2) + s_1 r^2 + s_2 r^4$$
$$y'' = y' \frac{1 + k_1 r^2 + k_2 r^4 + k_3 r^6}{1 + k_4 r^2 + k_5 r^4 + k_6 r^6} + 2p_2 x'y' + p_1(r^2 + 2y'^2) + s_1 r^2 + s_2 r^4$$
$$u = f_x * x'' + c_x$$
$$v = f_y * y'' + c_y \tag{5.8}$$

Where $\mathrm{k_x}$ are the radial distortion coefficients, $\mathrm{p_x}$ are the tangential distortion coefficients and $\mathrm{s_x}$ are the prism distortion coefficients.

The introduction of the distortion coefficients results in the intrinsic camera parameters entering the R|t matrix i.e. the matrix of extrinsic parameters. The resulting optimization problem is therefore nonlinear.

The Optimization parameters include the A matrix, the R|t matrix and the distortion parameters:$(k_1, k_2, p_1, p_2, k_3, k_4, k_5, k_6, s_1, s_2, s_3, s_4)$.

5.7 Results after Calibration

Below are the images before and after the distortion correction. On the left is the image before correction and on the right after the correction has taken place. See Figure 27.

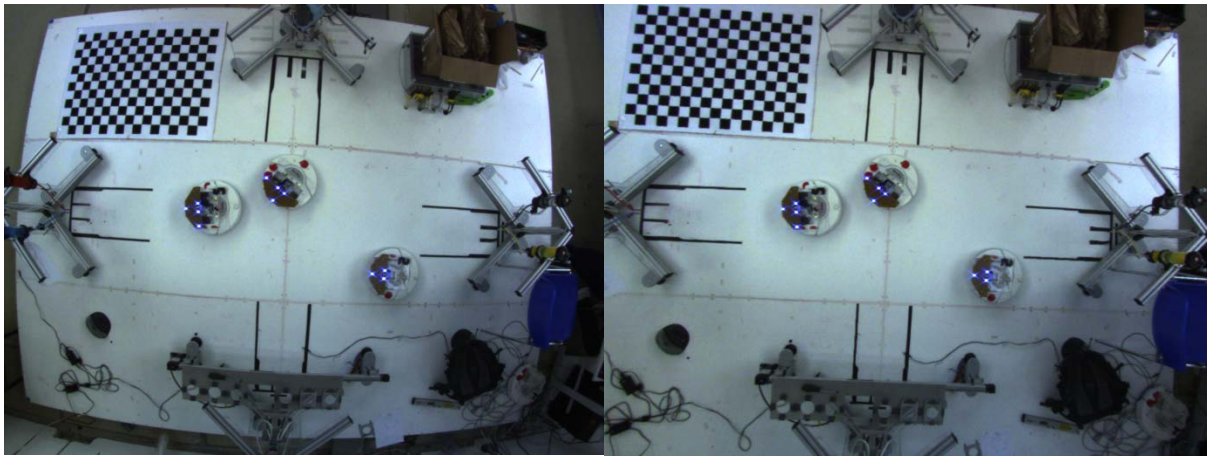


Figure 30: Before and after distortion correction

5.8 Localization using the camera after calibration

In general when obtaining images from a fixed point, the depth information is lost. Since we have a top view of the plant plane, the entire plant plane can be set to have Z=0 in the world coordinate system with respect to the camera. This is equivalent to removing the 3rd column of the R|t matrix making it square and invertible; Since R|t is of full rank. See equation5.5.

The final output is a reliable method of optically localizing the robots using the fish eye camera. The camera is currently the only feedback used for controlling the AGVs position.

5.9 Image processing and localization performance

- Average time required to run Image processing = 15ms
- Average time required to read Image from Camera = 100ms
- Average complete completion time = 115ms
- RMS error of mapping is 0.65 pixels
- Average angle noise $\pm$ 2 degrees

6. Battery exchange

Lusan Maharjan

The main goal of this task is to ensure that each AGV has enough battery level moving throughout the plant from the beginning to the end of the execution of the schedule and whenever one of the AGVs is running out of battery it will be exchanged with a fully charged one. The plant is operated on a time based system. All the AGVs should move according to the time based velocity profiles which are generated by the routing module. This means the robots are assumed to be at a specific place at a specific time. They should strictly follow the route provided by the routing module. If the AGV is out of battery, it stops and does not move anymore. . To avoid this problem the main control module replaces the low battery AGV with a full charged one before the AGV turns off.

For this procedure, the battery level of each AGV in the plant should be scanned. As soon as an AGV is detected with low battery level, some requirements for exchanging the AGV should be fulfilled. After the battery exchange requirements are fulfilled, the exchange process is performed.

6.1 Requirements for exchanging the AGVs

- The AGV with low battery should be either at its initial position or located at the storage station.
- If the AGV is at storage station, it must not carry any vessel.
- Filling or mixing process should not be in progress in any of the AGVs.

6.2 Getting the battery level

Awjoykanti Bonik

The voltage and the battery level can be checked by reading the sensors, as it was mentioned in chapter 2. There are different types of packet IDs for reading the battery properties. For example, the packet ID 21 is used to read the charging status, the packet ID 22 is used to read the voltage level, the packet ID 23 is used to read the electric current and packet ID 25 is used to read the battery level. Each robot is updating their voltage and charge level and they could be identified by meaning of their IP address; as mentioned in chapter 2, the updating time depends on the opcode we choose (*opcode 148* every 15ms

or *opcode 142* every certain chosen time or any other event). Therefore, the voltage level and battery charge level of each robot can be known.

Packet ID 22: Battery voltage level given in millivolts (mV). The voltage level decreases while the battery is depleting and increases while the battery is charging.

Packet ID 25: Battery charge level given in milliamp-hours (mAh). The charge value decreases when the battery is depleting and increases when the battery is charging.

Battery specification: 14.4 V (14400 mV) and 3.5 Ah (3500 mAh).

After the implementation of the microcontroller code, the voltages and battery charge levels are available on the screen.

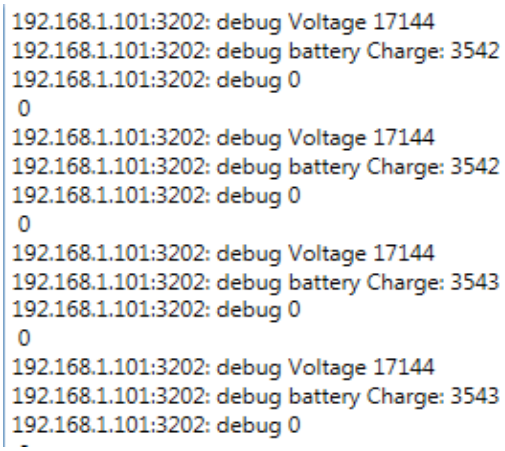


Figure 31: Battery level information

6.3 Implementation

Lusan Maharjan

During the normal operation of the plant, the battery levels of the operated robots are detected and the threshold level for the low battery notification (e.g. 30% of the full charge) is set, such that if the battery level of the operating robot is below the threshold level, the robot exchange function is called. The procedure for robot exchange is the following:

1. Make sure that the AGV with the low battery is not carrying a vessel. If the AGV is carrying a vessel then it should continue the operation until the

vessel has been released. This means that the exchange of the robots can only happen when the robot with low battery is, either at its initial positions or located at the storage station releasing the vessel for the hardening step. Also, check whether filling or mixing process is in progress. If the condition is true, wait for it to finish, then pause the operation of the plant.

2. Save the current state of the schedule i.e. the time at which the last task was executed and the corresponding task ID.
3. Using the camera, determine the positions of all the other AGVs. Considering these positions as obstacles, calculate the new path for exchanging AGVs using A* grid.
4. Using the A* class and the recently calculated grid, calculate two different paths for exchanging the robot with low battery level, with the AGV with fully charged battery at one of the charging stations. There are two charging stations, one charge station consists of full battery AGV and another is vacant. After the completion of the robot exchange, the vacant charged station will be occupied by low battery robot whereas another charged station will be empty.
5. Use feed forward control to drive the robots through the calculated paths and then use controller for the correction at the end.
6. Once the exchange of the robots is complete, send a notification to the main control routine, so that it can continue with the schedule.

6.4 Assumptions

- Fully charge battery always has fixed ID and it is located at one of the charge station.
- If an AVG is just in the storage, it does not carry any vessel. The AGVs in the other station carry vessels. Thus the low battery AGV can be exchanged only if it is in the storage.
- The exchange happens at the end of each group.

By taking these assumptions into account, it can be assured that all the requirements for the battery exchange are fulfilled if an AGV with low battery level is detected at the storage or at its initial position. This is because an AGV does not contain any vessel when it is at storage and the plant is in idle state

after the end of each group. Thus, there is no possibility of filling and mixing carried on other stations. This will be discusses in more detail while implementing it in the following section.

6.4.1 Implementation according to the assumptions

As mentioned earlier, the AGVs are going to be exchanged only if low battery level is detected in one of the AGVs and that the AGV is inside the storage at the end of each group. This means that the battery level of the AGV needs to be checked when it is inside the storage at the end of each group. The implementation will be done as follows:

1. Wait until the current group ends. Here, the group refers to collection of the movements of various AGVs for specific task. The combination of this entire group gives the complete operation of the plant.
2. Check whether the AGV is inside the storage station or not. If yes, check the battery level of the AGV.
3. If the battery level of the AGV is less than 30% of the full battery level, then perform an exchange of the AVGs. The steps involved in the movement of the AGVs during exchange will be described later.
4. Change the ID of the newly exchanged AGV with the ID of the AGV with the low battery level. This means that the fully charged AGV ID will be equivalent to the low battery AGV ID.

The steps involved in the movement of the AGVs during exchange are listed below.

1. Determine the current location of the robots (position and orientation)
2. Determine two collision free paths using A*,
    a. One from the current location of the low battery robot to the empty charge station
    b. Another from the location of the fully charged robot at the charge station to the current position of the low battery robot
3. Update the next intermediate positions as the temporary end points
4. Calculate the angle difference between the current point and the next point and then turn the robot for desired rotation

5. Calculate the distance and then calculate the velocity and move for an specific time(time can be set manually by user)
6. Update the new intermediate position as end point and repeat these process until it updates last point as its end point
7. Uses controller for correction

6.5 Result and Conclusions

During the normal operation of the plant, the low battery AGV was detected at the storage station without carrying any vessel. Initially, the full battery AGV was placed at charge station 1 and charge station 2 was empty. After that, the remaining two operating AGVs in the plant were detected and their positions were marked. The plant is currently in the idle state as it is in the end of one of the group. This means that there is no filling, no mixing and no movement of the AGV going around the plant at the moment. So, the schedule can be paused at that moment. Thus, the AGV exchange is carried out as all the requirements are fulfilled. The position of the low battery AGV is sent as the source point, the location of the charging station 2 which is currently empty as the destination and the positions of the two other robots as obstacles as the input parameters for the A* and then A* generates the new path for moving the AGV. Similarly, the position of the low battery AGV is sent as the destination point, the location of the charging station 1 where an AGV with full battery is supposed to be as the destination and the positions of the two other robots as obstacles as the input parameters for A* and another path is generated by. Then the AGV travels through all the points generated by the A* simultaneously using the feed forward control (only for translation whereas it still uses controller for rotation). Once the robot travels through all the points generated by A*, then finally a feedback controller is applied for final correction. This way, the battery exchange process is carried out between the two AGVs. After this, the plant can resume the schedule again so as to complete the remaining process.

6.6 Remark

A* calculates the path in centimeters which means that the list of intermediate points provided by A* is in cm. But the camera parameters are calculated in millimeters. Thus, a conversion of those intermediate points from A* is needed multiplying by10 if we need to use these value with reference to camera values. Similarly, the camera values should be divided by10 when calculating

the A* path. Thus, conversions from millimeter into centimeters and vice-versa are needed.

7. Integration

Angel Garza, Mauricio Martinez, Shehabeldin Abdelgawad

7.1 Flowchart

To produce the final product, the main program which controls the operation of the plant has to go through the following stages:

1. The user inputs the set of recipes to be produced.
2. The user specifies the AGVs available and the storage locations that are free.
3. As previously discussed in chapter3, the scheduler module receives a list of available AGVs, free storage locations and recipes. The scheduler then assigns the recipes to the AGVs and schedules the operations of the stations to produce the final products.
4. The router receives the output from the scheduler and finds collision free paths for each group of movements, by producing a velocity profile for each AGV, see chapter 3.
5. Start the execution phase.
6. Create the final products.
7. End of the execution.

The Execution phase iterates through the groups, each group's iteration is divided into 2 other phases, the "Feed-forward" and "Feedback" phases.

In the Feed-forward phase, the AGVs follow the velocity profiles in an open-loop sense with respect to the plant. The robots receive the velocity values to be followed from the main program on the PC. The velocity is regulated by a built-in low level controller on the AGV. However, as previously discussed in chapter 2, the encoders do not provide sufficient accuracy for the rotation and the routing module assumes zero time for rotations, see chapter 3, therefore all the feedback information which is required for checking the rotations are taken from the camera feedback.

In the feedback phase, the AGVs are controlled using the camera feedback through the control module. This is done iteratively until the desired pose is achieved. A single iteration contains the following steps:

1. The camera captures an image.
2. The Image processing module extracts the position and orientations of the AGVs in the plant coordinate system.
3. The controller receives the current pose of the AGVs from the Image processing module and the desired pose from the routing module, and then calculates the appropriate velocities for each AGV.
4. The controller sends velocity commands to the AGVs.

The execution of a single group of movements is illustrated by the flowchart below:

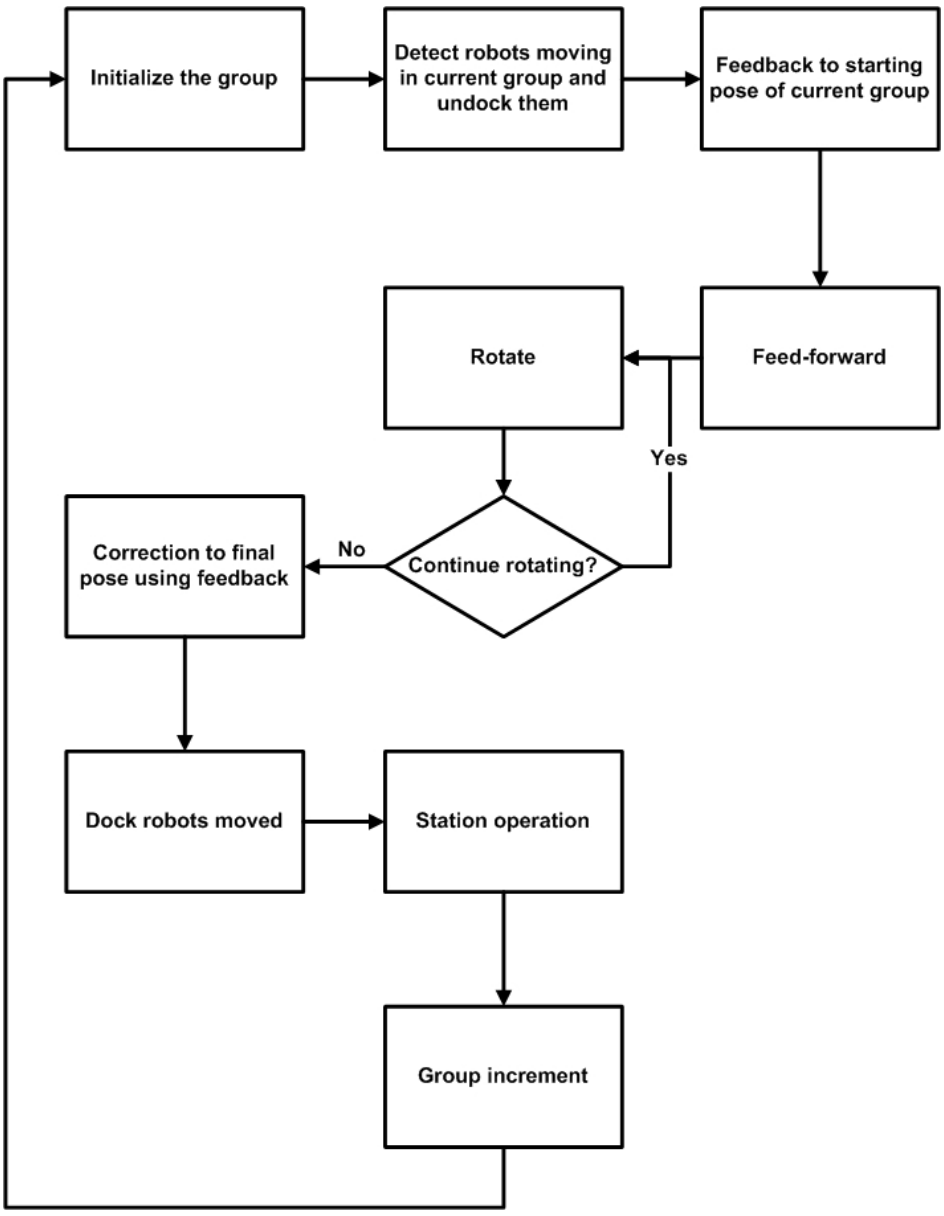


Figure 32 Flow chart

The plant proceeds through the following steps for each group:

1. Undocking if needed for AGVs that will move in the current group
2. Initial correction using camera feedback to starting point of current group
3. Use feed forward control for all AGVs of the group
4. For any rotation detected, stop all feed-forward and start control loop using camera feedback to obtain required orientation
5. Continue feed-forward control which ensures collision free movements
6. At the end of the active group, run the controller to correct the final pose of all AGVs in the current group
7. Dock

The use of feed-forward in conjunction with feedback ensures collision free movements, the feed-forward avoids collisions via the A* algorithm but does not account for rotation times or velocities associated with rotations. The rotations are taken care of by the controller using camera feedback.

8. Conclusion and further work

Angel Garza, Mauricio Martinez, Shehabeldin Abdelgawad

The plant can currently run autonomously which is an improvement to the plant's functionality. A reliable controller module was developed and the image processing algorithm was reworked, the current implementation achieves faster and more reliable results. However, the required time for execution is far from optimal, this is due to:

1. The routing module does not account for the time needed for AGV rotations
2. The AGVs move following a path with the sequence rotation-translation-rotation, paths and velocities are pre-calculated using this kind implementation.
3. All the events are time based and not event based, this is not realistic or safe and to decrease the chances of station mishaps a large amount of wait time is introduced into the scheduler.

The plant can be improved by, removing the restriction on the AGVs with regards to the rotation-translation-rotation movements and allowing curved paths.

Another improvement would be the use of events to trigger station operation (such as robot bumpers) instead of time based operation, thus giving more flexibility to execute the schedule and manage errors. The execution should be implemented in a separate thread to allow GUI updates. Implementation of an algorithm that would generate a collision free path based on camera feedback in conjunction with event based station operation, would remove the time based constraint on the system.

9. References

[1] Group project report on **"Advanced AGV Drive Control in a Pipeless Plant",** TU Dortmund, 2013

[2] OpenCV Documentation, www.opencv.org

[3] Emgu Documentation, www.emgu.com

[4] **"Computer Vision: A modern Approach",** by David A. Forsyth, Jean Ponce

[5] F. Hoffmann. **"Computer Vision in Automation & Robotics"** script, TU Dortmund, 2014

[6] **"Springer Handbook of Robotics"** by Siciliano, Khatib

[7] R. Craig Coulter, **"Implementation of the Pure Pursuit Path Tracking Algorithm"**, the Robotics Institute Carnegie Mellon University, Pittsburgh, Pennsylvania, 1992

[8] Michael O'Connor, Thomas Bell, Gabriel Elkaim, Dr. Bradford Parkinson, **"Automatic Steering of Farm Vehicles Using GPS"**, Stanford University, 1996

[9] Junfeng Wu, Wanying Zhang, Shengda Wang, **"A Two-Wheeled Self-Balancing Robot with the Fuzzy PD Control Method"**, College of Automation, Harbin University of Science and Technology, China, 2012

[10] **"Industrial Robotics: Theory, Modelling and Control, Chapter 4, Robot Kinematics"**, by S. Kucuk, Z. Bingul

[11] **"Introduction to Autonomous Mobile Robots"** by R. Siegwart, I. R. Nourbakhsh

[12] Adam Finkelstein, **"Kinematics & Dynamics"** script, Princeton University, 2005

[13] L. Gracia, J. Tornero, **"Kinematic Control of Wheeled Mobile Robots"**, Latin American applied research, v.38 n.1 Bahía Blanca ene. 2008

[14] Kurt, Tod E. "**Haking Roomba"**. Indianapolis, Indiana: Wiley Publishing, Inc., 2007

[15] **“iRobot Create Open Interface”**. iRobot. Web.<http://www.irobot.com/filelibrary/pdfs/hrd/create/Create%20Open%20Interface_v2.pdf>

[16] Nazari, Shaghayegh. **“High-Level Hierarchical Control of a Miniature Pipeless Plant with Mobile Robots”.** Master thesis, TU Dortmund 2013