PONTIFICIA UNIVERSIDAD JAVERIANA BOGOTA
ELECTRONIC ENGINEERING DEPARTMENT

POLITECNICO DI TORINO
III FACOLTÀ INGEGNERIA DELL'INFORMAZIONE

Research and development of robots cooperation and coordination algorithm for space exploration

Master of science in Mechatronic Engineering Thesis

Relatore
Prof. BASILIO BONA

Laureandi
JUAN FELIPE CABEZAS VALENCIA

JUAN FELIPE MATHEUS GOMEZ

2010-2011

TABLE OF CONTENTS

TABLE OF FIGURES

1. INTRODUCTION

Over the past several years, there has been a growing interest in the Mars exploration. Different approaches have been developed in different countries aiming to a possible human mission to Mars. The first step in the reach of this scope is the un-manned exploration what means the use of robots.

Until now there has been many un-manned missions to Mars (Sojourner – Mars Pathfinder, Spirit and Opportunity –Mars Exploration Rover amongst others), some successful and many others failed. All of them with the idea of explore Mars in different ways using one or more specific robots (orbital, lander or rover). In the last years the idea of using two or more robots for this task became stronger, since a group of robots can make simultaneous observations of the same phenomena from different perspectives.

1.1. SWARMS

In the research of robot teams arises the concept of SWARMS, a biological inspired model in which a group of robots behave like a colony of insects. This group is composed by a collective of simple, autonomous individuals that can achieve complex tasks using basic interactions and a decentralized control system.

1.2. A real mission using cooperating robots (Thales Alenia Space's TBRA)

ESA actual mars exploration projects now require a higher level of autonomy and adaptation in the robots, given the complexity of the new objectives for the future missions. Aware of this Thales Alenia Space with the cooperation of the University of Genoa has developed a project called Test Bench for Robotics and Autonomy (TBRA). The project focuses on the design and development of a modular and flexible software architecture, which eases the implementation and testing of innovative Guidance Navigation and Control (GNC) algorithms. In this architecture, each module can be regarded as a standalone entity with standardized interfaces to communicate with other modules and implementing a specific GNC task

During the last year TBRA software infrastructure has been deployed on the target hardware, consisting of a robotic platform based on commercial hardware (rover platform and GNC sensors) and the focus moved to the development of the functional modules.

The 2010 output of the TBRA R&D project is a rover capable of both indoor and outdoor autonomous navigation relying only on its onboard GNC sensors. TBRA has a 3-eyes stereo camera and 3D Laser Scanner which can be used independently or together to scan the rover surrounding in order to reconstruct the Digital Elevation Map of the environment. Rover localization is provided by the fusion of the data coming from the Visual Odometry, Wheels Odometry and Inertial Odometry subsystems. The Traversability GNC module, taking into account the rover platform constraints (slopes and discontinuities threshold values) generates the navigation map used for the Path Planning process. Then the locomotion module is in charge of executing the planned path and react to hazard detections. These medium and low-level actions are coordinated by a Navigator Module, which is in charge of executing the commands coming from the high-level autonomy module, which is in charge also of the re-planning of the mission in case of failures.

The TBRA Robotic Platform is composed by:

- MobileRobots® PowerBotTM Platform, which can be divided in:

    - PowerBotTM Base (including wheel encoders, sonar arrays and batteries)
    - Processing Units (Internal and Auxiliary PCs) and Network Hardware

- GNC Sensors and Actuators

    - Inertial Measurement Unit (IMU)
    - Inclinometers
    - Laser Scanner (including continuously rotating pan unit)
    - 3-Eyes Stereo Camera
    - Pan-Tilt unit (PTU)
    - 2-Eyes Stereo Camera

The Inertial Sensing System is a Systron Donner MotionPak® II: a solid state 6-dofs MEMS used for measuring linear accelerations and angular rates. It provides both 12 analog and 1 digital (RS232 DB25) outputs at a data transfer rate of (maximum) 32Hz.

The inclinometers are 2 Swiss Precision Instruments PRO 3600 with 0,01 [deg] resolution. They provide one RS232 (nonstandard connector) output at 1.87Hz (533ms).

The 3D laser scanner is a Sick LMS200 mounted on a Maxton continuously rotating pan unit. The Laser Scanner has a resolution of 0.25 [deg] on rotation and an accuracy of 1 [mm] up to 30m.

The 3-eyes stereo camera used for the visual perception tasks (3D map reconstruction from stereo vision) is a PointGrey Research Bumblebee XB3. This camera is equipped with 3 Sony Color CCDs (1/3") with a maximum resolution of 1280x960 pixels at 15 frames per second. Each unit has a 70 [deg] Horizontal and 50 [deg] Vertical Fields of view. The camera is factory pre-calibrated to have a 0,1 pixel RMS error.

The Pan-Tilt unit is a Directed Perception D46-70 with embedded controller. It is used to actuate the Bumblebee XB3 during perception in order to allow the inspection of the rover surroundings (360 [deg] motion) for 3D map reconstruction. The PTU provides a maximum resolution of 0.003 [deg] both on Pan and Tilt axes and a maximum speed of 60 [deg/s].

The 2-eyes stereo camera used for the Visual Odometry is a Point Grey Research Bumblebee 2. This camera is equipped with 2 Sony Color CCDs (1/3") with a maximum resolution of 1024x768 pixels at 20 frames per second. Each unit has a 97 [deg] Horizontal Field of view. The camera is factory pre-calibrated to have a 0.1 pixel RMS error.

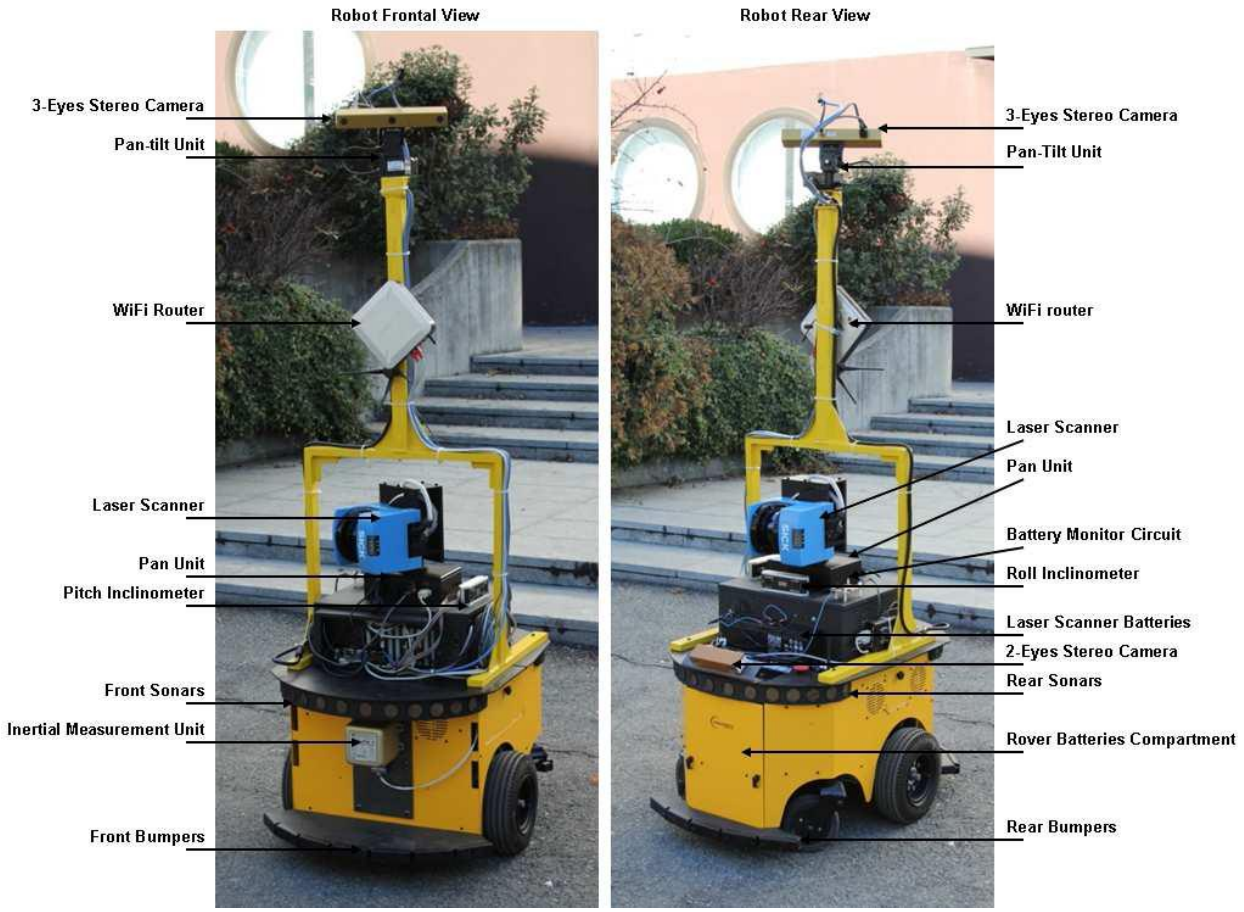


Fig. 1. Robot Overview

The architecture of the TBRA is show in the next figure:

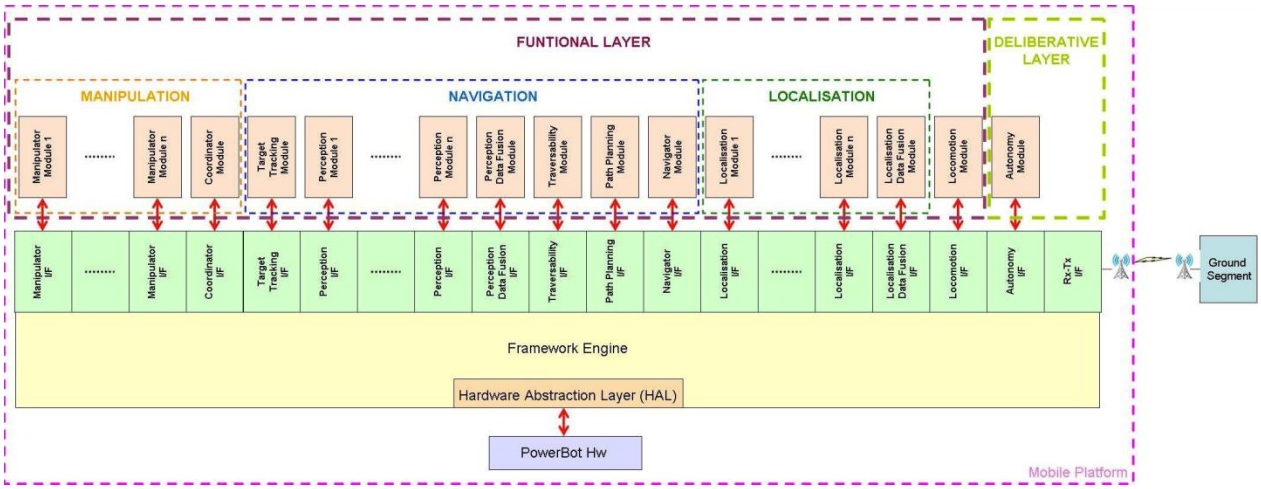


Fig. 2. TBRA Architecture

The Framework is the core of the complete system. It provides all the necessary interfaces (TM/TC) between subsystems and coordinates all the rover activities.

The main functions of the Framework are:

1) Provide interfaces to all the external functional modules
2) Provide an abstraction of the Powerbot HW (HAL)
3) Parse/interpret/route the commands between subsystems
4) Provide an interface to the "Ground Station PC"
5) Implements all the control logic needed to respond to the commands coming from each module

The TBRA Robotic Platform has the next modules that compose the architecture:

- Locomotion Module
- Localization Module
- Localization Data Fusion Module
- Perception Module
- Perception Data Fusion Module
- Traversability Module
- Path Planning Module
- Navigator Module
- Target Tracking Module
- Manipulator Module
- Coordinator Module

Thales Alenia Space for the development of the cooperative module has foreseen the buy of a Pioneer 3-AT platform from MobileRobots equipped with Inertial Unit, Omni-camera, PTU and a 3D TOF Camera (Show in figure 1).

As the TRBA project went forwards, more elaborated ideas of the possible applications and uses of the main robot (TBRA robotic platform), other secondary robots (Pioneer 3-AT) and the technologies developed, started to emerge and a space mission was proposed.

This mission consists in sending to Mars a group of heterogeneous robots (whit a centralized control) composed by a "master" robot more complex than the others (the ruler) and a group of simple "slave" robots, whit the scope of explore in a coordinated way some interesting areas of the planet, sample collection and transport of the samples to the Lander ship.

Fig. 3. Pioneer 3-AT

1.3. Our thesis

The technics and methods developed in this thesis are going to be focused on solving the problems of relative localization amongst robots and their movement maintaining a predetermined formation. To do the tests, it is going to be developed a coordinator module and a target tracking module; as a part of the TBRA (described in "*A real mission using cooperating robots"*) and all the technics and methods are going to be implemented in C language. The developed modules are going to be tested using the two robots previously described, the TBRA Robotic Platform and the Pioneer 3-AT.

2. OBJECTIVES AND CONTEXT

This thesis is going to be developed in collaboration with Thales Alenia Space Torino with the general objective of considering the possibilities of a planet exploration mission using a group of robots working in a cooperative way, that can acquire information about the presence of water or any kind of life at the present or in the past in an unknown planet; and also that can give information about the conditions of the planet, aiming to possible manned missions.

As said before, Thales Alenia Space with the cooperation of the University of Genoa has developed a project called Test Bench for Robotics and Autonomy (TBRA) described previously. The specific objective of this thesis is the development of two of the TBRA modules: The coordinator module and the target tracking module.

The task of the coordinator module is to provide a communication system among robots that make the group behave in a coordinated way. As one of these behaviors, the movement in formation of the robots will be developed in a close relation with the Path planning module previously developed in Thales Alenia Space.

The task of the target tracking module is to, using the 3-eyes stereo camera Point Grey Research Bumblebee XB3, mounted on the PowerBotTM; determine the relative position of a specific object. In this thesis a red sphere mounted in the slave robots is going to be used as a 3D target to determine the position of the robots respect to the Master robot.

As a part of the TBRA project, there are two robots, the PowerBotTM Platform and the Pioneer 3-AT, both described previously, that are going to be used as our heterogeneous team of robots with a centralized control, in which the PowerBotTM Platform plays the Master role and the Pioneer 3-AT plays the slave role.

Even when all the tests will be executed using only two agents, all the solutions proposed in this thesis are projected to function properly also with bigger groups of robots. The programs developed are meant to be flexible and scalable.

The robots are equipped with Windows and Linux OS. The programming language in which the programs are going to be written is C++.

The robots will communicate with each other using pc modems and routers, and are going to be identified using their IP addresses.

Because the modules are projected as a part of a space mission, all programs are going to be developed to work as follows: the master receive one specific order from earth and send the corresponding orders to the slaves. In the cooperative mode the slaves only receive form earth, the order to go in cooperative mode. Received this command, the slaves will only communicate with the master.

The reference system for each robot is centered in its own initial position, that's why in the robots interaction some transformation matrices are going to be needed.

All distances are measured and calculated in meters and the angles in radians.

The tolerance range for the distances is 20cm.

Most of the other modules of the TBRA are working already and the modules proposed in this thesis had to be developed to work under the conditions of the modules previously projected.

3. RELATIVE POSITION ESTIMATION OF A SPECIFIC OBJECT: TARGET TRACKING MODULE

To solve the relative position and orientation problem, it is going to be done an images processing approach as a part of the target tracking module.

The specific objectives of the module are:

- Find a target (object whit a specific color and shape).
- Determine the position ***x***, ***y***, ***z*** of the target.

The device that is going to be used for this purposes, is the stereo camera Point Grey Research Bumblebee XB3.

The camera is mounted in a PTU (Pan-Tilt Unit) shown in fig. 4 that makes the camera able to move with two degrees of freedom.

Fig. 4. Pan-Tilt Unit

The first thing done to develop the module is to set the stereo camera to percept the depth.

This task is performed due the stereo vision technology that allows range measurement using triangulation between slight offset cameras. The camera generates three images that are digitalized and stored. The software system analyses the images and establishes correspondence between pixels in each image. Based on the geometry of the camera and the relation between images, it is possible to determine the depth in a specific part of the captured scene.

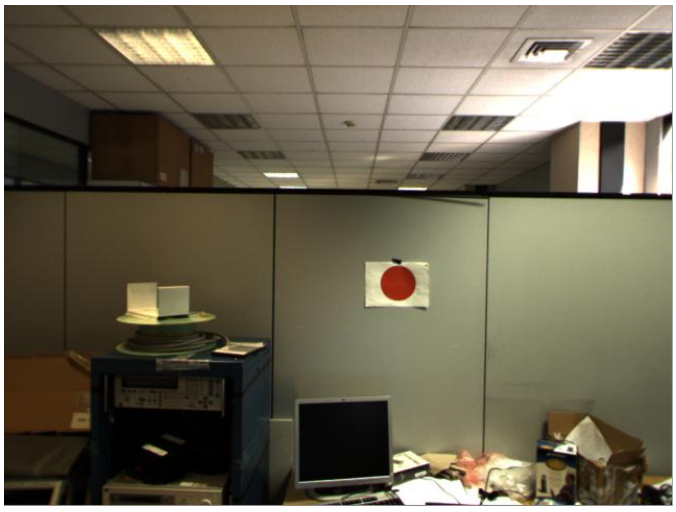


Fig. 5. Original image taken in laboratory

Fig. 6a. Left perception

Fig. 6b. Right perception

The figures 6a and 6b show two quiet similar images taken in laboratory. Closer inspection reveals a shift between closer objects and those that are further away. Based on the amount of shift, the system is able to determine the distance of the objects in the scene.

After the images processing, two additional images are generated, a stereo vision image and a depth image. In the depth image sown in fig. 7, closer objects are represented with brighter shades while objects further away are represented with darker shades. In fig. 8 the stereo image is shown.

Based on these two images the system estimates the depth of the scene.

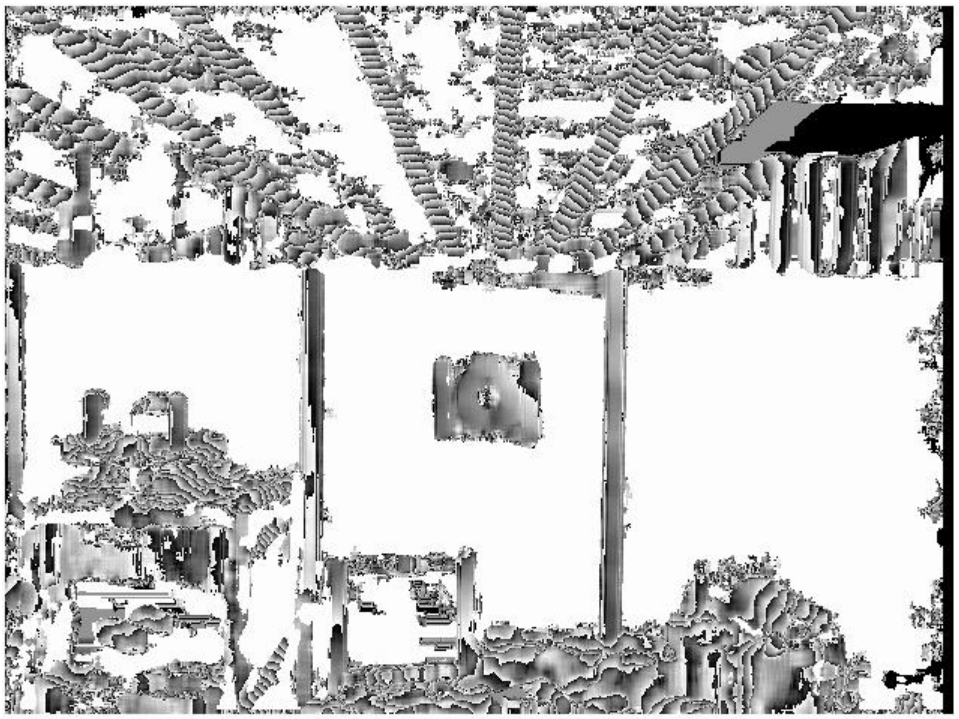


Fig. 7. Depth perception

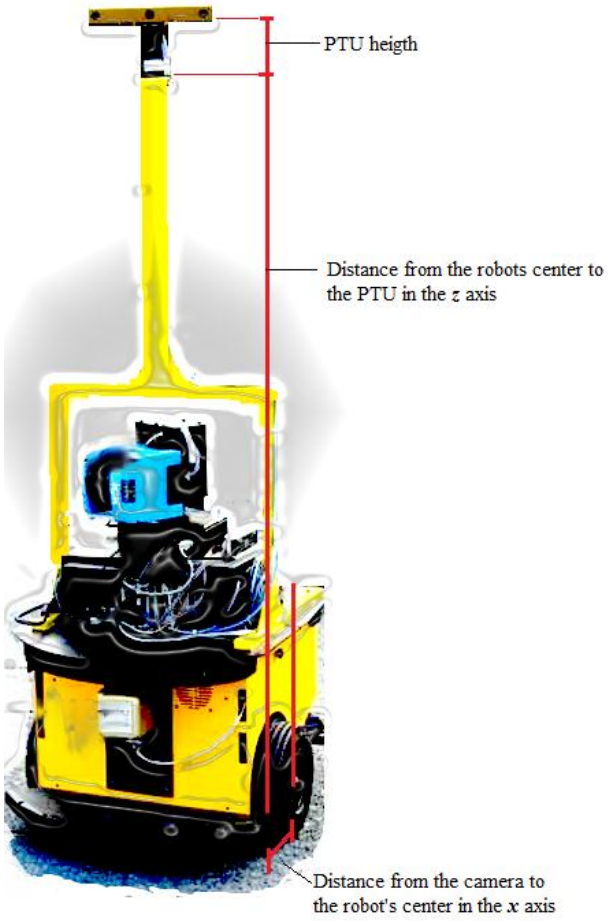
Fig. 8. Stereo image

In our context, measuring the depth is obtaining the ***x***, ***y*** and ***z*** coordinates of a determined part of an image, what means, we can obtain the position of a desiderated object respect to the camera (in this case the red circle on the wall in front). Once the camera has been mounted in the robot, the position of any object can be obtained in a reference system centered in the robot.

The reference frame of the robot is placed on the middle wheels axis as show in fig. 9b. Because the camera is not in the center of the reference frame, to obtained the coordinates ***x***, ***y*** and ***z*** of the tracked object, some transformation matrices have to be calculated based on the position of the camera respect to the center of the robot.

The robot structure and the position of the camera respect to the robots frame are shown in fig. 9a. The robot's reference frame, stereo-camera's reference frame and stereo-camera's structure are shown in fig. 9b.

The reference frame of the stereo camera is set in the left individual camera (fig. 9b) that is displaced 12cm in the ***y*** axis. Also the reference frame of the camera is rotated respect the reference system of the robot as seen in fig. 9b.



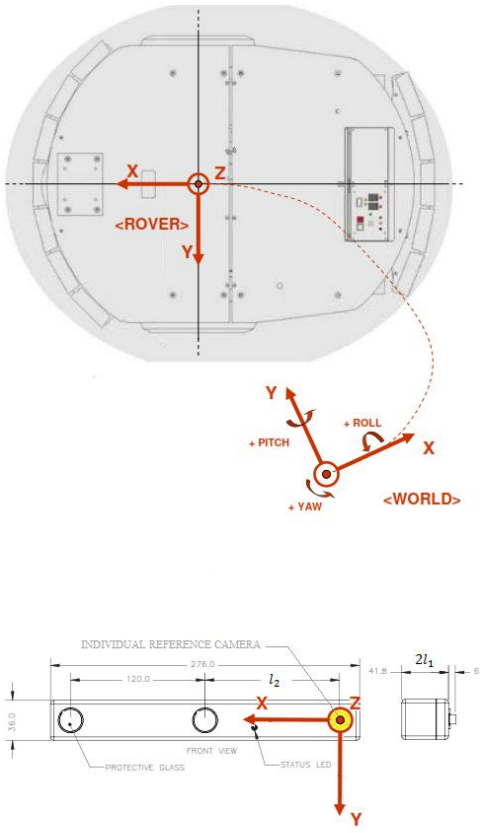


Fig. 9a. Camera position respect to the reference frame

Fig. 9b. Robot's and stereo-camera's reference frames

To calculate the relative transformation matrices, we set the reference systems as shown in fig. 10 where:

- R_c is centered in the individual camera's lens.
- R_4 is centered in the individual camera's center.
- R_3 is in the stereo-camera's structure center.
- R_2 in in the top of the mast.
- R_1 is in the base of the robot and R_0 is the robot's reference frame.

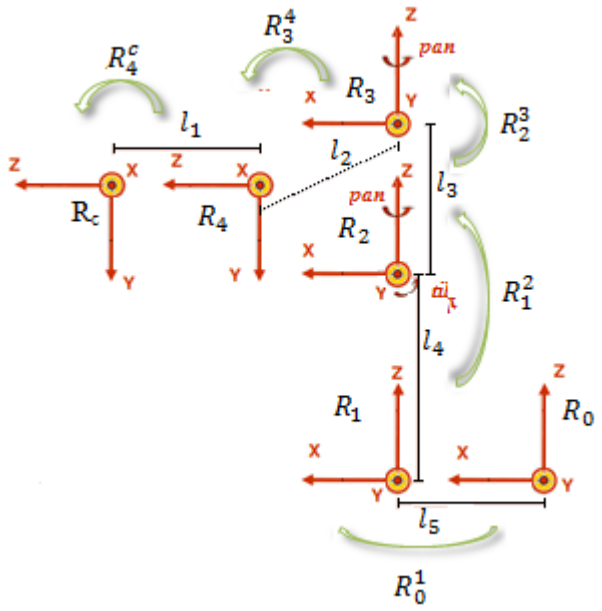


Fig. 10. Transformation from camera coordinates to robot coordinates

The distances shown in figure correspond to:

- l_1 **=2cm** is the distance between the individual camera lens and the center of the individual camera. This distance is shown in the stereo-camera's structure in fig. 9b.
- l_2 **=12cm** is the distance between the reference camera center and the center of the stereo-camera structure. This distance is shown in the stereo-camera's structure in fig. 9b.
- l_3 **=5cm** is the distance between the stereo-camera's structure center and the tilt joint of the PTU.
- l_4 =**1.71m** is the distance between the tilt joint of the PTU and the base of the robot (mast +PTU base).
- l_5 **=1cm** is the distance between the robot base and the robot's reference system.

The transformation matrices shown in fig. 10 are set as follows:

$$T_1^0 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0.02 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$ Corresponding with l_1

$$T_2^1 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ -1 & 0 & 0 & 0.12 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$ Corresponding with the camera orientation and l_2

$$T_3^2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0.05 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$ Corresponding with l_3

$$R_{t_3}^3 = \begin{bmatrix} \cos(tilt) & 0 & \sin(tilt) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(tilt) & 0 & \cos(tilt) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$ Rotation corresponding with the tilt angle

$$R_{p_3}^{\ 3} = \begin{bmatrix} \cos(pan) & -\sin(pan) & 0 & 0 \\ \sin(pan) & \cos(pan) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$ Rotation corresponding with the pan angle

$$T_4^3 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1.71 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$ Corresponding with $\boldsymbol{l_4}$

$$T_C^4 = \begin{bmatrix} 1 & 0 & 0 & 0.01 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$ Corresponding with $\boldsymbol{l_5}$

The final resultant transformation will be $T_C^0 = T_C^4 \cdot T_4^3 \cdot R_{p_3}^{\ 3} \cdot R_{t_3}^{\ 3} \cdot T_3^2 \cdot T_2^1 \cdot T_1^0$

Once the depth perception is done, the next step is to set the camera to identify a target.

The object that is going to be set as a target is a colored sphere that will be mounted in each slave robot. Due the symmetry of the sphere, the shape seen from the master robot will be circular no matter the heading angle of the slave robot, what will make easier the identification of the target.

The color of the sphere must be contrasting with the environment in the way the robot can be easily differentiated from a similar object that could be found in the exploring area. Also the color can be used to identify a slave robot from another one.

For the development of this thesis a red sphere is going to be used as the target. The resolution of the image will be set to 640x480.

3.1. General description of the algorithm

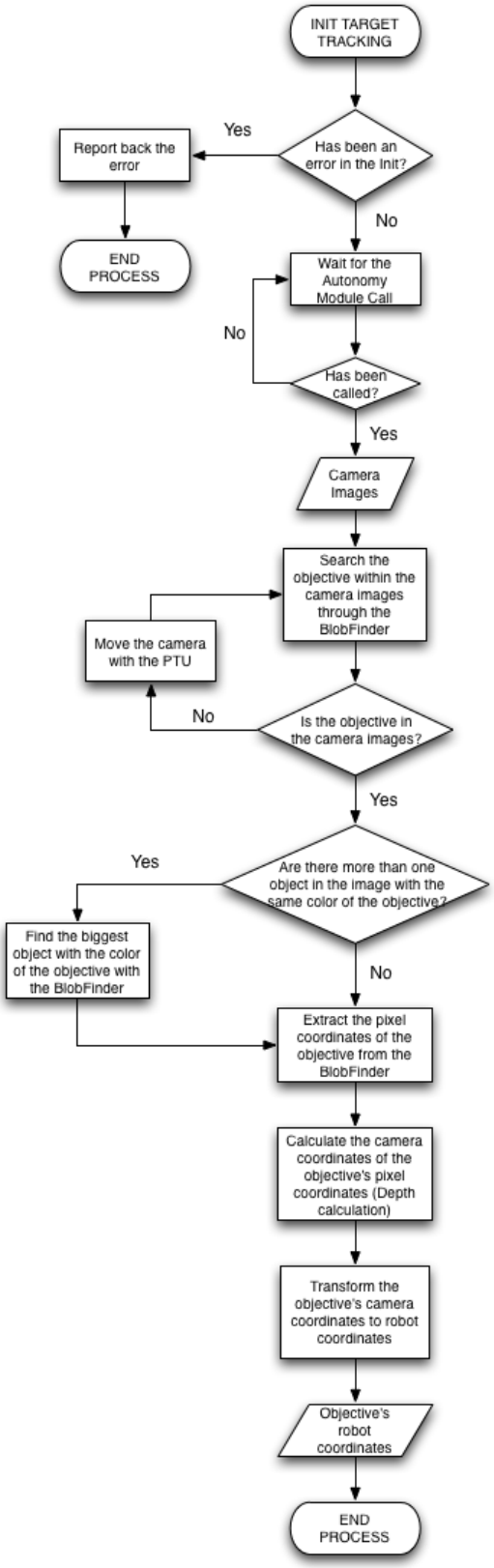


Fig. 11. Target tracking flow diagram

1. Connect to the TBRA and initialize the BUMBLEBEE XB3 camera
2. Wait for the AUTONOMY_COORDINATE call
3. Search with the camera, around the master robot, with the PTU to find shapes with the color to be tracked (in this case red)
4. Calculate the dimensions of the shapes found (the size of their area in the image)
5. Locate the center of the biggest shape with the color to be tracked
6. Calculate the depth of this point and their surroundings to make a mean
7. Transform the coordinates obtained from the depth calculation using the mentioned transformation matrices (from camera coordinates to robot coordinates)
8. Report back the coordinates

3.2. Specific description of the program

The target tracking program can be divided in the following parts:

3.2.1.Initialization

In this part the module connects with the TBRA, and also connects with the BUMBLEBEE XB3 camera. If the module can't connect with one of them it would generate an error and will inform about the problem.

If the connection process is successful the program will proceed with the camera initialization that includes different functions calls already developed by Point Grey Research, that come with the software development kit libraries, included in the camera set.

These functions calls determine a series of hardware image preprocessing, different camera and image output configuration parameters, such as the maximum disparity value, the image resolution, the type of camera configuration, etc., and software image processing.

Also the OpenCV (**Open** Source **C**omputer **V**ision) library is used through this program, and in this case, for the initialization stage of the module, is used to display in the computer screen, from which the target tracking program is being executed, what the camera is "seeing", verifying in this way that the camera has been correctly initialized.

In this part the transformation matrices are defined using the TMATRIX variable type that stands for "transformation matrix". This format or variable type, allows executing the different coordinate transformations in an easy and fast way using the function `TmatrixByTmatrix(Tmatrix1,Tmatrix2,&Tmatrixout)`. The definition of the variable type and the function is included in the MatrixLib.h and MatrixLib.cpp files developed by the Thales Alenia Space TBRA team.

The TMATRIX variable type is designed to work like a roto-traslation matrix, for example the definition of the T_1^0 is showed in the following piece of code:

```
//R0TR1 From camera len (R0) to camera center(R1)
//1st column
      R0TR1.trmat[0] = 1.0;
      R0TR1.trmat[1] = 0.0;
      R0TR1.trmat[2] = 0.0;

      //2nd column
      R0TR1.trmat[3] = 0.0;
      R0TR1.trmat[4] = 1.0;
      R0TR1.trmat[5] = 0.0;

      //3rd column
      R0TR1.trmat[6] = 0.0;
      R0TR1.trmat[7] = 0.0;
      R0TR1.trmat[8] = 1.0;

      //4th column
      R0TR1.trmat[9] = 0.0;
      R0TR1.trmat[10] = 0.0;
      R0TR1.trmat[11] = 0.02;
   //End R0TR1
```

$$T_1^0 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0.02 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Where the 1st, 2nd and 3rd columns represent the rotation part of the matrix and the 4th column represents the traslation part of the matrix.

Done this, the module awaits the request from the autonomy module to start searching and then tracking the objective.

3.2.2.Search and Track

Once the module receives the order from the autonomy module to track, the program will start with the process of searching the objective in the images obtained from the camera. The process of search ends when the BlobFinder function, that will be explained later, reports that a *blob* has been found (that in theory would be the objective)

To find the objective's camera coordinates both Point Grey Research Software Development kit and OpenCV libraries are used. The first thing to do was to determine the hue value of the objective to track in the image, in this case a red object.

To do this, through the window created from the initialization of the target tracking module, that shows the images from the camera as seen in fig. 5, using the function `cvSetMouseCallback`, clicking on the objective in the image displayed in the window (red circle in fig. 5), the pixel coordinates of this point are extracted and with the function cvGet2D the HSV (Hue, Saturation and Value) numbers of the point are obtained.

The previous process has been done several times and a hue mean value has been calculated. After this point the obtained value will be used for searching the objective in the image and the functions mentioned above, will not be used anymore in the final version of the module.

After obtaining the objective's hue value, it has been used to set an upper and lower hue limit (the hue value can change a little due to the camera, the environment light, etc. for this reason a range of hue has been created), that will be used to search in the image the objective comparing every image pixel hue value with these limits.

To do these comparison a monochromatic copy of the original image is done, and if a certain pixel of the original color image has a value between the upper and lower hue limit, the equivalent pixel of the monochromatic image will be filled with white, and in the case that the pixel of the color image is outside the hue limits the equivalent pixel of the monochromatic image will be filled with black as shown in fig. 12a. and fig. 12b.

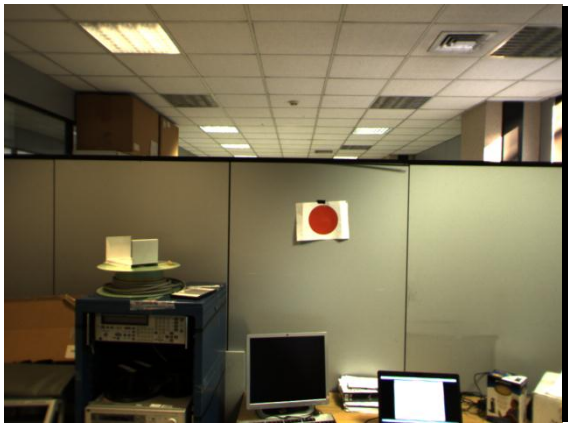


Fig. 12a. Color image

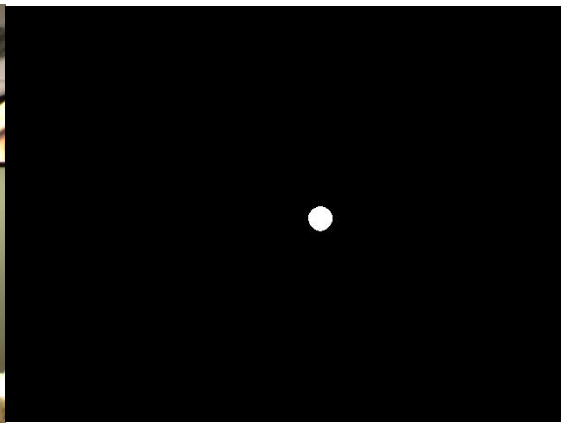


Fig. 12b. Monochromatic image used by the blob finder

With this monochromatic resultant image shown in fig. 12b, through the function FindBlobs and GetBlobs, included in the cvBlobFinder.h and cvBlobFinder.cpp, the blobs contained in the monochromatic image are identified and in a `CvBlob` dynamic array useful information is saved of each one, which include the *x* and *y* pixel coordinates where the blob starts, the width and height of the blob and its area.

Now that the blobs in the image are identified, the biggest blob is extracted from the array because, in theory, inside the area of this blob is the objective to be tracked (that's the reason why the color of the objective must be contrasting with the environment, otherwise any other bigger object with the same target's color in the environment could be confused as the objective)

Once the biggest blob has been identified, calculating the center of the blob is really easy, because the width and height are known. Knowing the pixel coordinates of the center of the biggest blob, with the function `triclopsRCD16ToWorldXYZ`, included in the Point Grey Research Software Development kit libraries, the camera coordinates of the center of the objective are extracted from the image (depth calculation).

To calculate a more accurate value of the coordinates of the objective a mean value is calculated, using also the XYZ values, extracted with the same function, inside a 10 pixel x 10 pixel area around the center of the objective.

3.2.3. Transformation of Coordinates and Report Back

At this point the objective's camera coordinates are known and the program proceeds with the transformation of this coordinates to robot coordinates. To do this the product $T_C^0 = T_C^4 \cdot T_4^3 \cdot {R_p}_3^3 \cdot {R_t}_3^3 \cdot T_3^2 \cdot T_2^1 \cdot T_1^0$ is calculated and the camera coordinates are transformed into robot coordinates through the roto-traslation matrix calculated.

The final step is to set the new target tracking module status with the function `SetStatusData`, in which the objective's robot coordinates and other status variables are saved and can be read by any other module connected of the TBRA through the function `GetTargetTrackingStatusData`.

4. COMMUNICATIONS

Each robot is equipped with a modem that gives an IP address that is going to be used for the identification of each robot. Also the robots are equipped with common routers that permit the connection to a wireless network (LAN).

The information that is going to be shared from earth to robot and from robot to robot is:

- Orders from Earth
- Orders form the master
- Relative positions of the robots
- Information regarding the success or fail of a task

To make the robots behave in a cooperative way, the first thing that is needed is a communication protocol.

4.1. Orders from earth

For the coordinator module, there are two types of orders that are received from earth:

1. "SLAVE": Sent to the slaves to put them in a cooperative mode and ready to receive orders from the master. Before receiving this order, the robots have an independent behavior from the other robots.
2. "GO2_COP X Y H FT L D N": Sent to the master to go to a specific destination moving in formation with the other robots. The parameters of this order are the following. The name in parenthesis is the corresponding input variable in the written code.
    - GO2_COP: Identifier for the cooperative mode.
    - X (par1), Y (par2) H (par3): X,Y are the destination coordinates and H identifies the heading angle. All distances are given in meters and the angle is given in radians.
    - FT (parc1): Indicates the Formation Type that could be "H" for an Horizontal formation what means one robot at the side of the other; or "I" for an Indian line formation what means one robot after the other.
    - L (parc2): Indicates who leads the formation, from first to last in the Indian line and from right side to left side in the horizontal formation. "S" for slaves leading and "M" for master leading.
    - D (par4): Indicates the dilatation coefficient in the formation that indicates the separation between agents given in meters.
    - N (par5) : Indicates the number of slaves involved in the formation.

4.2. Orders from master

1. **"GET_POSITION":** Returns to the master an array in the form [x,y] containing the position of the slave that received the order.
2. **"GO_TO X Y":** Order to move to the destination coordinates X,Y.

4.3. Information regarding the success or fail of a task

1. **"SUCCESS":** Message returned to the master when the slave has succeeded in the reach of a destination.
2. **"ERROR":** Message returned to the master when the slave couldn't arrive to destination.

3. **“DONE”:** Message sent from the master to the salve when the coordinate task is done and taking the slave out from the cooperative mode.

4.4. Programming communications: Sockets

At the programming level, the communications will be performed using SOCKETS. A buffer called “prueba” will contain all the information to transmit. After the sockets, ports and buffer definitions, the sockets must be initialized.

To set a communication link a request have to be done. The robot that wants to transmit the information does the request using the function *connect* with the parameters relative to the robot that will receive the information. In the other side the robot that is meant to receive must accept the request using the function *accept*. Once the socket is set, the robots start sending and receiving information using the functions *send* and *recv*.

All the information needed to set the communications is acquired in the registration of the slave robots, as a part of the **COORDINATOR MODULE.**

To finish transmitting, all sockets must be closed using the following functions:

```
closesocket(ConnectSocket);
WSACleanup();
```

5. MOVE IN FORMATION: COORDINATOR MODULE FOR TBRA

After the protocol definition two programs called AUTONOMY_COORDINATE are going to be proposed, one for the Master and one for the slaves. The final scope of this module is to make robots move in formation.

5.1. AUTONOMY_COORDINATE (Master side)

As said before the coordination task developed is the movement in formation supported on the path planning module developed previously by the Thales Alenia Space robotics engineers. The mentioned task is performed sequentially because it depends on the movement of the master robot, this last, determined by the path planning module that works sequentially.

The path planning module already implemented in the TBRA has not been projected to work in cooperative tasks involving a group of robots; it is projected to work in an operative single robot as a part of a classic space mission. All algorithms, calculations and programs will be developed under this condition.

When moves, the master robot follows a trajectory traced by the path planning module. This trajectory is determined with base on the observations made by the robot using its different sensors. Depending on the height of the obstacles and terrain, the robot makes a map and plans a trajectory. This path planning is not executed in parallel with the movement and the robot has to stop periodically to make measures. After this, the robot could continue with its previous trajectory or could change it.

The following images are taken from the Navigation Map Visualizer that generates an image in which it can be seen: the robot's perception in each moment, the intermediate points and the path, all generated by the path planning module.

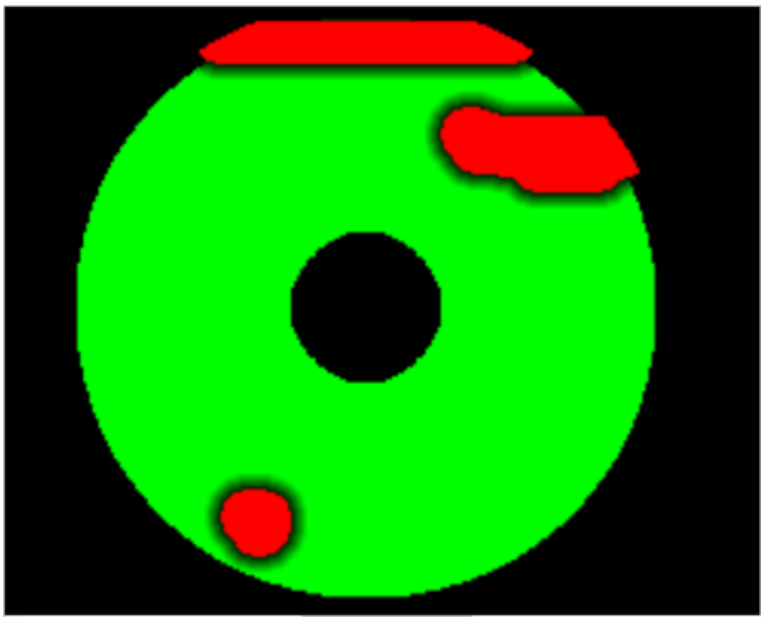


Fig. 13a. Robot's first perception.

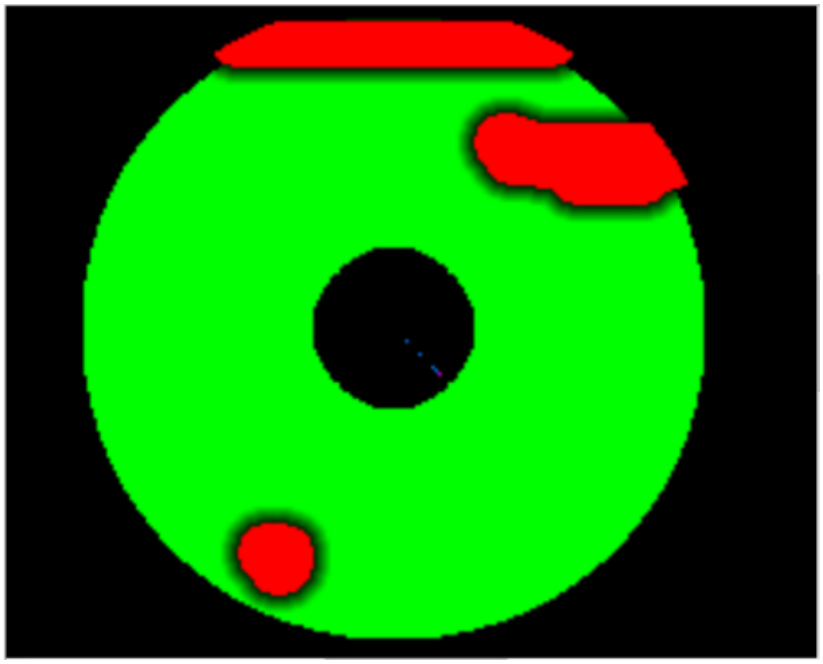


Fig. 13b. Intermediate points

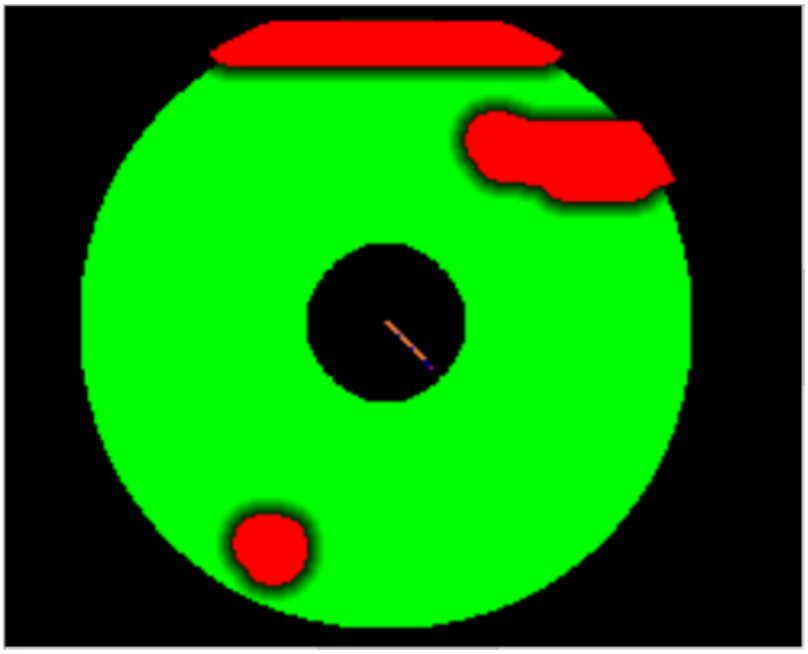


Fig. 13c. Robot trajectory

The fig. 13a shows the first perception made by the robot. Is circular with radius 5m, because the perception is done using images processing and the images are taken from a camera, above the robot, that turns 360° generating a navigation map of the robot nearest environment.

The black areas are unknown for the robot (Unsafe), the light green areas are known and safe and the red areas are obstacles to avoid. The camera is set in the way the robot's structure won't be a part of the seen environment generating, in the first perception, the black central circle in the navigation map.

The fig. 13b shows, in the central black circle, some blue points that correspond with the trajectory intermediate points (path). The calculated intermediate points are heading to a location out of the black central circle. The robot is trying to get out of an area considered unsafe.

In the fig. 13c, it can be seen the trajectory followed by the robot indicated with an orange line.

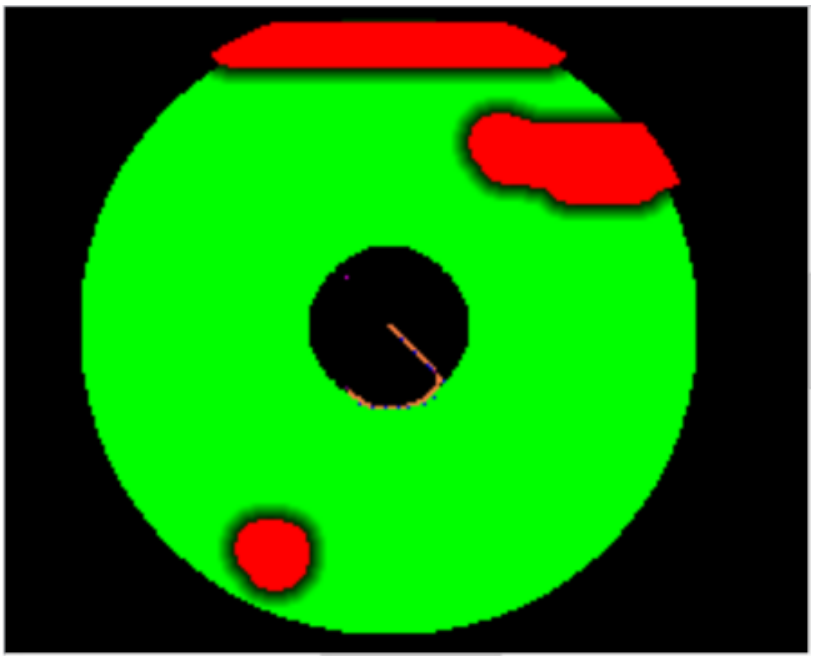


Fig. 14a. Trajectory avoiding unsafe area

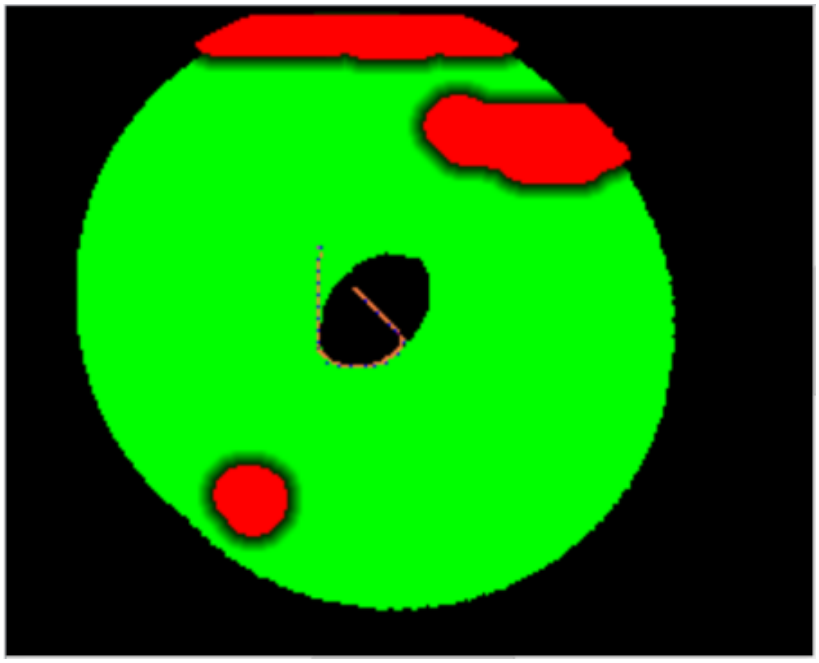


Fig. 14b. Second environment perception

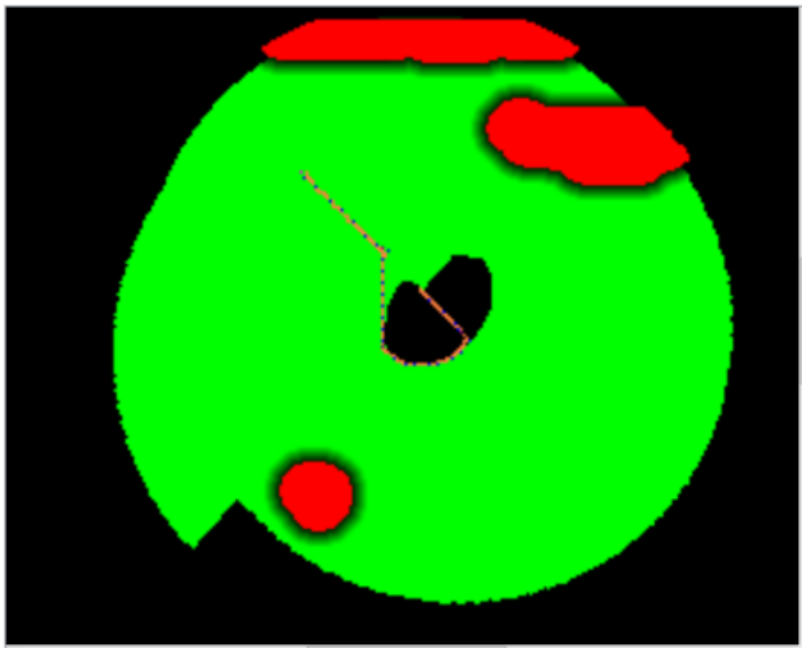


Fig. 14c.Third environment perception

Fig. 14d. Fourth environment perception

In fig. 14a it can be seen how the robot is heading to its final destination avoiding the unsafe area.

Figs. 14b, 14c and 14d shows the sequential environment perceptions performed by the path planning module and how the unknown areas get smaller as the navigation map gets bigger.

Given the circumstances of movement of the master robot, it can be seen that a movement in formation, constant in time, can't be performed but it can be approximated using the algorithm proposed in this thesis.

The stops performed to make measures, can be seen as intermediate destination coordinates for the master that can be used to calculate the intermediate destination coordinates for the slaves (Leader referenced position).

Each time the master knows its own intermediate destination coordinates, it will send to the slaves the order to go to the intermediate calculated coordinates that will correspond with the original formation coordinates but displaced and rotated. Calculated these coordinates, the master will start its trip and just after, it will send the move order to the slaves. Because the transmission of the orders doesn't generate significant delays, all robots will start its own trip almost at the same time.

In the movement from one point to another there are going to be intermediate points, in which the robots are going to be in formation during the journey simulating a constant formation movement. Depending on the environment in which the robots have to execute this task, in the trajectory between one intermediate point to another, the formation could not be too exact. This happens due the conditions of the path planning and the navigation modules in which the trajectory plan can't be performed in parallel with the movement. The path planning module is projected as an independent module and the coordinator module proposed in this thesis can only work under the path planning conditions.

5.1.1. General description of the algorithm

1. Registration of the slaves
2. Save all the slaves position
3. Navigation Map generation
4. Target tracking task
5. Calculate the initial formation coordinates
6. Send the move in formation orders (GO_TO X,Y)
7. Wait until all the slaves arrival
8. Positions verification
9. Navigation Map generation
10. Target tracking task
11. Calculate master destination coordinates
12. Start Master's movement
13. Return to number 5 until the master robot reaches its destination.

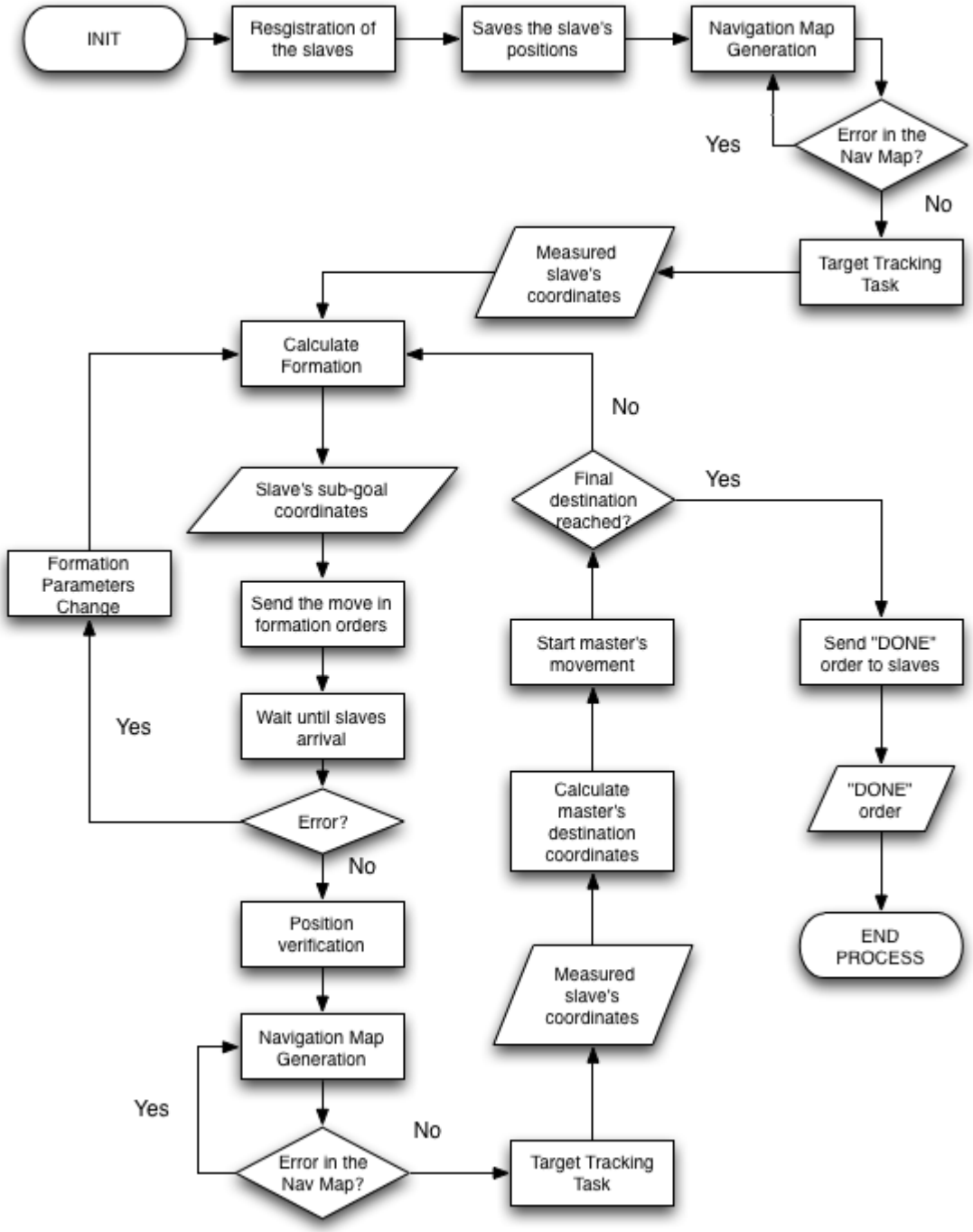


Fig. 15. Autonomy coordinate (master side) flow diagram

5.1.2. Specific description of the program

The AUTONOMY_COORDINATE program for the master can be divided in the following parts:

5.1.2.1 Registration and slave's position saving

The first step in the program running is the registration of all the slaves. The master robot enters in a "listening mode" and works like a network server, waiting to receive the IP addresses of all slaves and saving them in the "ClientListS" string of a structure called "robot".

Each time a connection request is accepted a thread is created and the data saving is done. In parallel the master continues running the main program, waiting for the other slaves registration until the last one is registered.

For the probes done in this thesis, because there are only two robots (Master and slave), no thread is created but the code is projected to work with more than two robots, where a thread could be useful. The thread is written but no used for now.

The ID of the robots (IP address) will be used posteriorly by the master to send the respective orders to the slaves.

As a second part of the registration, once the master has the IP address of all the slaves, ask to each robot its own position and save it in the xy array of the "robot" structure, coupling it with the corresponding ID.

The "robot" structure is defined as follows:

```
struct list{
   char ClientListS[15];
   float xy[2];
};
  struct list robot[MaxClients];
```

The variable MaxClients indicates a maximum number of slaves that can be known.

The complete registration program code can be seen in the attached file. The registration is done in the case RunningStep=0. The position obtaining is done in the case RunningStep=1 subcase step=0.

5.1.2.2 Navigation Map generation

To use the path planning module to obtain the intermediate points in the trajectory from the actual position of the robot to the final destination, a navigation map must be generated before each movement and path calculation.

For this task the following function is used: `am.GetNavigationMapData(navmap);`

The navigation map is saved in the variable *navmap*

5.1.2.3 Target Tracking task

The TARGET TRACKING MODULE developed in this thesis and explained in chapter 3, is used in the performance of the COORDINATOR MODULE as follows.

All robots use in the same way the Path planning module to move, perceiving the world using the cameras and sensors, and so, each robot generates a navigation map centered in itself. This means each robot has its own reference frame.

To move in formation, the information shared between robots regards coordinates and positions that need to be set using a unique reference frame.

To solve the problem, one possible solution would be to correct the position in which the slave thinks it is, overwriting it with the position obtained using the TARGET TRACKING MODULE.

This solution will be useful but implicates deep and complex changes in modules previously created. The modules proposed in this thesis are projected to work under the conditions of the modules already existent and don't require complex changes in other modules, consequently the subsequent solution is proposed.

Neglecting the heading angle, the transformation between the slave's reference frame and the Master's reference frame can be done applying a traslation.

Using the TARGET TRACKIN MODULE (TTM) the coordinates (*x, y, z*) of the slave robot can be obtained, based on the Master's reference frame. These values are going to be used later to apply the relative transformations to the coordinates shared between the robots.

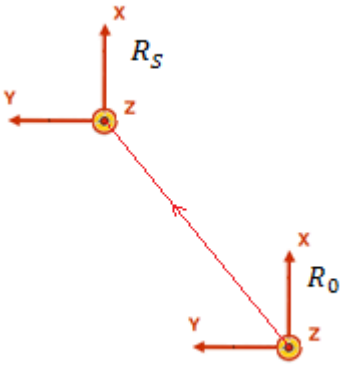


Fig. 16. Reference frames of the slave robot (s) and the master robot (0) using the simulation orientation

The transformation is done as follows:

$$\begin{bmatrix} x_s \\ y_s \\ z_s \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} + \begin{bmatrix} x_t \\ y_t \\ z_t \end{bmatrix}$$

Where x_t, y_t and z_t are the coordinates (x, y, z) obtained using the target tracking module.

Each time the master calculates the destination coordinates for the slave, the transformation is applied sending the correct coordinates to the slave.

To obtain the slave's coordinates from the Target tracking module, the next function is used :

```
am.GetTargetTrackingStatusData(statusTargetTracking);
```

After the images processing, this function saves the position of the target in the structure `statusTargetTracking`. To use these values the next assignations are done:

```
slavex=statusTargetTracking.estimation_tmat[9];
slavey=statusTargetTracking.estimation_tmat[10];
```

Finally with the *slavex* and *slavey* values, corresponding with the *x* and *y* values acquired using the TTM, the traslation vector is applied obtaining all coordinates respect to a unique reference frame, in this case the Master's reference frame.

The *z* axis coordinate is neglected in the previous development because for the formation is not needed.

In this thesis the *heading* angle is not estimated and all calculations are done with *heading angle* equal zero. In the case the heading is known, a roto-traslation matrix should be used.

In the attached code, this roto-traslation matrix is set to easily be used in future works.

5.1.2.4 Orders writing

In these steps all orders are prepared and written to be sent to the slave robots. A string called "prueba" is constructed with the order and the respective parameters.

To perform the movement in formation, before writing the orders, all values of the destination coordinates are calculated (case RunningStep=1 subcase step=1). The destination coordinates are calculated depending on the formation type and based on the number of agents and on the destination coordinates of the master robot. The first thing to do is calculate the formation, neglecting the angle and then multiply the coordinates system by a rotation matrix.

The order to be sent is written in the form GO_TO X,Y. X and Y are the parameters to be calculated.

5.1.2.4.1 Horizontal formation

```
if(strncmp(parc1,"H",(int)strlen(parc1)) == 0){
      if(strncmp(parc2,"M",1) == 0){
                y=Localy;
                x=x+par4;
          }
    else if(strncmp(parc2,"S",(int)strlen(parc2)) == 0){
                y=Localy;
                x=x-par4;
    }
  }
```

The "if" conditions are set to operate in the respective cases. In this one, "H" identifies the formation type the "M" or "S" indicates who leads the group.

The code fragment showed gives the coordinates for all the slave robots, adding (for "M") or subtracting (for "S") the dilatation coefficient (par4), to the previous robot *x* coordinate. For an horizontal formation all robots must have the same *y* coordinate value.

For the first slave the *x* coordinate of the previous robot will be the *x* destination coordinate of the master robot. This is done assigning before the "for" cycle the master destination coordinate values ("Localx" and "localy") to the variables *x* and *y*.

The formation must be headed in the direction of the movement so after calculate the coordinates *x* and *y* the angle ɵ of movement can be calculated as follows:

$$\theta = \tan^{-1}(finalx - x/finaly - y) - \pi/2$$

$\pi/2$ must be subtracted because the formation must be perpendicular to the movement direction.

In the code we used the C++ function artan2 as follows:

```
angle=atan2((par2-Localy),(par1-Localx))-(pi/2)
```

The pi value was previously set to pi=3.14159265. Calculated the formation coordinates and the angle is just remaining the rotation that can be done applying the rotation matrix around *z* axis.

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The obtained equations are shown in the next piece of code:

```
rx=((x-Localx)*cos(angle)-(y-Localy)*sin(angle))+Localx;
ry=((y-Localy)*cos(angle)+(x-Localx)*sin(angle))+Localy;
```

Because the reference system is centered in the initial position of the master and the slave's coordinates are calculated based on the master's coordinates, the roto-traslation matrix must be applied to the actual coordinates of the master, so we first transfer the coordinate reference system to the master actual position, then we do the rotation and finally we move the coordinates to the original reference system. *rx* and *ry* are the final coordinates that are going to be sent.

The final coordinates are calculated and sent in a "for" cycle to each robot in the GO_TO order obtaining a movement as shown in fig. 17.

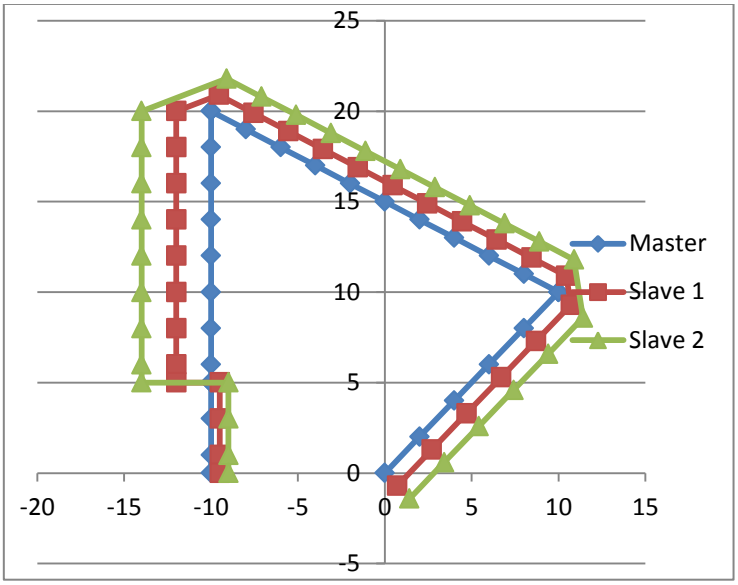


Fig. 17. H Movement graphic result

The Figure 17 was obtained saving the intermediate coordinates calculated for each robot. The figure shows the movement in Horizontal formation of 3 robots, with the master first and separated 1 m. The first destination coordinates were [10,10], the second [-10,20], the third [-10,5] and the last [-10,0]. The rhombus, squares and triangles, represent the intermediate points between two consecutive destinations, of the Master, slave 1 and slave 2 respectively.

It can be seen how in the first moment ([0,0]) the formation is done oriented to the destination coordinate [10,10] and each time the destination point changes, the formation is headed to the next destination point.

In the trajectory from [-10,20] to [-10,5], the dilatation coefficient was changed to 2m and in the trajectory from [-10,5] to [-10,0], the leader robot was changed to slave and the dilatation coefficient was changed to 0.5m to show different cases of the same formation.

The behavior in the intermediate trajectories depends only in the path planning module, and depending on the obstacles and characteristics of the terrain, the formation could not be too

exact. However, in all intermediate points the formation is done, approaching the global movement to a movement in formation constant in time.

5.1.2.4.2 Indian line

```
else if(strncmp(parc1,"I",(int)strlen(parc1)) == 0){
   if(strncmp(parc2,"M",(int)strlen(parc2)) == 0){
         y=y-par4;
         x=Localx;
   }
   else if(strncmp(parc2,"S",(int)strlen(parc2)) == 0){
         y=y+par4;
         x=Localx;
   }
}
```

The "if" conditions are set to operate in the respective cases. In this one, "I" identifies the formation type the "M" or "S" indicates who leads the group.

The code fragment showed gives the coordinates for all the slave robots Adding (for "M") or subtracting (for "S") the dilatation coefficient (par4), to the previous robot *y* coordinate. For an Indian line formation all robots must have the same *x* coordinate value.

For the first slave the *y* coordinate of the previous robot will be the *y* destination coordinate of the master robot. This is done assigning before the "for" cycle the master destination coordinate values ("Localx" and "localy") to the variables *x* and *y*.

After the formation, the coordinates *x* and *y* are transformed into *rx* and *ry* respectively, applying the relative roto-traslation matrix under the conditions explained in the previous case.

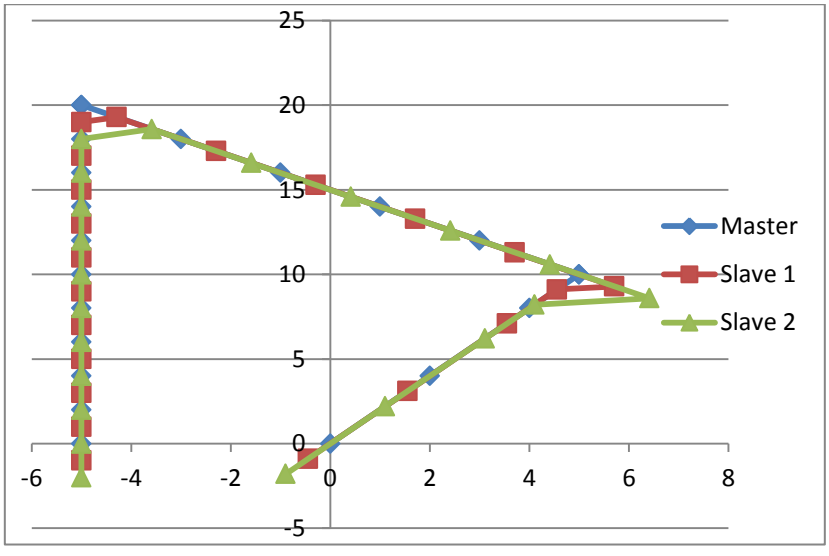


Fig. 18. I Movement graphic result

The figure 18 shows the movement of 3 robots in Indian line formation.

As well as in the previous formation, in the first moment ([0,0]) the formation is done and then the movement begins. In this test the first destination coordinate was [5,10] with the master heading the line. It can be seen that reached this point, the formation is done once again, oriented to the next destination point. In this moment the trajectories of the slave robots diverge from the master trajectory, due the redo of the formation, when the destination coordinates change.

Except in the case of a change of destination point, the trajectories of all robots overlap as should be due the formation type.

In the last trajectory form [-5,10] to [-5,0] the order changed and a slave starts heading the formation to show another case of the same formation. Even if a slave leads the group, the destination coordinates are based on the position of the master robot.

The conditions of the movement between points are the same described in the precedent formation.

5.1.2.5 Sending

After preparing all the information to send, the orders are saved in a string called “prueba” and using sockets, as explained in the communications chapter, the algorithm sends the orders one by one, to each IP address saved in the “robot” structure at the moment of the registration.

5.1.2.6 Moving

To perform the movement in formation as exact as possible, the master robot starts moving and just after, it sends all the slave’s orders. Because the transmission doesn’t generate significant delays, all robots start moving almost at the same time, holding the formation while the moving conditions (obstacles and terrain) for all robots remain invariable.

To perform the movement, the proposed coordinator module makes use of the navigator module and the path planning module, giving only the final coordinates.

The destination coordinates of all robots are calculated based on the destination coordinates of the master (Leader Referenced Position). To calculate these last coordinates, the following considerations were taken.

To make robots move in formation, the first thing to do is to determine some intermediate points in the master robot’s trajectory. After, using these intermediate points, we can calculate the sub-goals for the slave robots.

Two approaches are proposed to determine these intermediate points.

5.1.2.5.1 First approach: straight line trajectory aiming to the final destination coordinates

In this approach the Navigator module and the path planning module, are only used for moving from one point to another but the modules are not used for calculating the intermediate points.

To update its own position, the master uses the Localization Data Fusion Module as shown in the next piece of code (case: RunningStep=1; subcase step=1):

```
      am.GetLocalisationDataFusionStatusData(statusLocalisation
DF);
   for(int i=0; i<12; i++){
         tmat.trmat[i] = statusLocalisationDF.currPos_tmat[i];
   }
   clib::Tmatrix2Cpose(tmat, &robotPose);
   Localx=robotPose.pos.x;
   Localy=robotPose.pos.y;
```

To update the slaves position data, after each movement the order "GET_POSITION" is sent to each robot and the "robot" structure is overwrote with the new information.

To move from one point to another, the coordinator module proposes the shortest trajectory that is a straight line divided in intermediate points, in which the formation is going to be performed.
The intermediate points are separated for a fixed distance called *distfy.*

To determine the intermediate destination points, the distances *addx* and *addy* are calculated based on the fixed distance *distfy;* and posteriorly added to the actual coordinates of the master *Localx* and *Localy*.

To calculate the distances *addx* and *addy* the next equations system is proposed.

$$1.\, m = \frac{(y_{final} - y_{initial})}{(x_{final} - x_{initial})}$$

$$2.\, y - y_0 = m(x - x_0)$$

$$3.\, x^2 + y^2 = h^2$$

Where for the equation 1:

- *m* is the slope of the straight proposed trajectory.
- x_{final} =*par1* is the final destination *x* coordinate.
- y_{final} =*par2* is the final destination *y* coordinate.
- $x_{initial}$=*localx* is the actual *x* coordinate of the master robot.
- $y_{initial}$=*localy* is the actual *y* coordinate of the master robot.

For the equations 2 and 3:

- *m* is the slope calculated in the equation number 1.
- *x=addx* is the desired increment to *Localx*
- *y=addy* is the desired increment to *Localy*
- x_0=*Localx* and y_0=*Localy* are two points the line passes through.
- *h=distfy* is the fixed separation distance between to intermediate destination points.

Using the equations 2 and 3, the values of *addx* and *addy* can be calculated. The result is written in C++ language and used as shown in the next code fragment. (case RunningStep=1; sub case step=5)

```
addx=(sqrt(pow(distfy,2)*(pow(m,2)+1) - pow(m,2)*pow(Localx,2)
+ 2*m*Localx*Localy - pow(Localy,2)) - m*(m*Localx-Localy)
)/(m*m+1);

addy=(sqrt(pow(distfy,2)*(pow(m,2)+1) - pow(m,2)*pow(Localx,2)
+ 2*m*Localx*Localy - pow(Localy,2))*m + (m*Localx-Localy)
)/(m*m+1);


if((par1-Localx)>addx){
Localx=Localx+addx;
}
else if((par1-Localx)<-addx){
Localx=Localx - addx;
}

else Localx=par1;

if((par2-Localy)>addy){
Localy=Localy+addy;
}
else if((par2-Localy)<-addy){
Localy=Localy-addy;
}
else Localy=par2;
```

The *if* conditions are set to determine the sign of the values *addx* and *addy* in response to the direction of the movement.

In the case the remaining distance |*par1-Localx*|<*addx* the destination coordinate *Localx* takes the value of the final destination coordinate *par1* as the las step to complete the movement in the *x* axis.

In the same way when the distance |*par2-Localy*|<*addy* the destination coordinate *Localy* takes the value of the final destination coordinate *par2* as the last step to complete the movement in the *y* axis.

5.1.2.5.2 Final approach: Based on the path planning module

Making work together the Coordinate module and the path planning module we could obtain a more realistic formation movement. In this approach the intermediate points are those proposed by the path planning module.

As explained before, the path planning module makes a visual perception of the environment and calculates a trajectory divided in sub goals that are going to be used for calculating the sub goals of the slave robot.

In short, each time the master robot executes the following algorithm:

- Makes a visual perception
- Calculates a trajectory
- Calculates the master's sub goal
- Calculates the slave's sub goal
- Start moving
- Sends the move order to the slave

Once the master arrives to a sub goal, it waits until all slave's arrival and then repeats the algorithm until the final destination is reached, performing a movement in formation that will be as exact as the environment will permit.

To calculate the slave's destination coordinates, the first thing to do is to *call* the path planning module to obtain the master's sub goal. To do this the following functions are used:

- Create a navigation map:

```
sprintf(cmd2Navigator.command, "CREATE_NAVMAP");
    am.SendCommand_Navigator(cmd2Navigator);
```

- Use the navigation map, set the final destination coordinates (*par1*,*par2*,*heading*) and set the current position coordinates (*Localx* ,*localy*):

```
fds.setNavMap(&navmap);
fds.setGoal(par1,par2,0);
fds.setStart(Localx,Localy);
```

- Error rectification:

```
retval = fds.getPath(path, 2, 0.2);
```

- When no error is found, the last valid values are saved as sub goals:

```
subx=path.viaPoint[wp-1].X;
suby=path.viaPoint[wp-1].Y;
```

Once the master's sub goal is calculated, the slaves sub goals can easily be calculated using the same calculations described in ORDERS WRITING.

After calculate the master's destination sub goal coordinates, it uses the navigator module, coping the destination coordinates *Subx* and *Suby* after the order "NAVTOTARGET" in a string and then sending it as a command to the main system as follow.

```
cmd2Navigator.counter++;
sprintf(cmd2Navigator.command, "NAVTOTARGET %f %f 0.0 0.2",
Localx, Localy); //Heading 0.0 rad, Boundary 0.2 m
am.SendCommand_Navigator(cmd2Navigator);
```

Just after the master movement begins, the slave's coordinates are calculated.

In this case to calculate the formation angle, the parameters used are the actual position and the next sub goal (in the first approach the angle was determined using the actual position and the final destination).

This means that the formation heading angle can change during the trajectory what gives a more realistic move in formation as shown in fig. 18f.

In the case the master robot changes its trajectory considerably due an obstacle; also the slave robot will change its trajectory keeping the formation.

The calculation of the formation angle for this approach is done as follows:

```
angle=atan2((suby-Localy),(subx-Localx))-(pi/2)
```

The rotation matrix is the same used in the first approach with the new angle calculations.

After calculate the slaves coordinates the orders are sent.

Even when the master starts moving before the command sending, the movement of all robots starts almost at the same time because the calculation and transmission delay are too short.

A simulation of the Indian Line formation (I) where the master is leading the formation, with an obstacle in the trajectory.

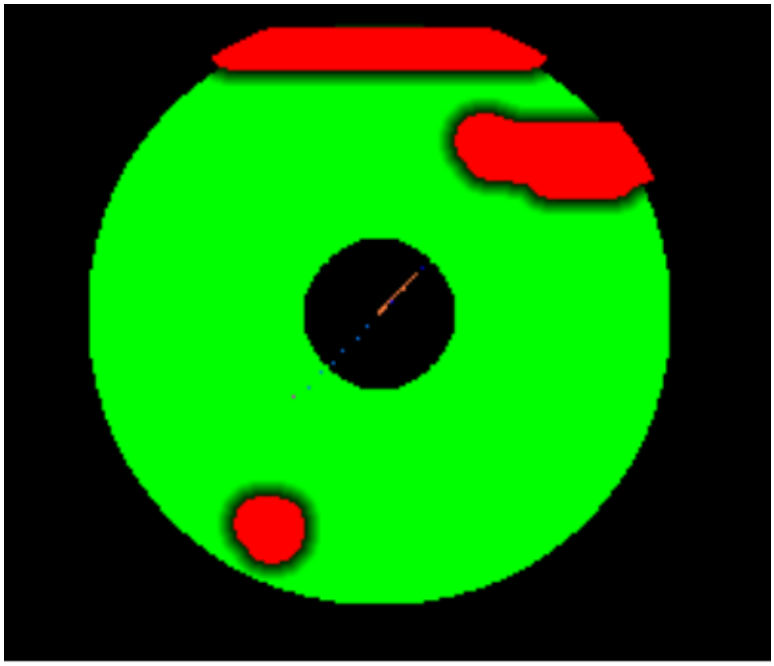


Fig. 19a. First environment perception and Indian Line initial formation (slave)

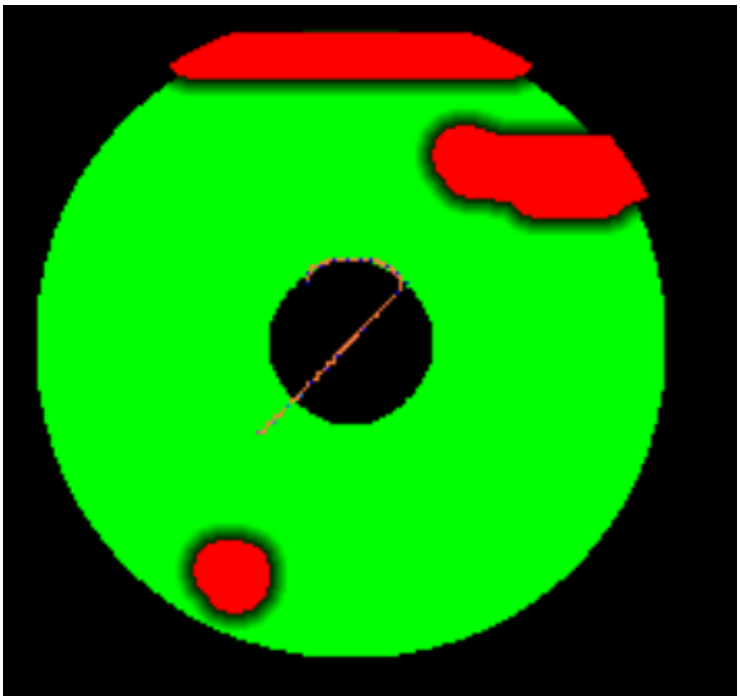


Fig. 19b. Master moves forward-Slave avoiding unsafe area

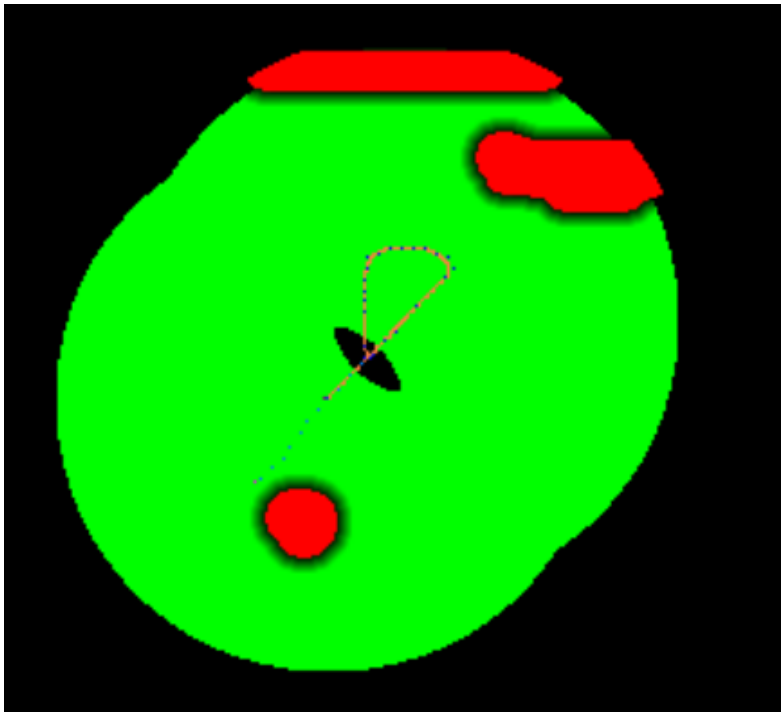


Fig. 19c. Master waits until the slave arrives in formation

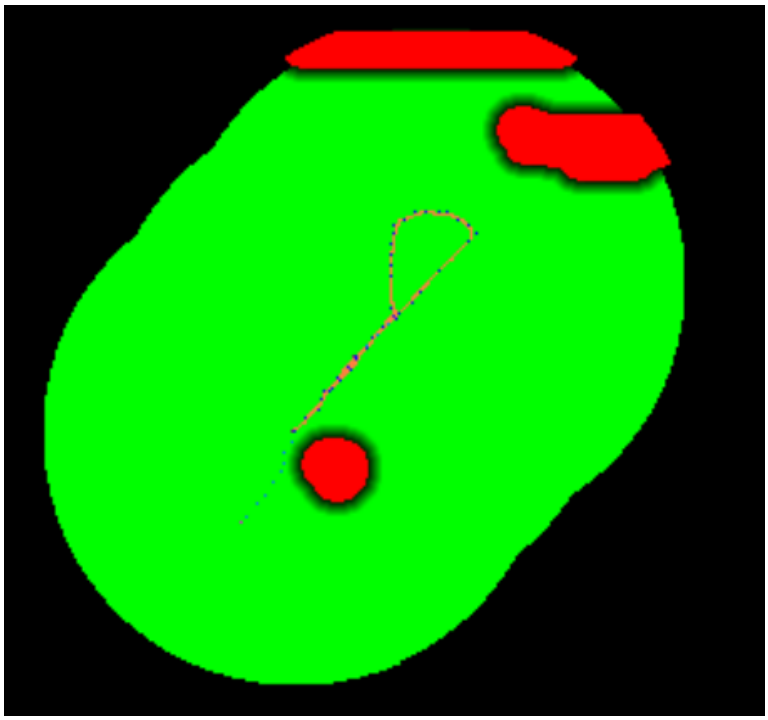


Fig. 19d. Robots moving in formation avoiding the obstacle

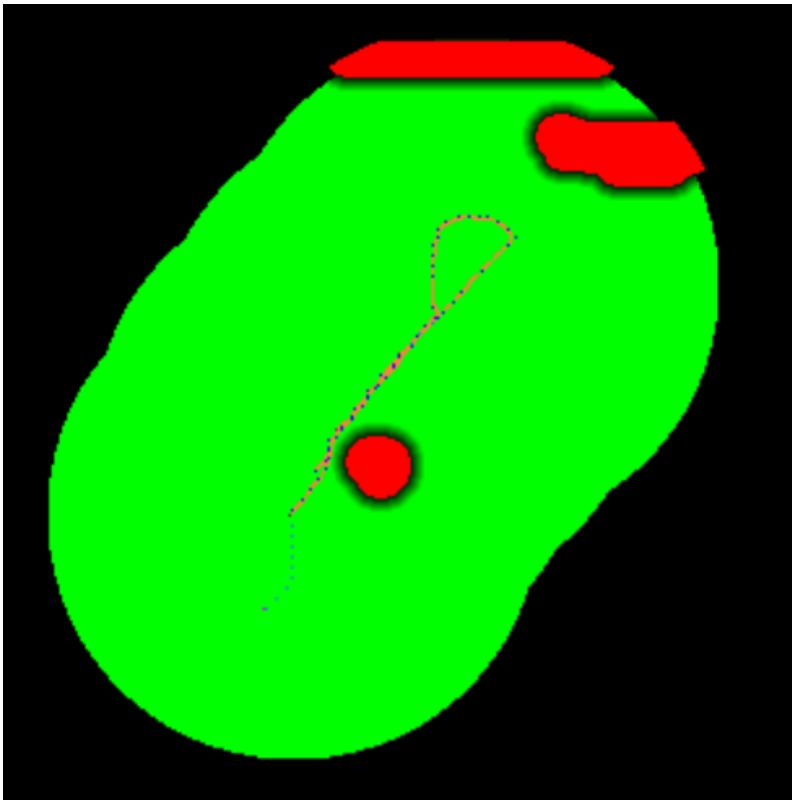


Fig. 19e. Robots moving in formation avoiding the obstacle

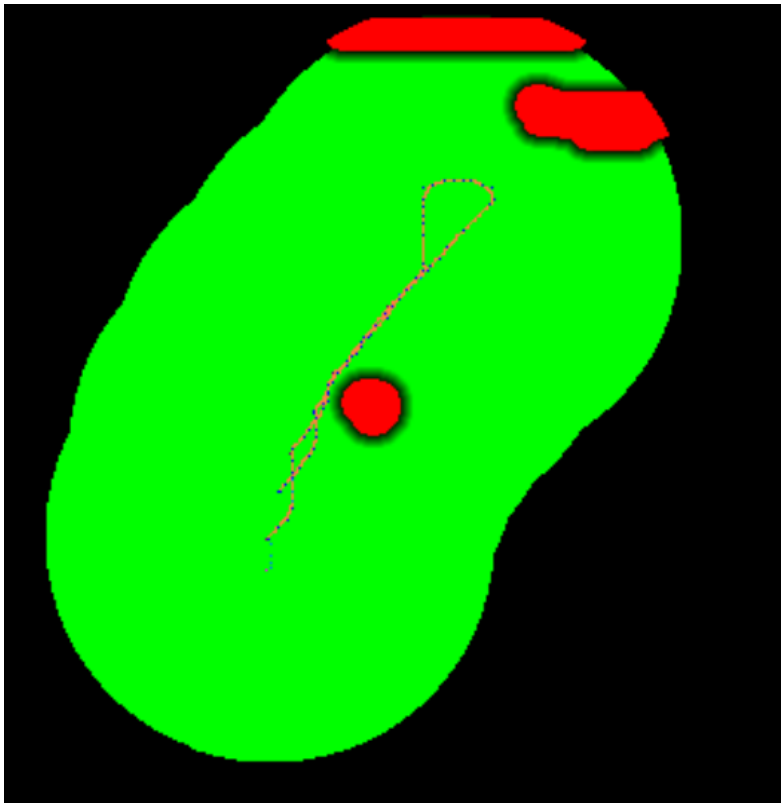


Fig. 19f. Change in the formation angle due the avoiding maneuver

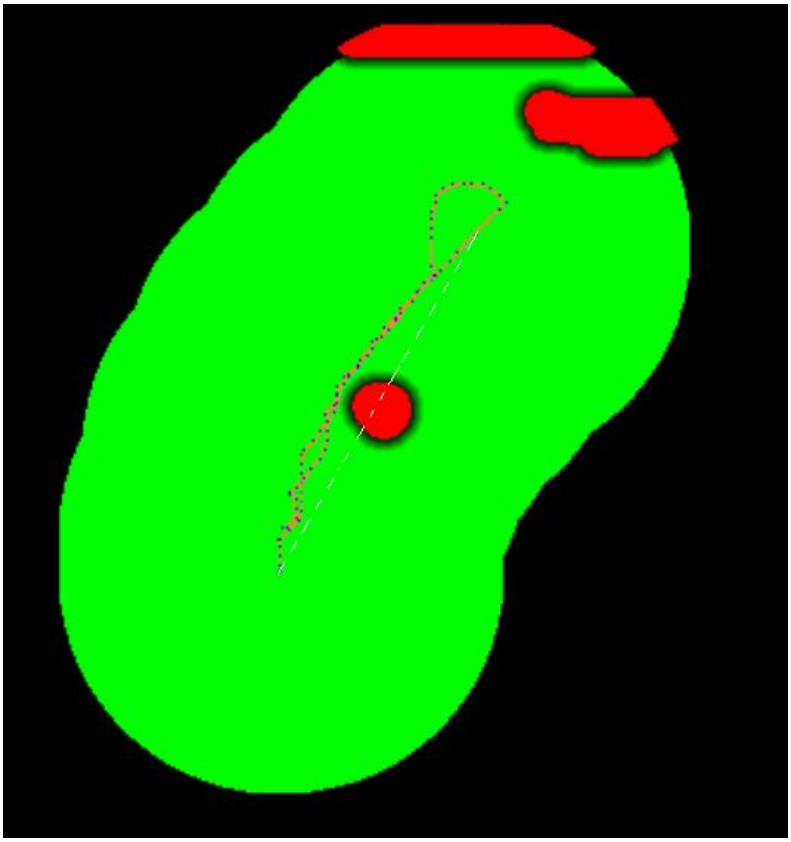


Fig. 19g. Robots arrival in formation - Complete trajectory avoiding obstacle keeping the formation

The previous simulation shows the execution of the order **GO2_COP -7 4 0 I M 1 1** what means the robot should move performing an Indian line formation, from the initial coordinates (0, 0) to the final coordinates (-7, 4) with a heading angle zero. The master will lead the formation and the slave must keep a separation of 1m.

The *x* axis aims to the top of the page, the *y* axis aims to the left side of the page and respecting the right hand rule, the *z* axis aims out of the page as seen.

As can be seen there's an obstacle in the trajectory so the path planning should calculate a trajectory avoiding the obstacle.

In fig. 19a it can be seen the slave moving to the first formation.

Figs. 19b and 19c show the first movement in formation. The robots can't keep the formation at this point, because the slave robot is trying to avoid an unsafe area. The master robot moves and then waits until the slave robot arrives in formation.

Figs. 19d, 19e and 19f show the robots moving in formation avoiding the obstacle. In fig 19f the formation angle changes considerably due the followed trajectory. The robots keep in formation heading to the next sub goal.

Fig 19g shows the robots arrival to destination what means the end of the coordinator task. In this figure it can be seen that even when the trajectory wasn't straight, the robots formation was kept between the tolerance limits.

A simulation of the horizontal formation (H) is shown in the succeeding images where the robots execute three consecutive orders with different goal coordinates keeping the formation.

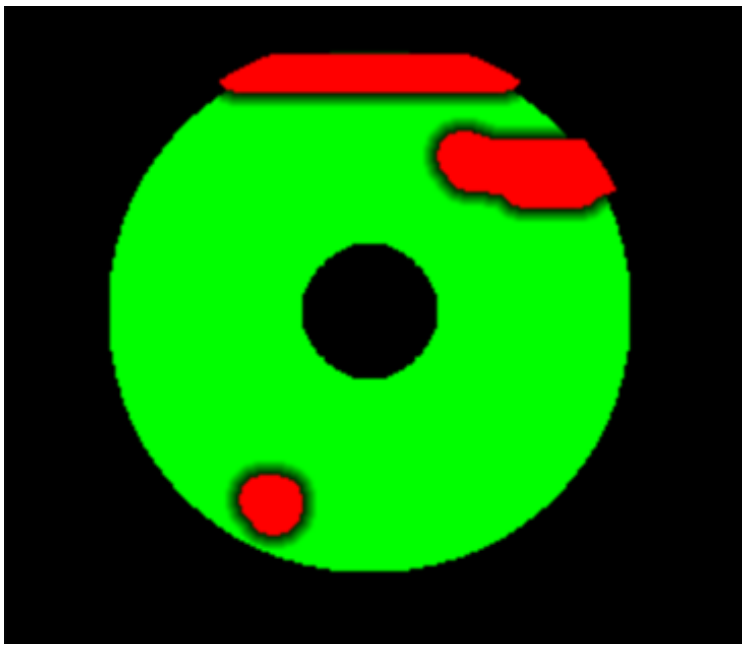


Fig. 20a. First environment perception

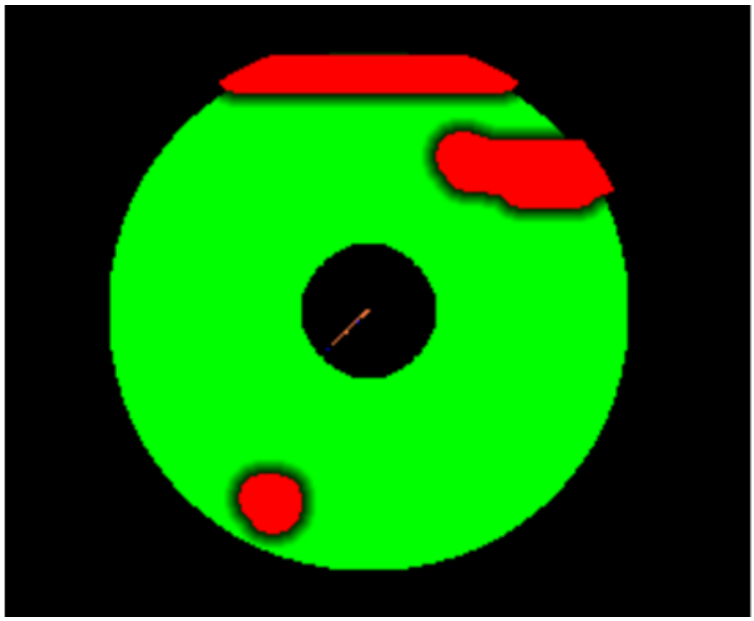


Fig. 20b. First formation (slave)

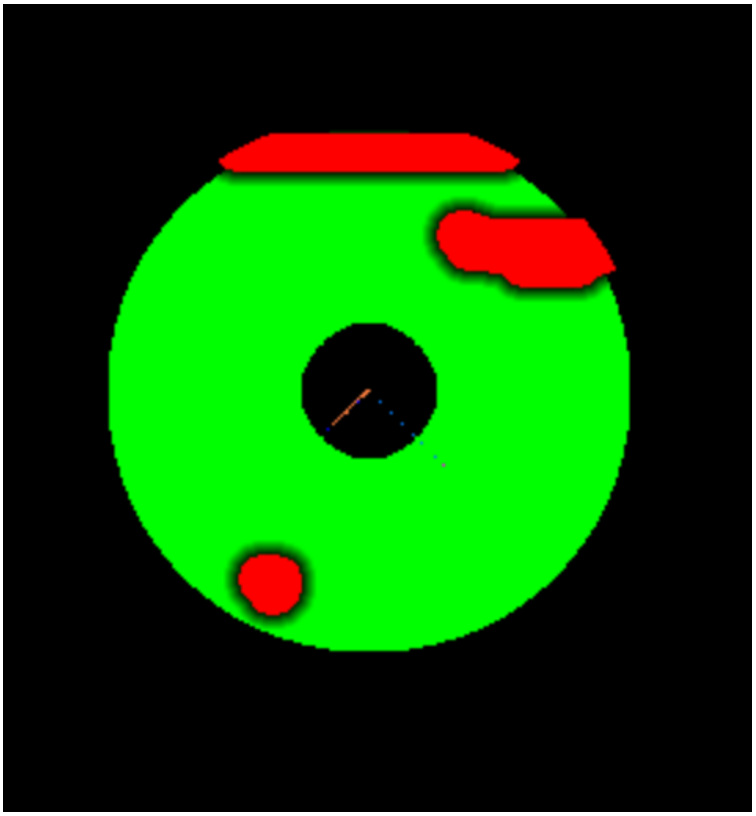


Fig. 20c. Master's Path planning

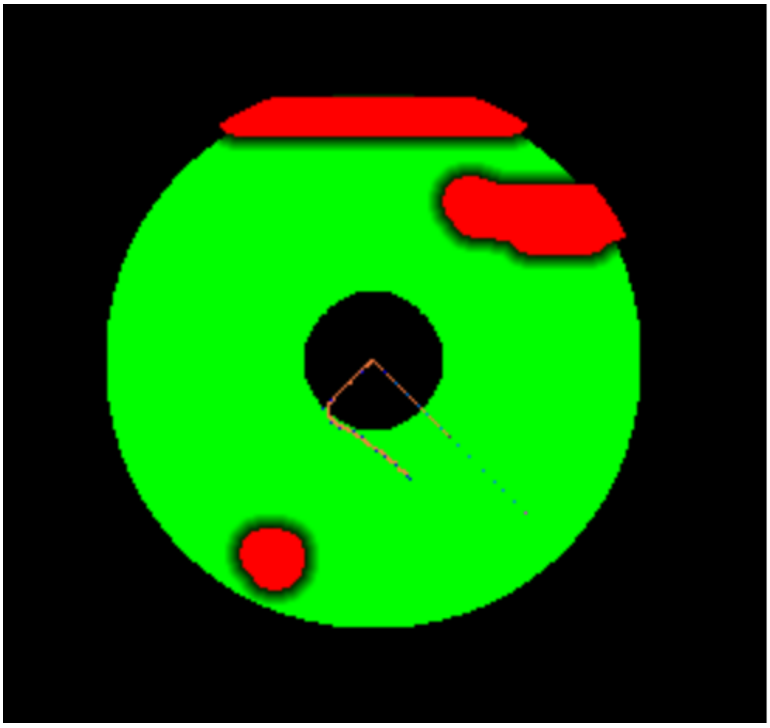


Fig. 20d. Movement and Master's path planning

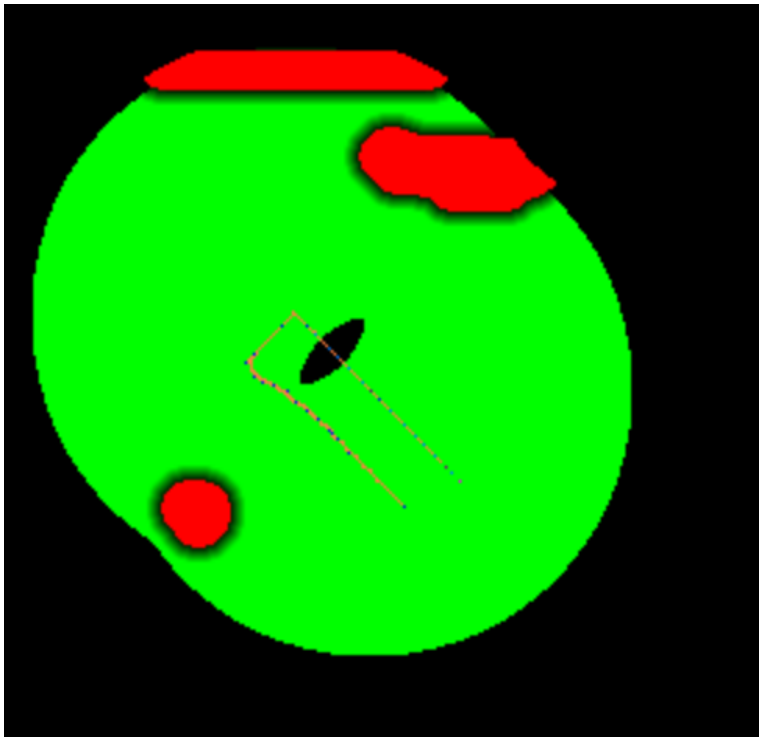


Fig. 20e. New environment perception, movement and Master's path planning

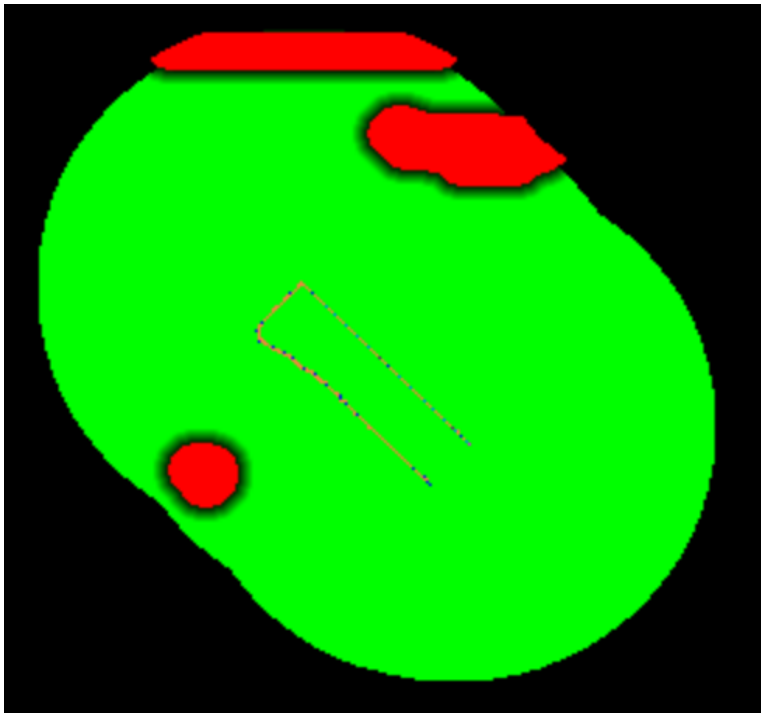


Fig. 20f. Final destination reached

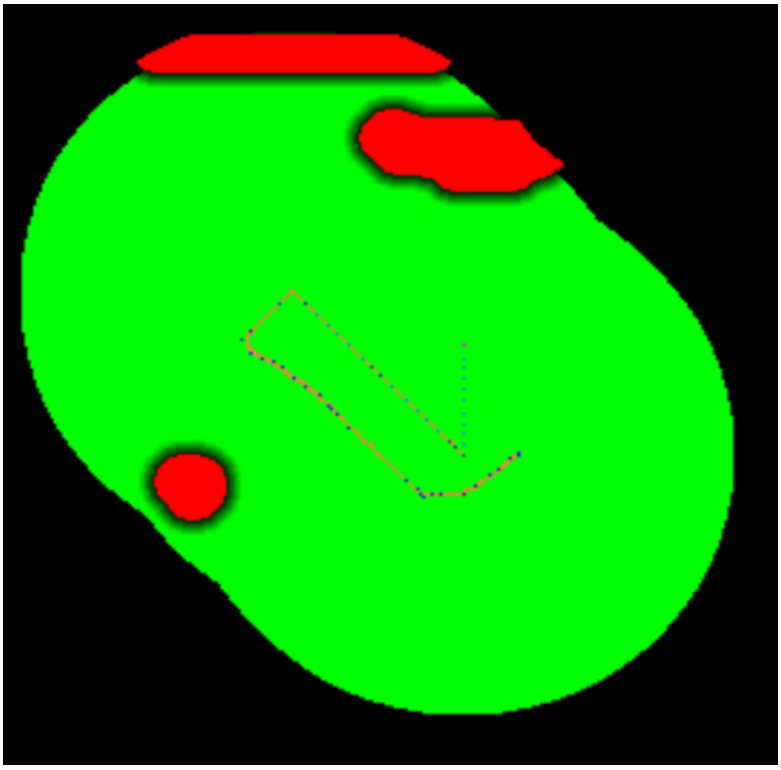


Fig. 20g. Formation heading to a new goal (slave) and Master's path planning

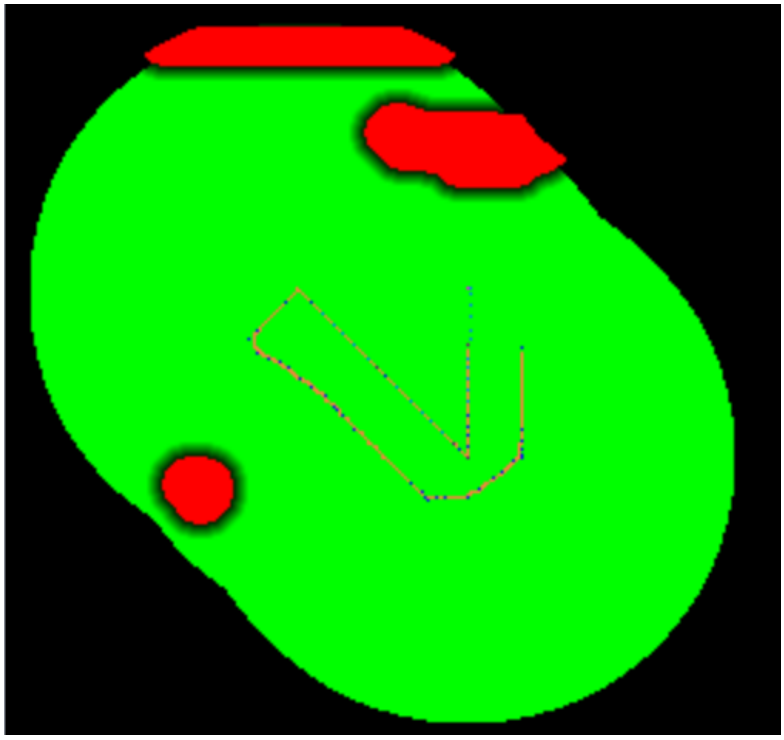


Fig. 20h. Movement in formation and Master's path planning

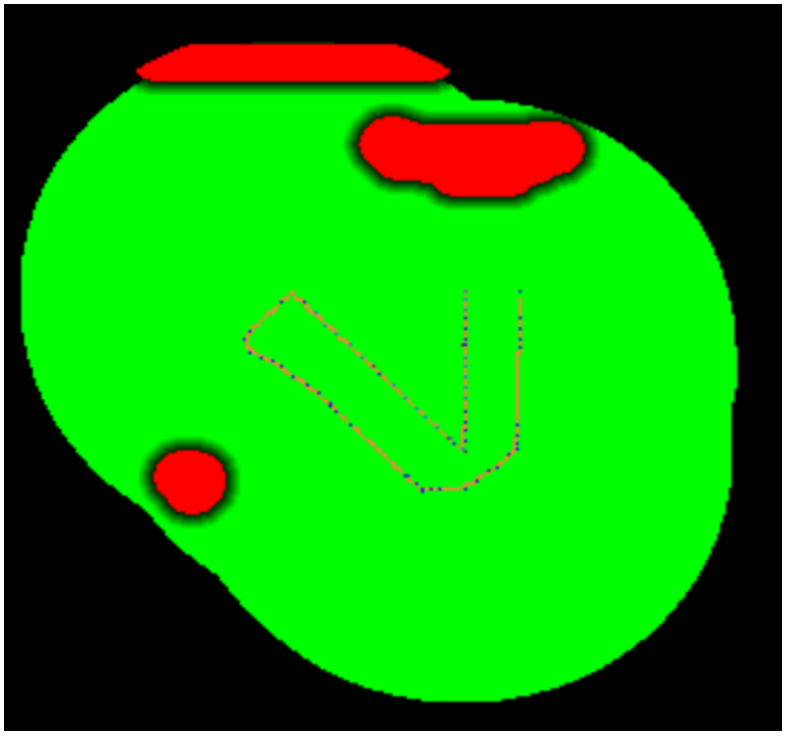


Fig. 20i. Destination coordinates reached

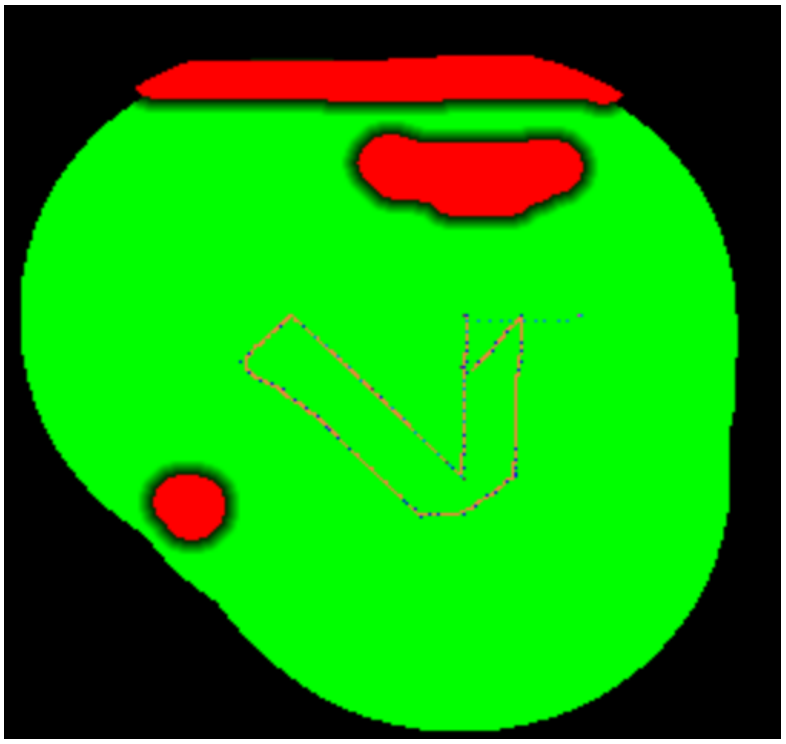


Fig. 20j. Formation heading to a new goal (slave) and Master's path planning

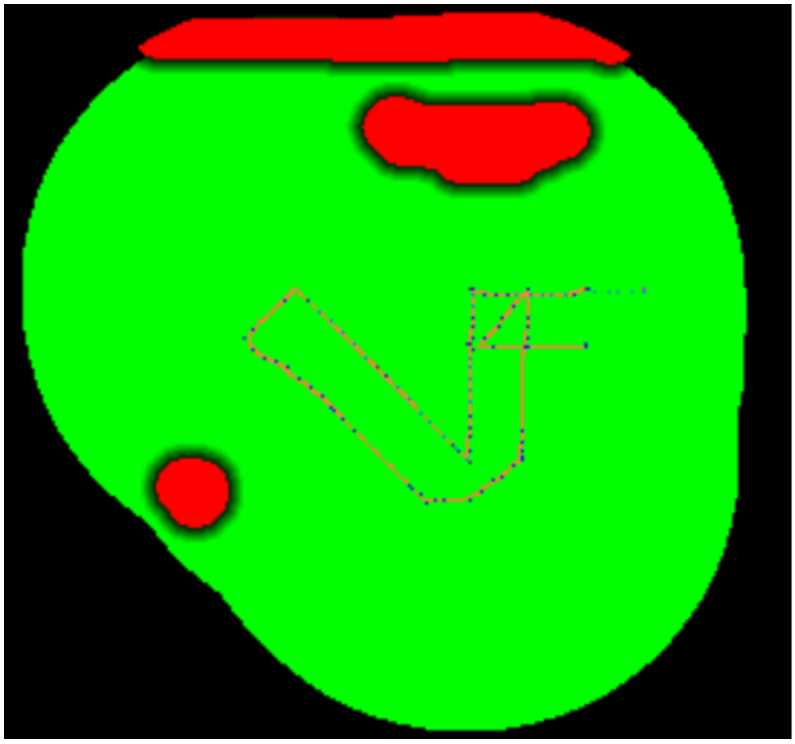


Fig. 20k. Movement in formation and Master's path planning

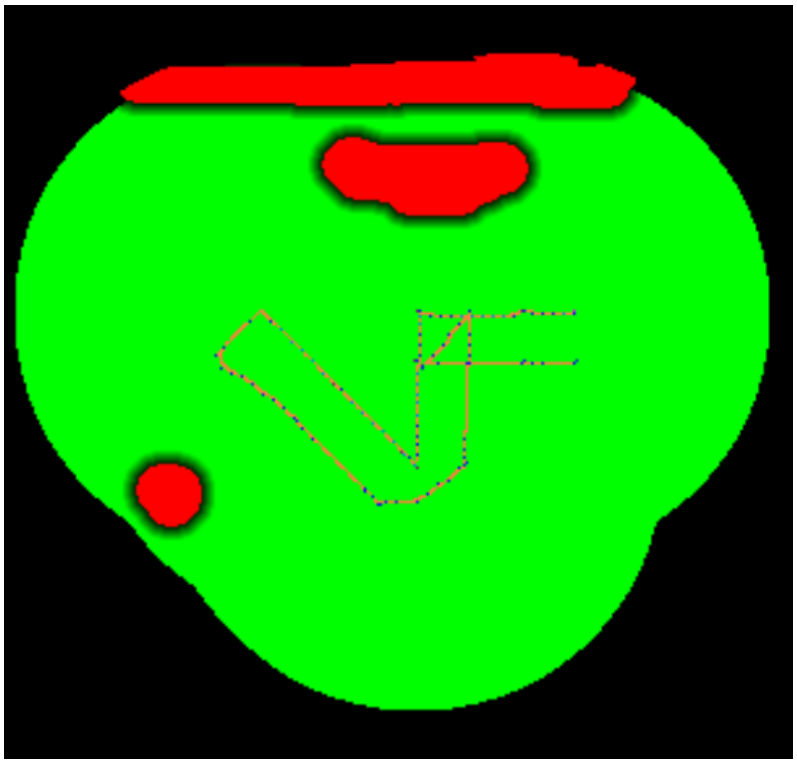


Fig. 20l. Destination reached

The previous images show the movement in formation from the origin to tree final destinations (-3,-3), (0,-3) and (0,-6).

Figures from20a to 20f show the executing of the order ***GO2_COP -3 -3 0 H M 2 1***

In the figure 20a, the first perception is show. Figure 20b shows the slave performing the first formation aiming to the final destination coordinates (-3,-3). In figure 20c it can be seen the formation and the first sub goal (determined by the path planning module) that is used for calculating the slave's sub goal coordinates.

Figures 20d, 20e and 20f show the movement in formation performed following the path planning sub goals.

Figures from 20g to 20i show the executing of the order ***GO2_COP 0 -3 0 H M 2 1***

Figure 290 show the formation performed by the slave, aiming to the new final destination coordinates (0,-3). Figures 20h and 20i show the moving in formation.

Figures from 20j to 20l show the executing of the order ***GO2_COP 0 -6 0 H M 2 1***

Figure 20j shows the formation performed by the slave aiming to the new final destination (0,-6). Figures 20k and 20l show the moving in formation of the robots, following the path planning sub goals.

5.1.2.6 Error response

Each time a robot moves the following errors may occur:

- A path can't be found
- The goal is inside an obstacle
- A generic error occurred

After each movement, the master robot waits for an answer from the slaves.

In the case there was no error, all slave robots send the message "SUCCESS" indicating they arrived to the desired position.

In the case one or more of the slave robots cannot reach destination due an error, the master receives a string containing the word "ERROR" and reacts depending on the current formation type.

If at the moment of the slave robots movement error the type of formation wasn't Indian line with the master robot heading, the formation type is changed to Indian line with the master first and 1m of separation; and the movement continues.

If at the moment of the slave robots movement error the type of formation was Indian line with the master heading the formation, the master reacts to the error sending to the slaves, the order to go to the final destination breaking the formation.

The motivation for this behavior is that in the case the formation type is not the Indian line, the trajectories of all robots are different and it is possible that one or more of the slave robots could not reach their goals but the master still can, so changing the formation to Indian line

with the master first and separation 1m, all trajectories would be the same and there is no reason why the slaves could not reach destination if the master can.

In the case the formation type is Indian line with the master first, and an error in the movement occurs, the goal is not reachable in formation but is still possible that the robots reach the goal if the formation is broken because in this case each robot will plan its own trajectory.

In the code the *error* variable made evident the situation described before and is used as a flag to react to the situation. The complete code can be seen in the attached files (case: RunningStep=1; subcase: step=2)

In the case the final destination is reached by all robots (case: RunningStep=1; subcase: step=6) or in the case the goal is impossible to reach by the master robot due an error in the navigator module (RunningStep=2), the master sends to the slaves a string containing the word "DONE" to notify them, the task is over.

The following simulation show the error response

To make the error behavior visible the first thing to do is to move the robots to a safe position like (-3,0) in horizontal formation.

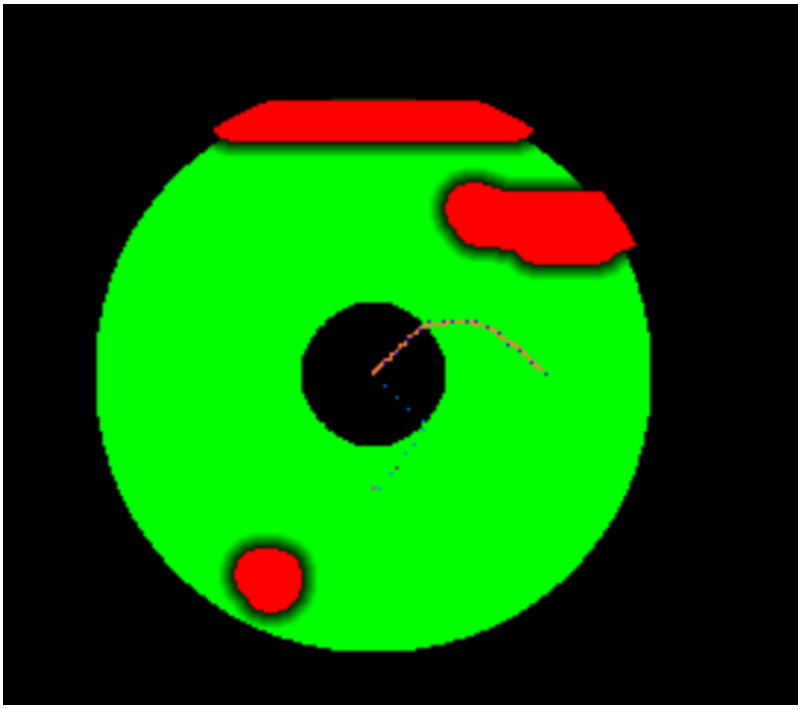


Fig. 21a. First formation aiming to final destination

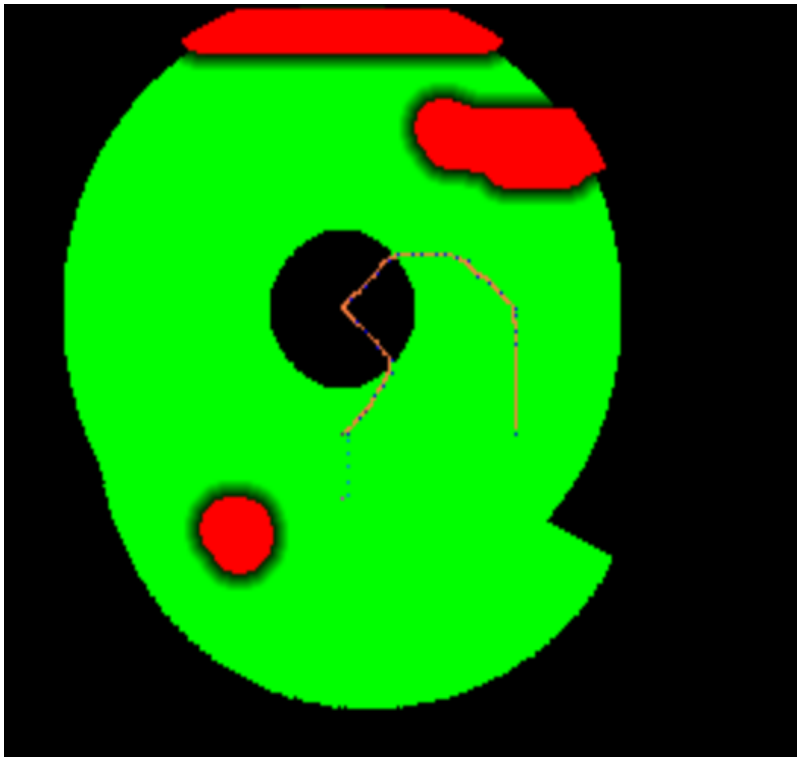


Fig. 21b. Moving in formation and Master's path planning

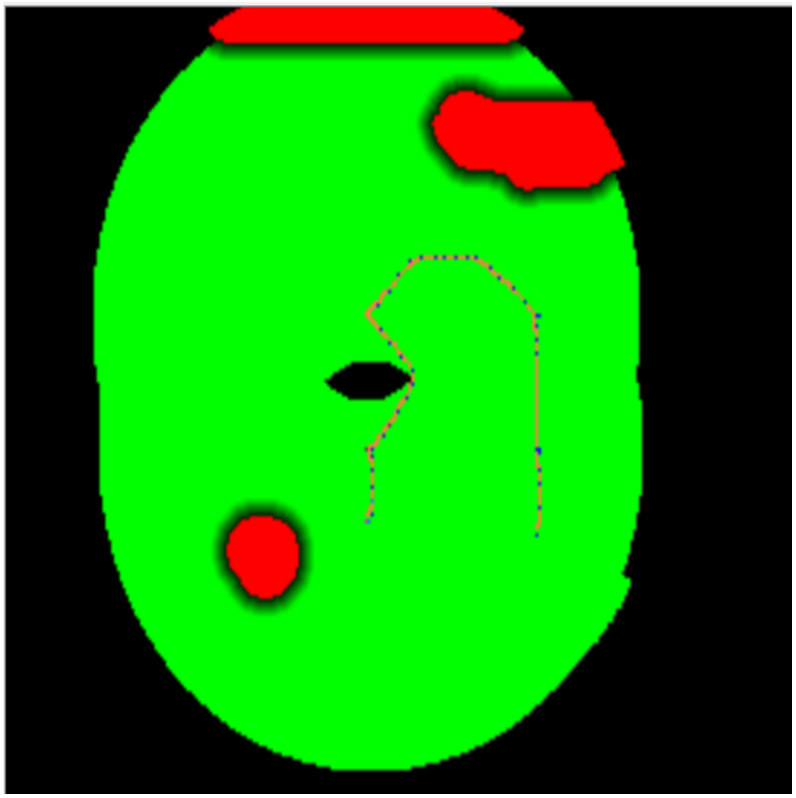


Fig. 21c Final destination reached in formation

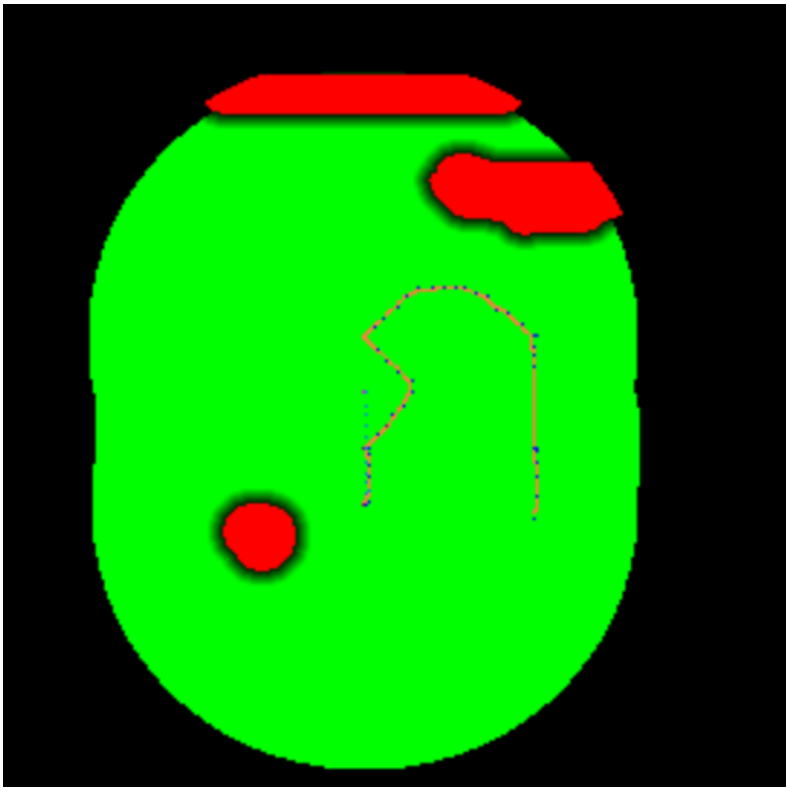


Fig. 21d. Master's Path planning aiming to new final destination

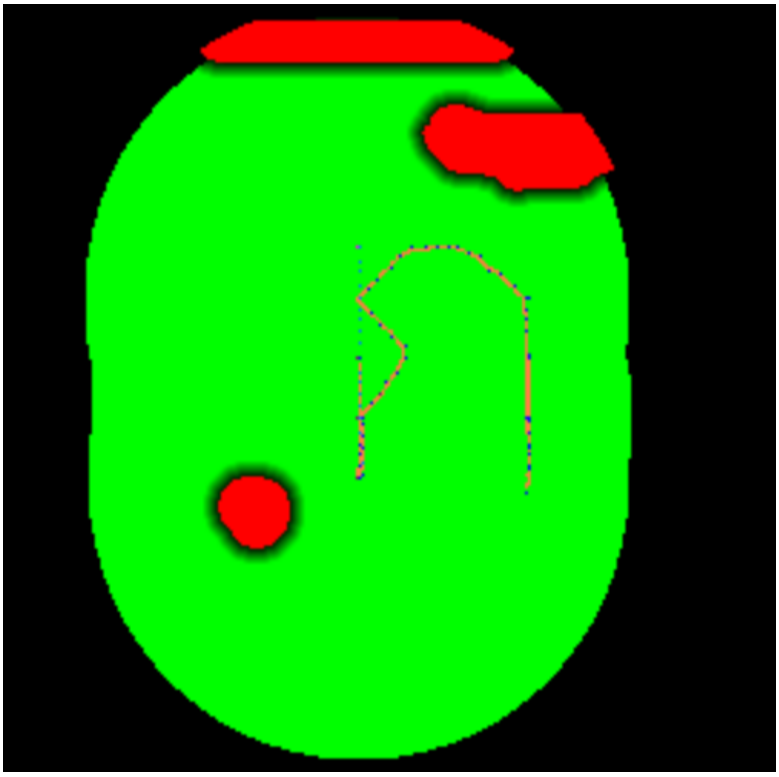


Fig. 21e. Moving in formation and master's path planning

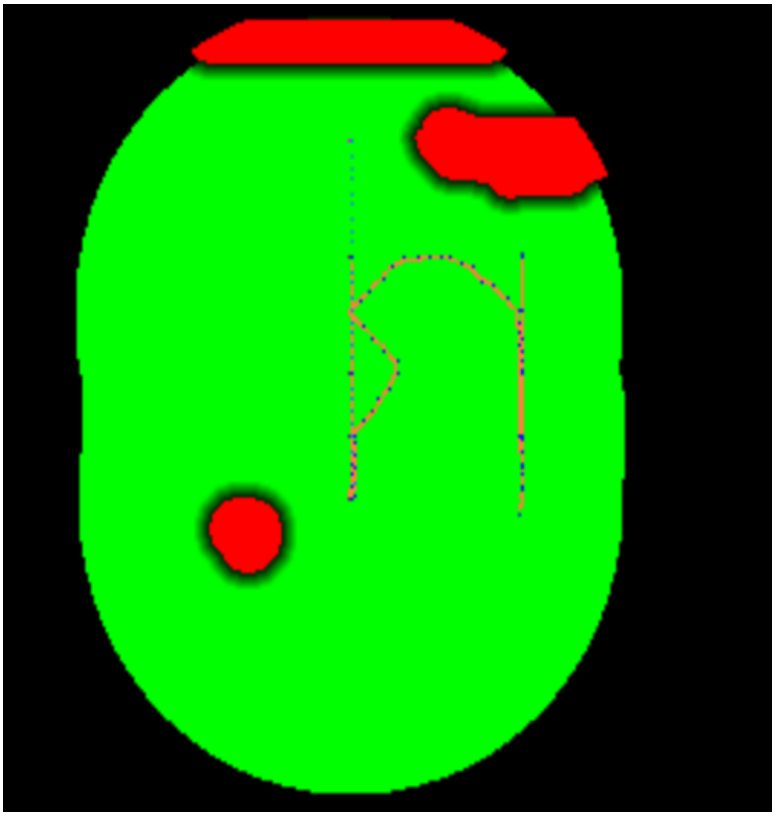


Fig. 21f. Moving in formation and master's path planning

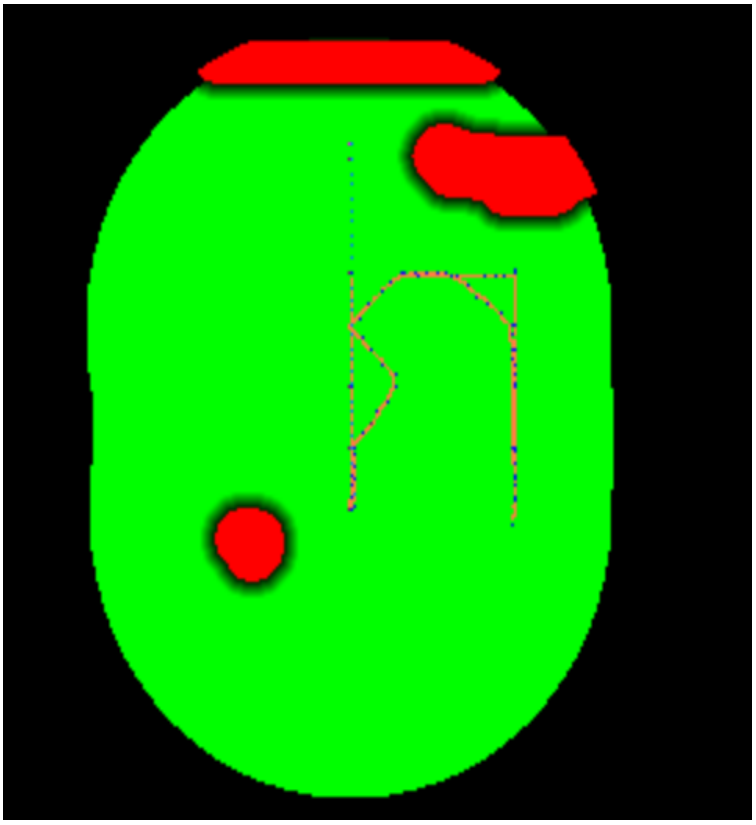


Fig. 21g. Formation change due an error: Slave can't reach destination

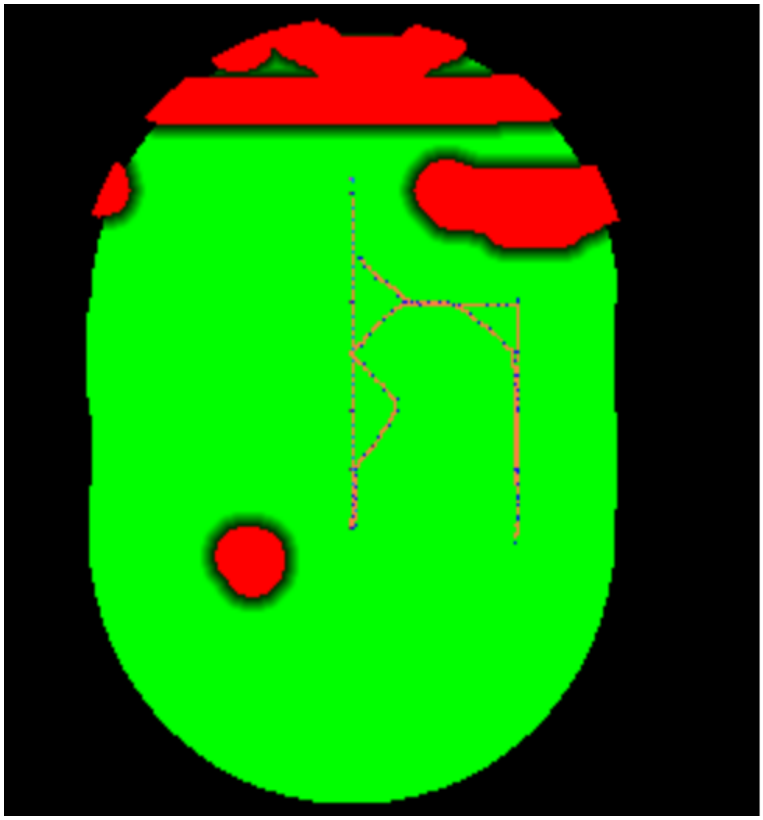


Fig. 21h. Formation changing due an error

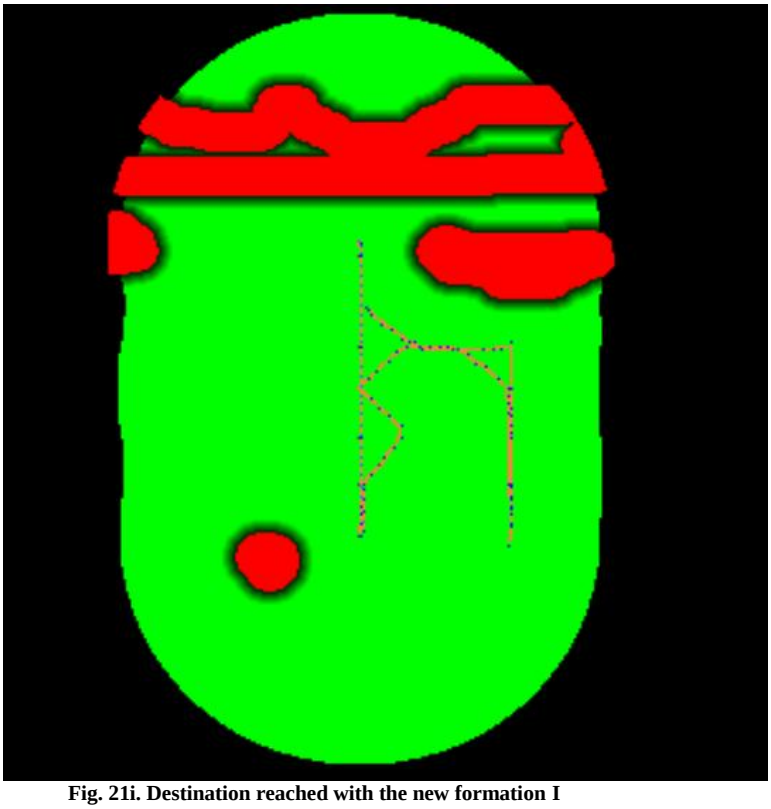


Fig. 21i. Destination reached with the new formation I

Fig. 21a, 21b and 21c show the executing of the order **GO2_COP -3 0 0 H S 3 1**, what means move in a horizontal formation with the slave first and a separation of 3m, with destination (-3,0)

In fig. 21a it can be seen the slave performing the first formation heading to the final destination.

Fig. 21b shows the first movement in formation. The formation can't be kept because the master robot is trying to get out of the unsafe area. Fig. 21c shows the arrival.

Figs. from 21d to 21i show the executing of the order **GO2_COP 3 0 0 H M 3 1**, what means move in a horizontal formation with the slave first and a separation of 3m, with destination (3,0).

Fig. 21d shows the path planning to the new destination coordinates, departing from the last position reached.

Figures 21e and 21f show the robot moving in formation until is impossible for the slave robot keep in formation. At this point an error occurs, the sub goal for the slave is inside an obstacle, so this robot reports an error and the master reacts sending the order of change the formation and continues moving.

This behavior can be seen in figs. 21g and 21h where the slave robot changes drastically its trajectory to form in Indian line behind the master robot (In a safe formation), until the robots reach the final destination.

In this case the goal was unreachable in H formation but due the formation change the robots were capable of reaching the goal.

5.2. AUTONOMY_COORDINATE (Slave side)

On the slave side, the robot waits orders from earth. With the command “SLAVE” the robot goes into slave mode. First, after receiving this order, the robot will try to communicate with the master through the LAN and register for further communication between them. After this the slave will wait for further instructions but from the master only, until the cooperative mission is completed or if there is an error during the navigation.

In case success or error of the global cooperative mission, the robot will stop being in slave mode and will await new orders from earth.

5.2.1. General description of the algorithm

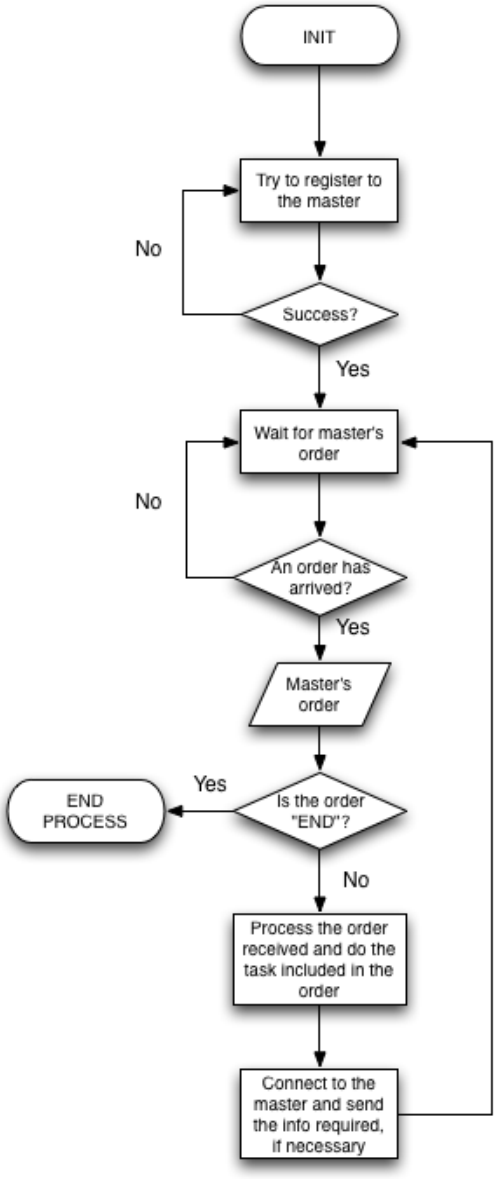


Fig. 22. Autonomy coordinate (slave side) flow diagram

1. Registration
2. Wait instruction from master
3. Process the instruction received
4. Do the task included in the instruction
5. Report back if necessary
6. Return to step 2

5.2.2. Specific description of the program

The AUTONOMY_COORDINATE on the slave side can be divided in the following parts:

5.2.2.1. Registration

In this step the robot will try to connect via LAN to the master, knowing the master's IP address by default, until it succeeds. Just after it successfully connects to the master, it will send his own IP address, which is read from the system inside the program (this way if the router changes the slave IP address it won't be a problem), so the master can identify it for further communications.

Once the IP address has been send correctly, the slave will enter into a "Listening Mode", waiting for future instructions from the master

5.2.2.2. Order Processing

At this point the slave will wait for master's instructions and the master will sent different instructions to the slave through the LAN. This orders received from the master are processed and different actions are made to satisfy the requirements from the master.

As said before, the master orders are **"GET_POSITION"** and **"GO_TO X Y"**, but the code is easily adaptable to more orders, because all the order processing is made through a "switch case", in which every order code is separated from the others.

The orders specific processing is shown next:

- GET_POSITION: When this order is received the robot reads the data of the odometry to get his actual position. This is done through a call of a function already implemented in the TBRA. Then the robot prepares this data to be sent back to the master. The function mentioned is shown next:

```
      am.GetLocalisationDataFusionStatusData(statusLocalisat
ionDF);
      for(int i=0; i<12; i++)
      tmat.trmat[i] = statusLocalisationDF.currPos_tmat[i];

      clib::Tmatrix2Cpose(tmat,&robotPose);
      Localx=robotPose.pos.x; //Local x coordinate
      Localy=robotPose.pos.y; //Local y coordinate
```

- GO_TO X Y: In this case the robot calculates the distance to the destination and gives the order to move to the destination to the module NAVIGATOR. This order is N2T X Y H B, where X and Y are the final coordinates in meters, H is the heading in radians and B is the boundary in meters, which is the radio of the area where the robot can consider inside the goal and end the navigation. The communication with the module NAVIGATOR is done with the next code:

```
cmd2Navigator.counter++;
sprintf(cmd2Navigator.command, "NAVTOTARGET %f %f
0.0 0.2", dxdydh[0], dxdydh[1]); //Heading 0.0 rad,
Boundary 0.2 m
 am.SendCommand_Navigator(cmd2Navigator);
```

  After this the robot waits for the report of the module NAVIGATOR: in case of success it prepares to report the "SUCCESS" to the master, or in case of error it reports back to the master the "ERROR" message so the corrections can be made in the master to the formation calculations.

5.2.2.3. Reports/Info Sending

Now the robot prepares itself to report back to the master, with the odometry information or the navigator report. In the first case is easier because the master is waiting for an immediate answer and the channel of communication is still open so the process is instantly. In the case of sending to the master the navigator report the slave will do a process similar to the registration in which it will try to communicate with the master until succeeds, and will inform that the task has been completed or has been an error.

To manage these two different kinds of process of sending data, two flags have been created so the robot knows if it has to report immediately or if it has to search the master in the network and try to connect to it: `datarequest` which means instant report; and `reportflag` which mean that the slave will have to search the master in the network.

5.2.2.4. End of Mission

At this point the server will send an "END" order, so the slave will know that the mission has been completed, or to be more precise, that the cooperative mode is ended, due to the success of the mission or due to a "fatal" error in the development of the mission.

In this case the robot closes all the SOCKETS and returns to the initial state where it receives orders directly from earth.

6. REAL TESTS

6.1. RELATIVE POSITION ESTIMATION OF A SPECIFIC OBJECT: TARGET TRACKING MODULE

Depth perception test

This test was done in the laboratory, using the 3-eyes stereo camera PointGrey Research Bumblebee XB3 mounted in a base in the laboratory. A red circle printed in paper was used as target. The real distance between the reference frame of the camera and the target was 2.5m

The environment of this prove is shown in fig. 23.

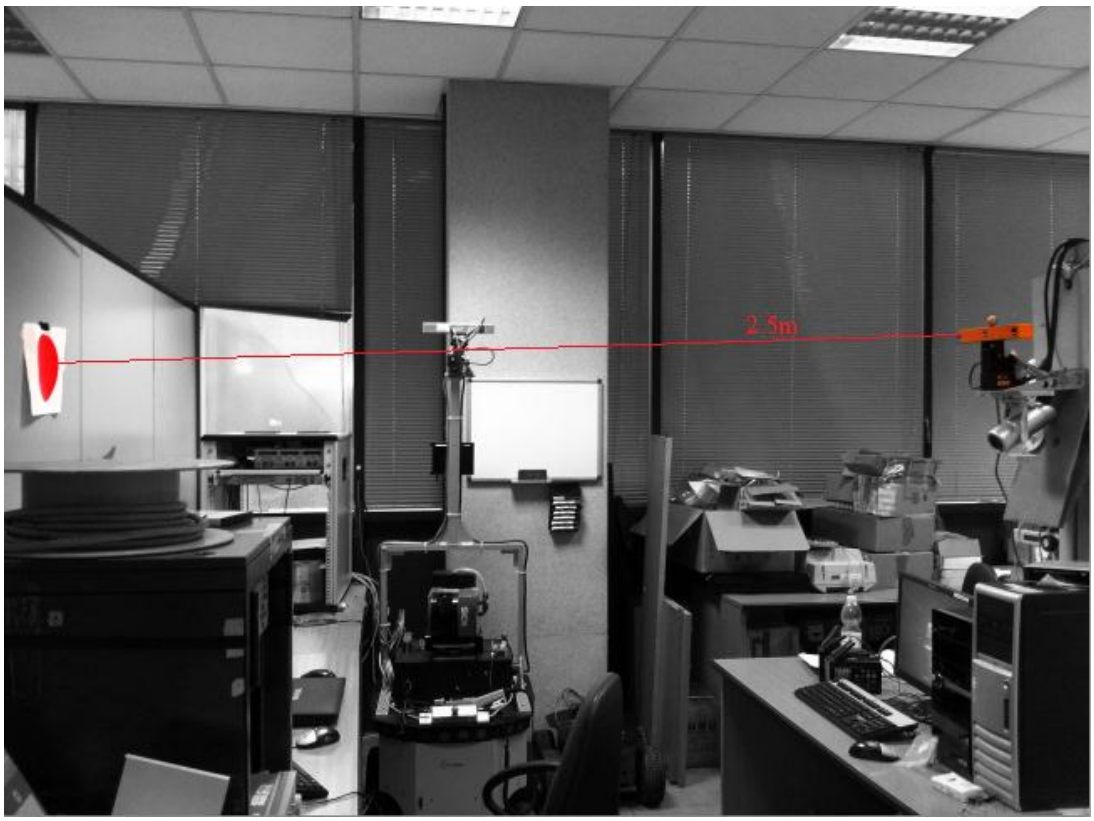


Fig. 23. Target tracking prove environment

Results:

The program calculates the distance values continuously for a predetermined period of time. For this test the time period was set to 10s.

The mean distance measured with the Target tracking module was 2.42m what means an error of 0.08m that is under the tolerance range of 20cm.

The next test for the Target tracking module was done as a part of the test for the Coordinator module to be explained.

Real time tracking test

Using the printed red circle as a target, we set the tracking time to 180s and move the circle to different positions expecting the stereo-camera to move following the moving of the target.

Fig. 24. Tracking the moving target in real-time

Results:

For slow and short moves of the red circle, the camera has moved following the target. The response had some delay.

6.2. MOVE IN FORMATION: COORDINATOR MODULE FOR TBRA

Indian line formation

For this test the order given to the robot was **GO2_COP 6 0 0 S I 2 1** what means move in Indian line formation with destination (6,0) and heading angle h=0°, with the slave first and with separation 2m.

As said before the initial position of the slave robot in its own reference frame is (0,0) but in the master reference frame is different. At this point is not possible to change it unless the slave robot moves.

To correct the coordinates sent to the slave robot, the robot was placed 2m in front of the master robot and this distance was used as a translation applied to the calculated slave's destination coordinates. A rotation is not needed because at this point the heading angle is set to 0°.

The test was performed in the hallway of the laboratory because at the moment it is not possible to take the robots out of the building, that's why the H formation was only performed in simulation.

Results:

Once the order was sent to the robots, the target tracking task and the path planning task started; and took a few seconds generating a delay in the movement.

After the mentioned tasks the first formation was performed. The slave robot started moving and eventually reached its formation destination. The final position of the slave was between the error margins.

Once the first formation was done, the master robot started moving and a few seconds after the slave robot started moving, performing the movement in formation until the first sub goal was reached.

Once the sub goal was reached, the master started again the path planning task and the target tracking task. After a few seconds the master started moving and then the slave started moving.

This process was repeated until the master robot reached its final destination.

The Indian line formation was kept with some heading angle errors as expected. During each movement the robots tried to keep the 0° heading angle but in long distances, small errors generate significant deviations in the trajectories.

Fig. 25a. Initial formation

Fig. 25b. Intermediate point formation

Fig. 25c. Final formation

The images above show the initial formation, an intermediate point and the final formation. It can be seen how in the first moment the robots are in an almost perfect formation, but after the movement, the formation is not perfect due the heading angle errors during the trajectory.

In short, the test reveals a good calculation in the formation coordinates with several delays due the target tracking task and the path planning task. Also there was an error in the heading angle as expected. The formation was kept under the conditions explained in this thesis.

The robots coordinates were measured and the results are shown below:

Step	Master's x coordinate (m)	Master's y coordinate (m)	Slave's x coordinate (m)	Slave's y coordinate (m)
Initial Formation	0	0	1,8	0
1th group displacement	1,9	0	3,9	0
2nd group displacement	4	0,1	5,8	-0,2
3th group displacement	5,9	0,3	7,9	-0,5

7. CONCLUSIONS

1. The cooperative tasks in a space mission permits to take advantage of the heterogeneity of the robots that makes some tasks easier and permits the obtaining of more kinds of information.

2. A space exploration mission could become more efficient, adopting a formation move. Because each robot makes its own map, using for example the H formation, a bigger area could be explored. The exchange of generated maps information could increase the performance of this task.

3. The algorithm developed works even when only one robot is capable of the map generation, in our case the master robot. The slave robots can navigate based only in the instructions given by the master robot. This statement represents an advantage when in mission, for any reason the other robots are not able to generate its own map.

4. Even when in the trajectories between intermediate points the formation could seem broken or inexact, the movement approaches very well to a movement in formation constant in time. The algorithm only guaranties the formation in the intermediate points.

5. As the number of robots increases, the delay generated by the communications could become significant. However, the number of robots will not increase too much because of the payloads and the costs that a big number of robots implicate in a space mission.

6. In this moment, the planning of the trajectories based on images processing generates significant delays due the "stop-and-go" navigation politics. Future develops will increase the images processing performance and will adopt continue navigation politics that will speed up the path planning and navigation task.

7. To perform a movement in formation constant in time, is needed a path planning projected for this scope, that could plan the trajectory, in parallel with the movement of the robot.

8. The algorithm is written and projected to be scalable and to be easy to modify. To make any type of formation it is enough to place the robots in the desired formation for the first time and then, they will move keeping the formation until the master robots reaches the final destination.

9. The detection of the heading angle of the robots would increase the performance of the algorithm because in the real test it is difficult to keep a heading angle equal zero and this generates significant deviations in the trajectories.

10. The modular architecture of the TBRA gives a lot of flexibility to the module's developer, because the module can be tested singularly, even when not all the robot modules are working when the module is being tested.

8. FUTURE WORKS

1. Make the path planer work in a way that the route generated also tries to keep the formation in a cooperative mission.

2. Make the path planning works in association with other robots path planning, in the way two or more maps generated by different robots could be merged obtaining bigger maps in a shorter period of time.

3. Make the path planning module work in parallel with other modules like the navigator module and the cooperative module, would increases the performance in developing a cooperative task.

4. Implement more orders that could be useful in a space mission taking advantage of the heterogeneity of the robots, like a cooperative pick and transport of a desired sample, or a shared map generation mode.

5. Take advantage of the formation properties explained in conclusion 8 and implement other formation types using more than 2 robots.

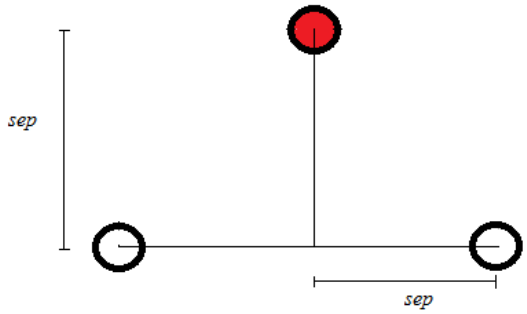


Fig. 26. Triangular formation

For example to keep a 3 robot triangular formation as shown in fig 33, would be enough to set the position of the slave robots as follows:

$$Y_s = Localy + sep$$

$$X_s = Localx + sep(-1)^i$$

Where (X_s, Y_s) are the formation coordinates for the slaves and $(Localx, Localy)$ are the master's coordinates. i is a variable incremented each time the move order is sent to a slave and must be set to 0 after each formation movement.

More complex formations with more robots can be performed calculating the function for the first formation coordinates for the slaves. Due the properties of the algorithm the formation will be kept until the master reaches the final destination.

6. For the heading angle detection, an algorithm of images processing can be implemented by means of the cameras mounted in the robots used for this thesis. The cameras could be set to detect a

shape instead of detect a color. A colored sphere can be mounted in the robots with a patron of colors that permits the detection of the heading angle using the images processing.
Supposing the master robot is tracking a specific slave robot, the camera will identify the sphere mounted in the robot and will percept two predominant colors in the sphere.

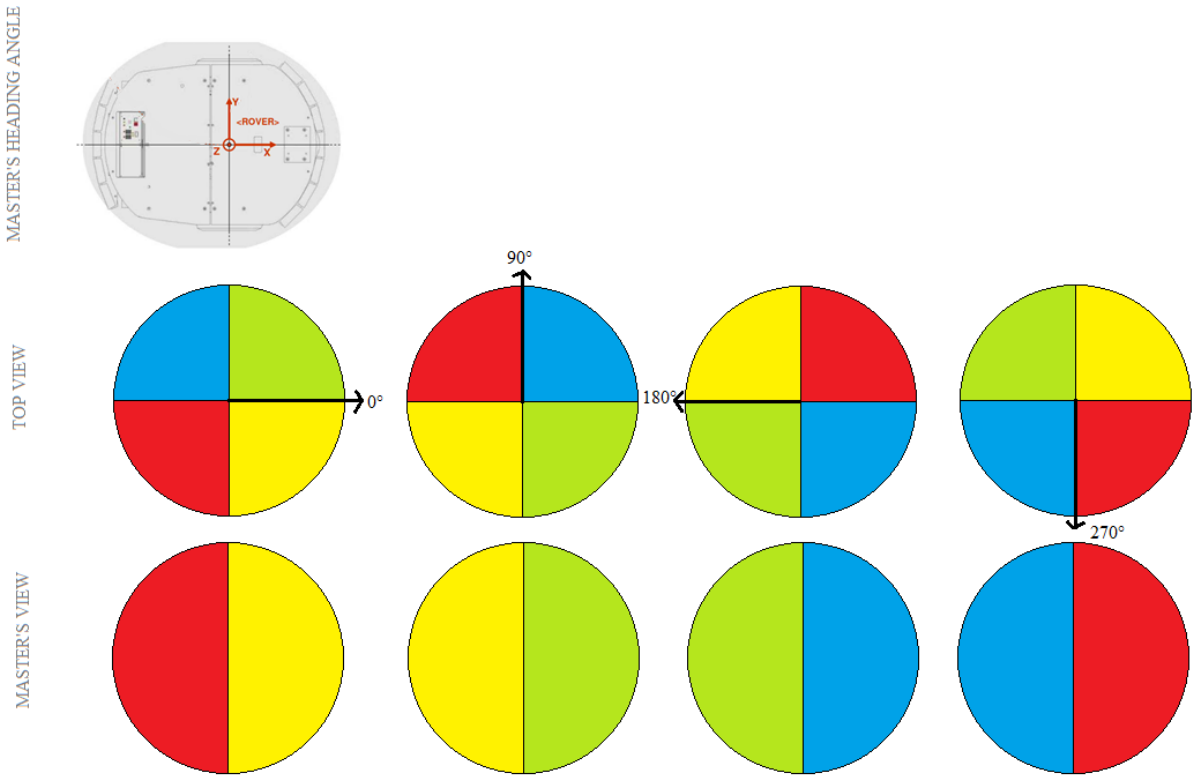


Fig. 27. Heading angle detection based on the sphere colors proportion concept example.

For this example is assumed that the master robot is at the right side of the slave robot and that the y-axis coordinates for both robots are the same. The camera orientation angles are neglected and the heading angle of the master robot is oriented in the x-axis direction. The slave's heading angle detection would be relative to the master's heading angle.

Four possible perceptions of the sphere made by the master robot are shown in fig. 34 Depending on the detected colors and on its proportion, the slave's heading angle could be estimated. For example, if the two predominant colors are red and yellow, the slave's heading angle is near 0°. A more exact estimation could be done using the proportion, for example a proportion of 50% red and 50% yellow, means 0° exactly.

For intermediate angles, is possible that more than two colors could be seen but two colors will be predominant.

7. The code written for this thesis is designed only for Windows, so it must be ran remotely from an external computer (the modules must connect through the Wi-Fi router to the TBRA, that runs on Linux, at the start up of the robot). In the future this code must be modified so it can be ran directly on the robot to avoid any extra communication delays, so the proper modifications must be done in the code to make it OS independent.

9. REFERENCES

[1] Tuan-Jie Li, Gao-Wei Yuan, Fei-Jun Wang. “Behavior Control of Multiple Robots Exploring Unknown Environment”. July, 2009. IEEE.

[2] Filippo Arrichiello. “Coordination Control of Multiple Mobile Robots”. November 2006. Tesi dell’Università Degli Studi Di Cassino

[3] http://ants.gsfc.nasa.gov/ArchandAI.html

[4] MEPAG 2-Rover International Science Analysis Group (2R-iSAG) “Two rovers to the same site on Mars, 2018: Possibilities for Cooperative Science”. July 10, 2010.

[5] Shashank Srivastava, G.C Nandi “Localization of Mobile Robots in a Network using Mobile Agents” 2010 IEEE.

[6] Andrew Howard, Maja J Matarid and Gaurav S. Sukhatme “Putting the ‘I’ in ‘Team’: an Ego-Centric Approach to Cooperative Localization” 2003 IEEE.

[7] Ling Wang, Xuanping Cai, Weihong Fan, Hengyang Zhang “An Improved Method for Multi-Robot Cooperative Localization Based on Relative Bearing” 2008 IEEE

[8] F. Dellaret, D. Fox, W. Burgard, S. Thrun “Monte Carlo Localization for Mobile Robots" 1999 IEEE.

[9] Chuho Yi, Il Hong Suh, Gi Hyun Lim, and Byung-Uk Choi “Bayesian Robot Localization using Spatial Object Contexts” 2009 IEEE/RSJ International Conference on Intelligent Robots System.

[10] Yuanqing Lin, Paul Vernaza, Jihun Ham, and Daniel D. Lee “Cooperative relative robot localization with audible acoustic sensing”

[11] J. Spletzer, A. K. Das, R. Fierro, C. J. Taylor, V. Kumar, and J. P. Ostrowski “Cooperative localization control multi-robot manipulation” 2001 IEEE/RSJ International Conference on Intelligent Robots System

[12] Yi Feng, Zhigang Zhu, Jizhong Xiao. “Heterogeneous Multi-Robot Localization in Unknown 3D Space”. Juny, 2006. IEEE

[13] Ross Mead, Rob Long, and Jerry B. Weinberg, "Fault-Tolerant Formations of Mobile Robots". March, 2009. IEEE.

[14] Edwin Carvalho, Miguel Pedro Silva and Carlos Cardeira “Decentralized Formation Control of Autonomous Mobile Robots” 2009 IEEE

[15]Huaxing Xu, Haibing Guan, Alei Liang, inan Yan “A Multi-Robot Pattern Formation Algorithm Based on Distributed Swarm Intelligence”

[16] Zhao Pengchong, Liang Alei, Liu Liang, Chen Ying, Guan Haibing, Yan Xinan. “An Exploration Algorithm for A Swarm of Homogeneous Robots”. September, 2009. IEEE

[17] Nicolas Hutin, Claude PCgard, Eric Brassart . “A Communication Strategy for cooperative robots”. October 1998. IEEY/RSJ

[18] Tucker Balch, Ronald C. Arkin. “Behavior-based Formation Control for Multi-robots Team”. 1999. IEEE

[19] Ross Mead, Jerry B. Weinberg, and Jeffrey R. Croxell. “A Demonstration of a Robot Formation Control Algorithm and Platform”. 2007. IEEE

[20] J¨urgen Leitner .“Multi-robot Formation Control Using Leader-follower for MANET”. IEEE

[21] Jakob Fredslund, Maja J Mataric’. “Robots in formation using local information”. IEEE

[22] Javaid Khurshid, Hong Bing- Rong, Gao Qing- Ji . “Dynamic Growth of Robot Formation Using Only Local Sensing and Minimal Communication”. December 2004. International Conference on Control and Automation. IEEE

[23] Sérgio Monteiro Estela Bicho. “Robot formations: robots allocation and leader–follower pairs”. May, 2008. IEEE International Conference on Robotics and Automation.