2009

Penn State University

Mechanical & Nuclear Department

Joan Singla Milà

[SOFTWARE DESIGN FOR IGVC COMPETITON]

Description of the software design to acquire different sensor data and process it to map the environment surrounding of the robot to plan a path to explore it or to get different GPS points.

ABSTRACT

The project here described is about the hardware and software design of a robot for the IGVC competition. The whole product was designed, built and coded in less than 9 months to compete on this international contest that brought more than 30 teams together.

Thanks to a couple of sensors, a laser which scans the region to detect obstacles and a camera to process the painted lines, the robot had the ability to process all this information and negotiate the barrels, fences, lines… completely autonomous to explore the environment and reach a set of GPS locations.

What makes this project mostly different from the other ones (not only from IGVC '09 also the previous years) was the use of MATLAB/Simulink to code all the program structure and algorithms. The use of this language has inherent benefits like the easiness to be debugged that enabled the creation of better code that otherwise would be much more difficult.

Connecting with the previous point, the main subject of this project is the software of the robot for the IGVC. It starts describing the basic system structure and how it was created using a basic design concept and taking into account the IGVC requirements and course characteristics. Then every single structure is analyzed and explained as well as the different algorithms and strategies to success on each problem.

To sum up, the structure and algorithms here presented are not a final diagram or a fix strategy, the code shown it is just a start point subject to multiple modifications and later iterations because the more you test it, the more you discover new weak points, new ways to resolve situations…

TABLE OF CONTETNS

Chapter 1 | INTRODUCTION

This document describes the work done in Penn State University as part of the IQS research master which took place between October 2008 and June 2009. The main aim of this master was to develop and test a program to run the IGVC contest.

1.1 IGVC contest description

The IGVC offers a design experience that is at the very cutting edge of engineering education. It is multidisciplinary, theory-based, hands-on, team implemented, outcome assessed, and based on product realization. It encompasses the very latest technologies impacting industrial development and taps subjects of high interest to students.

Design and construction of an Intelligent Vehicle fits well in a two semester senior year design capstone course, or an extracurricular activity earning design credit. The deadline of an end-of-term competition is a real-world constraint that includes the excitement of potential winning recognition and financial gain.

Students at all levels of undergraduate and graduate education can contribute to the team effort, and those at the lower levels benefit greatly from the experience and mentoring of those at higher levels.

Team organization and leadership are practiced, and there are even roles for team members from business and engineering management, language and graphic arts, and public relations. Students solicit and interact with industrial sponsors who provide component hardware and advice, and in that way they are able to get an inside view of the industrial design and look for job opportunities.

1.1.1 Technologies Involved

The technologies involved in the IGVC come from a wide range of disciplines and are those of great current interest in both industry and engineering education. Engineering students in all disciplines today would do well to have a familiarity with each of the technologies described in the Table 1, Table 2 and Table 3.

ELECTRICAL ENGINEERING			
Detectors	Voltage Regulation	Electrical Protection	Potentiometers
Sensors	PWM Amplifiers	Stepping Motors	Radio Controlled Stop
Ultrasonics (SONARS)	Traction Motors	Power Requirements	Multiplexing
Radar Ranging	Actuators	Servo Systems	Batteries

Table 1 | IGVC technologies related with electrical engineering

COMPUTER SCIENCE ENGINEERING			
Intelligent Decision making	Computer Programming	Software System Interfaces	Computer Modeling & Simulation
Neural Networks	Machine Vision	Data Acquisition	Computer Graphics
Image Analysis	Intelligent Control	MATLAB	Guidance Systems
Fuzzy Logic	Software Engineering	Thresholding	Video Cameras
Multiple CPUs	Artificial Intelligence	PID Controllers	Control Algorithms
Data Fusion	Microcontrollers	Frame Grabbers	

Table 2 | IGVC technologies related with computer science engineering

MECHANICAL ENGINEERING			
Mobility	Autonomous Systems	Thermal Management	Computer Aided Design
Robotics	Traction	Component Packaging	Body Styling
Mobile Robots	Vehicle Dynamics	Chain Drives	Bump Sensors
Speed Control	Power Requirements	Differential Drive	Finite Element Stress Analysis
Power Sources	Engineering Mechanics	Suspension Systems	Welding
Weight Distribution	Tire Treads	Reliability/Durability/Maintainability	
Bearings	Turning Radius	Solid Modeling	Fiberglass Forming
Weather Proofing	Rolling Friction	Hydraulic Drives	Tachometers
Dampers	Articulation		

Table 3 | IGVC technologies related with mechanical engineering

1.1.2 Applications of IGVC in the real world

The technologies involved in the IGVC are those of emerging and burgeoning industries today. Among those applications are many with great opportunities for breakthroughs and innovation, and employment opportunities for knowledgeable young engineers abound. Table 4, Table 5 and Table 6 resume them:

MILITARY MOBILITY	
Mine Detection	Leader - Follower
Platooning	Mobile Robots
Lane Detection & Following	Surveillance Systems
Unmanned Weapons Deployment	

Table 4 | IGVC applications related with military mobility

INTELLIGENT TRANSPORTATION SYSTEMS (ITS)	
Collision Avoidance	Adaptive Cruise Control
Obstacle Detection	Leader - Follower
Lane Detection & Following	Driver Aides
Lane Departure Warning	Vehicle Safety Systems
Automated Highway Systems	Navigation Systems
Unmanned Maintenance Vehicles	

Table 5 | IGVC applications related with intelligent transportation systems (ITS)

MANUFACTURING	
Mobile Robots	Machine Safety
Machine Operations	Material Handling
Unmanned Storage Systems	

Table 6 | IGVC applications related with manufacturing

1.1.3 Vehicle configuration

The competition is designed for a small semi-rugged outdoor vehicle. Vehicle chassis can be fabricated from scratch or commercially bought. Entries must conform to the following specifications:

- *Design:* Must be a ground vehicle (propelled by direct mechanical contact to the ground such as wheels, tracks, pods, etc or hovercraft).
- *Length:* Minimum length three feet, maximum length seven feet.
- *Width:* Minimum width two feet, maximum width five feet.
- *Height:* Not to exceed 6 feet (excluding emergency stop antenna).
- *Propulsion:* Vehicle power must be generated onboard. Fuel storage or running of internal combustion engines and fuel cells are not permitted in the team maintenance area (tent/building).

- *Speed:* For safety, a maximum vehicle speed of five miles per hour (5 mph) will be enforced. All vehicles must be hardware governed not to exceed this maximum speed. No changes to maximum speed control hardware are allowed after the vehicle passes Qualification.
- *Mechanical E-stop location:* The E-stop button must be a push to stop, red in color and a minimum of one inch in diameter. It must be easy to identify and activate safely, even if the vehicle is moving. It must be located in the center rear of vehicle at least two feet from ground, not to exceed four feet above ground. Vehicle E-stops must be hardware based and not controlled through software. Activating the E-Stop must bring the vehicle to a quick and complete stop.
- *Wireless E-Stop:* The wireless E-Stop must be effective for a minimum of 50 feet. Vehicle E-stops must be hardware based and not controlled through software. Activating the E-Stop must bring the vehicle to a quick and complete stop. During the competition performance events (Autonomous Challenge and Navigation Challenge) the wireless E-stop will be held by the Judges.
- *Payload:* Each vehicle will be required to carry a 20-pound payload. The shape and size is approximately that of an 18" x 8" x 8" cinder block. Refer to section I.3 Payload.
- *Apriori Data:* The intent is to compete without apriori or memorized data. Course position data should not be mapped/stored. This is difficult to enforce, each team is expected to comply with the intent. Both the Autonomous Challenge and Navigation Challenge courses will be changed after each heat and between runs to negate any memorization or course familiarization techniques.

1.1.4 IGVC challenges

The IGVC competition is divided into 3 main categories: Autonomous, Design and Navigation challenge and one more JAUS which is optional. Each of them is independent from the others.

1.1.4.1 Qualification

All vehicles must pass qualification to receive standard award money in the Design Competition and compete in the performance events (Autonomous Challenge and Navigation Challenge). To complete Qualification the vehicle must pass/perform the following eight criteria.

- *Length:* The vehicle will be measured to ensure that it is over the minimum of three feet long and under the maximum of seven feet long.
- *Width:* The vehicle will be measured to ensure that it is over the minimum of two feet wide and under the maximum of five feet wide.
- *Height:* The vehicle will be measured to ensure that it does not to exceed six feet high; this excludes emergency stop antennas.

- *Mechanical E-stop:* The mechanical E-stop will be checked for location to ensure it is located on the center rear of vehicle a minimum of two feet high and a maximum of four feet high and for functionality.
- *Wireless E-Stop:* The wireless E-Stop will be checked to ensure that it is effective for a minimum of 50 feet. During the performance events the wireless E-stop will be held by the Judges.
- *Max Speed:* The vehicle will have to drive at full speed over a prescribed distance where its speed will be determined. The vehicle must not exceed the maximum speed of five miles per hour. No change to maximum speed control hardware is allowed after qualification. If the vehicle completes a performance event at a speed faster than the one it passed Qualification, that run will not be counted.
- *Lane Following:* The vehicle must demonstrate that it can detect and follow lanes.
- *Obstacle Avoidance:* The vehicle must demonstrate that it can detect and avoid obstacles.
- *Waypoint Navigation:* Vehicle must prove it can find a path to a single 2 meter navigation waypoint.

During the Qualification the vehicle must be put in autonomous mode to verify the mechanical and wireless E-stops and to verify lane following and obstacle avoidance. The vehicle software can be reconfigured for waypoint navigation qualification. For the max speed run the vehicle may be in autonomous mode or joystick/remote controlled. Judges will not qualify vehicles that fail to meet these requirements. Teams may fine tune their vehicles and resubmit for Qualification. There is no penalty for not qualifying the first time. Vehicles that are judged to be unsafe will not be allowed to compete. In the event of any conflict, the judges' decision will be final.

1.1.4.2 Autonomous challenge

A fully autonomous unmanned ground robotic vehicle that must negotiate around an outdoor obstacle course under a prescribed time while staying within a pair of lines, respect the 5 mph speed limit and avoid the obstacles on the track. Figure 1 shows different examples of obstacles configurations.

Judges will rank the entries that complete the course based on shortest adjusted time taken. In the event that a vehicle does not finish the course, the judges will rank the entry based on longest adjusted distance travelled. Adjusted time and distance are the net scores given by judges after taking penalties, incurred from obstacle collisions, pothole hits, and boundary crossings, into consideration.

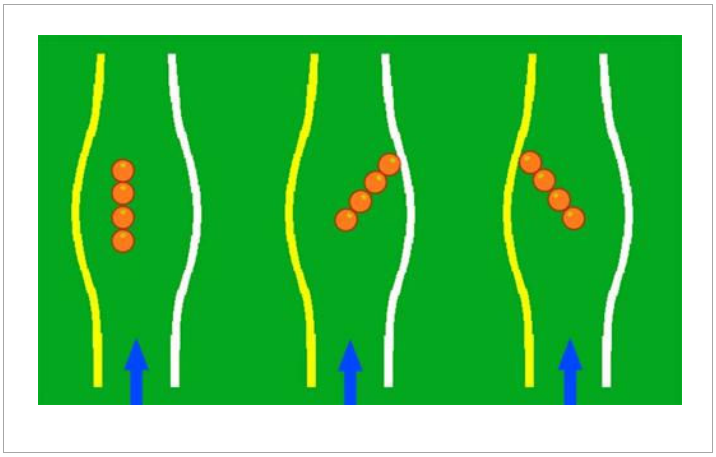

Figure 1 | Examples of obstacle configurations on the Autonomous Course

1.1.4.3 Design competition

Although the ability of the vehicles to negotiate the competition courses is the ultimate measure of product quality, the officials are also interested in the design strategy and process that engineering teams follow to produce their vehicles. Design judging will be by a panel of expert judges and will be conducted separate from and without regard to vehicle performance on the test course. Judging will be based on a written report (should not exceed 15 letter-sized pages), an oral presentation and examination of the vehicle.

Design innovation is a primary objective of this competition and will be given increased attention by the judges. Two forms of innovation will be judged: first will be a technology (hardware or software) that is new to this competition; and second will be a substantial subsystem or software upgrade to a vehicle previously entered in the competition. In both cases the innovation needs to be documented, as an innovation, clearly in the written report and emphasized in the oral presentation. Either, or both, forms of innovation will be included in the judges' consideration.

1.1.4.4 Navigation challenge:

Navigation is a practice that is thousands of years old. It is used on land by hikers and soldiers, on the sea by sailors, and in the air by pilots. Procedures have continuously been improved from line-of-sight to moss on trees to dead reckoning to celestial observation to use of the Global Positioning System (GPS).

The challenge in this event is for a vehicle to autonomously travel from a starting point to a number of target destinations (waypoints or landmarks) and return to home base, given only the coordinates of the targets in latitude and longitude. Figure 2 shows the typical configuration for a course used in the navigation challenge.

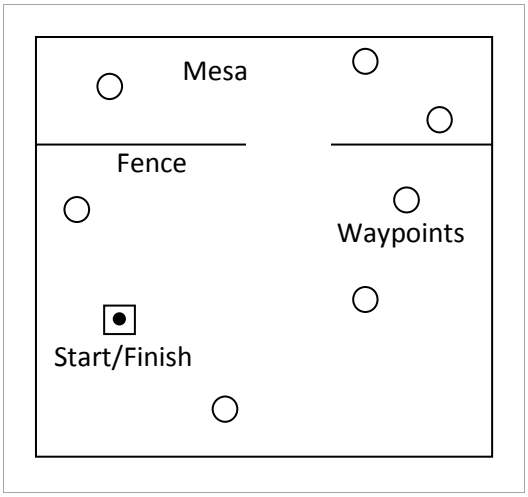

Figure 2 | Typical course configuration for the Navigation Challenge

1.1.4.5 JAUS challange

The Joint Architecture for Unmanned Systems (JAUS) is a set of standardized messages suitable for controlling all types of unmanned systems, and is soon to become an Aerospace Standard of the Society of Automotive Engineers (SAE).

There are two aspects to JAUS Challenge: a written/oral presentation which will be added to the Design Competition and a practical demonstration. The written/oral presentation shall include a description of the student team's implementation of JAUS in their design report and the practical demonstration will consist of JAUS messages begin sent to the vehicle from an IGVC developed JAUS control box via an 802.3 Ethernet link.

1.2 Design process

This is the second year that Penn State is participating in the IGVC, so we have focused this year on developing processes for continuous improvement. Further, from lessons learned last year, we have established three core design concepts as follows:

1.2.1 Use iteration to learn from mistakes quickly

To formalize an iterative design methodology, we used the V design approach shown in Figure 3. We analyzed the requirements and restrictions for the project to create a set of desired design objectives. Given these design objectives we decomposed them into subsystems and further to algorithms. The remainder of the report provides details of each decomposition and integration process.

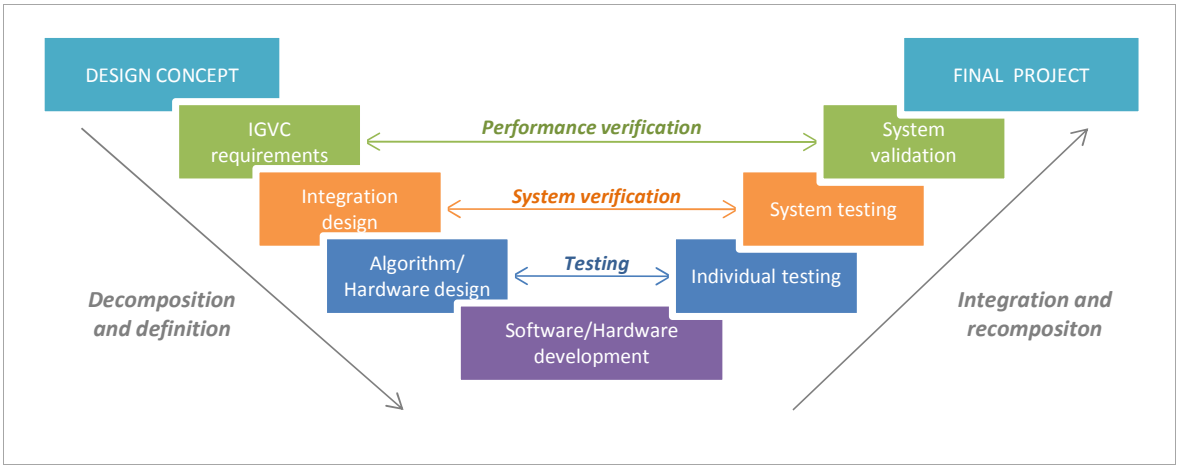

Figure 3 | The "V" Systems Engineering Model

1.2.2 Prepare for competition by using competition

For each algorithm, we divided the team in at least two groups that compete to develop the best solution. This way, performance becomes a first priority, multiple team members learn and hence can scrutinize each other's code, and team members become proud of their code. After each competition, we compared all solutions so that the best methods are shared.

1.2.3 Fast algorithms are better than fast execution

IGVC robot software will always be a prototype, and last year's entry taught us that debugging bad algorithms to create outstanding code has more performance payoff than using complied language or fast processors on otherwise bad code. MATLAB/Simulink is a high level language that supports this design philosophy, and it is OS, computer and version agnostic. For this reason it is used throughout the engineering curriculum to the point that PSU no longer teaches C or C++ to most engineering students.

1.3 Team Architecture

Because of our competition design principles, each team member was at least an expert on two groups shown in the Figure 4. Logs have been kept of all design activity, and the team has devoted 1500 hours to date towards the development of Penn State Lion One.

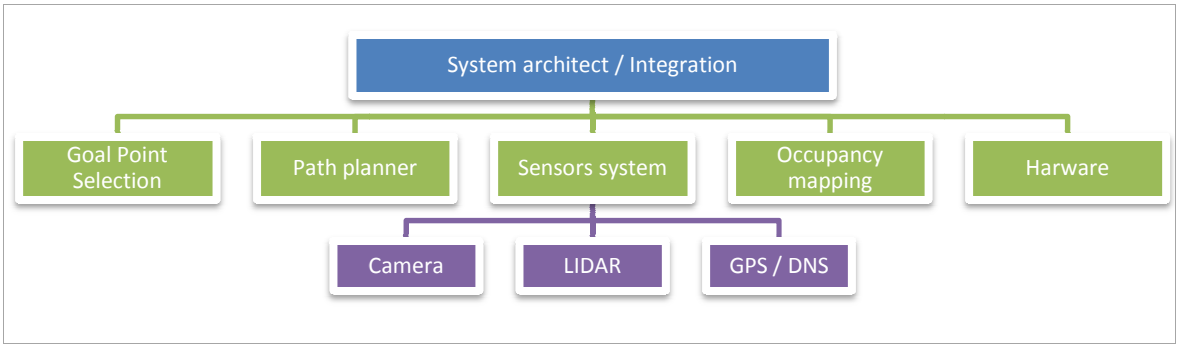

Figure 4 | Team architecture

In my case, inside the team I was involved mainly in developing the software side; taking the lead on programming the path planner and the goal point selection as well as the system architect/Integration.

Chapter 2 | HARDWARE DESIGN

Although my main task in the team was not related with the hardware I thought it would be advisable to include a brief description of the robot and the equipment used to compete.

2.1 Platform description

Starting from the previous robot used in IGVC contest 2008, the team has followed an iterative design process to reach the final platform, dealing with all previously known hardware problems and new ones found on the way.

The four sketches in the Figure 5 show the different versions of the hardware platform and the main changes with regard to the previous one:

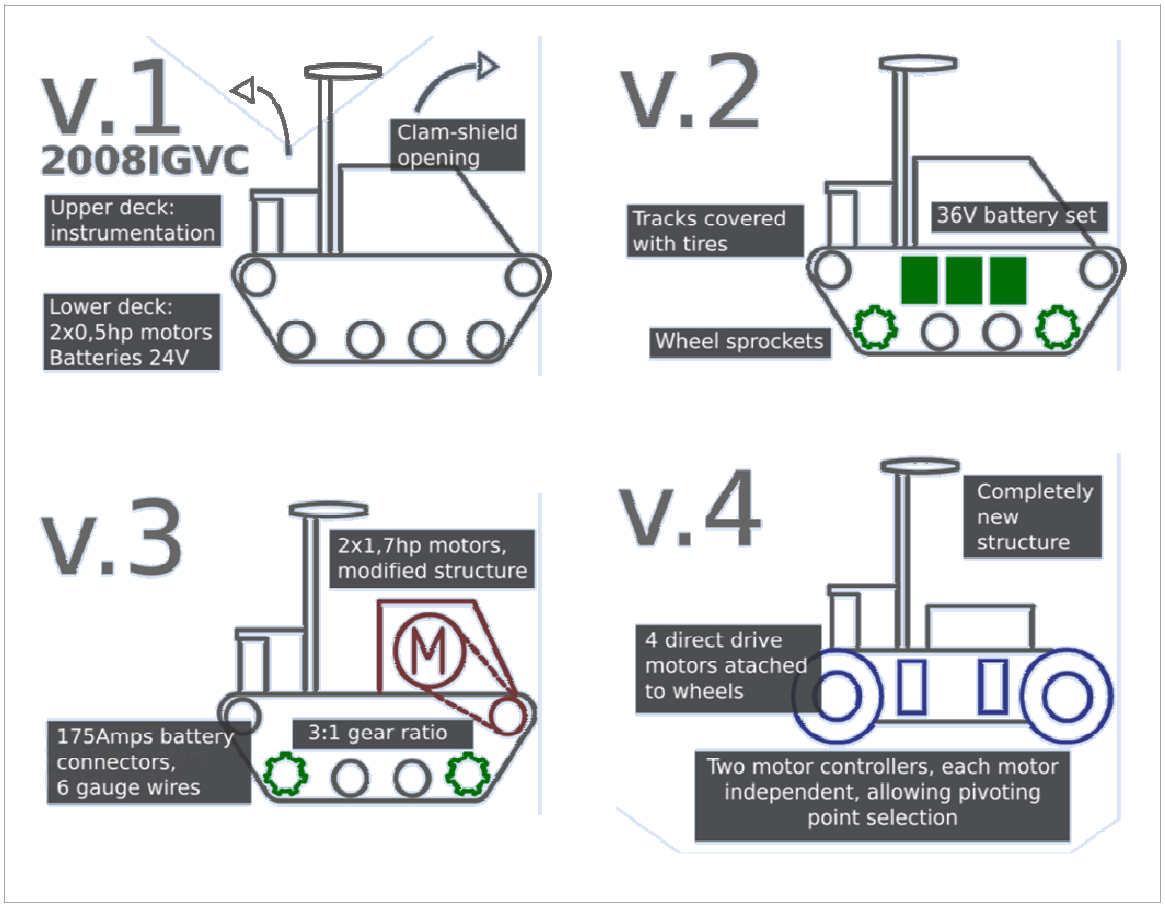

Figure 5 | Different platform versions for the hardware iterative design

Version 1 was the one used in IGVC 2008 and it was track driven and included a two-decked platform. The lower platform contains two 0.5 hp DC motors and four batteries, providing two separate 24V sources, one used by the motion system, the other by the instrumentation equipment. The upper deck holds all the electronics. The access to the lower deck is through a

clam-shelled opening system, which proved to be impractical. The main issues about this version were that sometimes the tracks derailed preventing the robot to keep moving and the speed was not enough to complete the course in a good position.

For version 2 the team focused on solving the problems of insufficient power and traction, increasing the motor voltage to 36V, adding guide sprockets wheels to the track system, and rubber overlay to the treads. With these changes, the robot was still barely fast enough and appeared a new problem: the guide sprockets significantly increased wear on the tracks.

Version 3 was used to test a new set of drive motors. These solved all the previous power issues, but further increased the rate of damage to our track system. Recognizing that design of a high-speed tank-drive UGV is a tradeoff between reliability and power consumption, and that reliability is always difficult to obtain in a prototype robot, we decided to migrate to a simpler and more effective wheeled robot.

Version 4, the final version used in the IGVC 2009 competition, was a four motor direct-drive wheeled vehicle. Although this approach had more limited traction and was more expensive, it had proven to be the most efficient and robust. Further, the way the motors were powered enabled independent control of the front and rear axles.

Figure 6 | Set of pictures of the 2009 IGVC robot

2.2 Electrical system and electronics

2.2.1 Power and communications

Several measures have been taken to ensure efficient power distribution while minimizing interference. As shown in Figure 7, the drive train batteries are contained in the lower deck and are physically and electrically separated from the electronics that are located on the upper deck. Two 12V 18Ah lead-acid batteries in series are provided for the electronics, and two more are present for the motors. Appropriate fuses have been provided for the batteries.

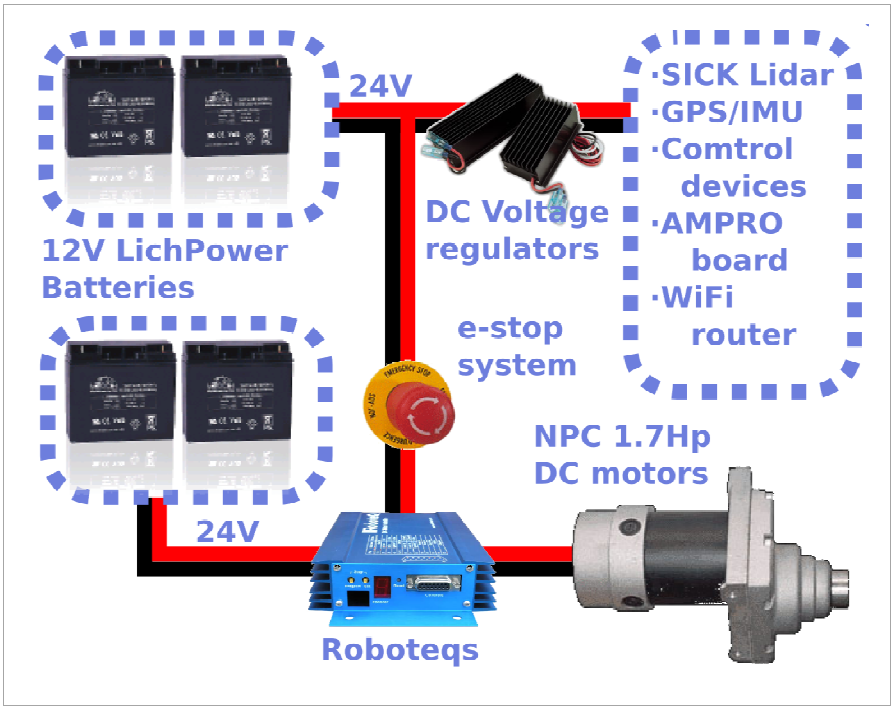

Figure 7 | Power supply diagram

The data connectivity diagram of Figure 8 shows all major robot components. As an innovation, Arduino[1] PICs have been installed as interfaces to all low-level hardware. Each subsystem is thus linked to all others via Ethernet, providing high data rates as well are isolating electrical ground between all devices. The Arduino used for motor control also can read signals from a RC receiver when the RC mode is selected, allowing us to drive the robot safely when autonomous mode is not desired.

[1] Arduino is a physical computing platform based on a simple I/O board and a development environment that uses the Wiring library to simplify writing C/C++ programs that run on the board. Arduino can be used to develop stand-alone interactive objects or can be connected to software running on a computer (e.g., Adobe Flash, Processing, Max/MSP, Pure Data, SuperCollider).

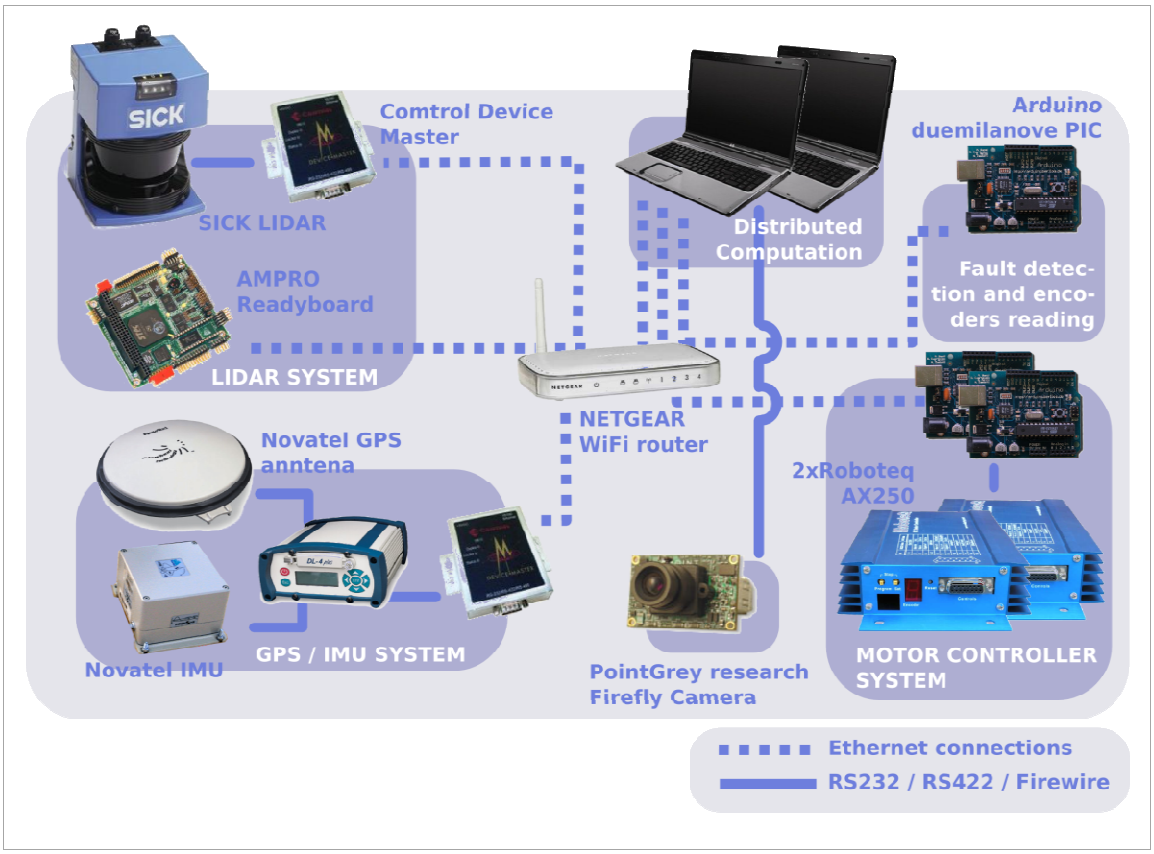

Figure 8 | Connectivity diagram

2.2.2 Sensors

Table 7 briefly describes the main sensors included in the hardware platform:

SENSOR	DESCRIPTION	PICTURE
Camera - Point Grey Research Firefly MV	▪ Resolution 720 x 480 ▪ 60 frames per second	
SICK LMS	▪ Range – 30 meters ▪ Scan rate – 37.5 Hz ▪ 0.5 degree resolution	
NovAtel DL4 plus OEM4 dual frequency GPS receiver	▪ Dual frequency receiver ▪ Position accuracy of 2 centimeters	
Honeywell HG 1700 Military tactical Grade IMU	▪ Ring-laser gyro with laser-calibrated MEMS accelerometer ▪ Drift bias – 10 deg/hr ▪ Acceleration bias – 3 milli-g ▪ Velocity and sampling rate – 600 Hz	
US Digital s2 2048 Optical Encoder	▪ 2048 counts per revolution ▪ 5V supply (from Arduino board)	

Table 7 | Sensors description

Due to time constraints, even the hardware platform was ready to attach the optical encoders, the Arduino board and its code to acquire the sensors data was not ready so it was not used for the competition.

2.3 Safety, Reliability, Durability

The construction of the IGVC robot incorporates proper engineering practices whenever possible to make the platform both safe and reliable. Per the competition requirements, the robot is equipped with a wired and wireless emergency stop (e-stop). In addition to these two e-stops, two Hela switches are in place to as a physical break the batteries powering the motors and the motor amplifier. Other than the wheels, there are no exposed moving parts within the robot, thus preventing injuries while servicing.

Each sensor system (laser scanner, GPS/IMU, encoder and vision) is isolated both physically and electrically. A physical mount is made for each sensor system which can be quickly unbolted from the frame as needed for rapid debugging, and lexan plates isolate each subsystem from through-chassis electrical faults. The only electric connection between each module and the rest of the robot is for power and a single Ethernet. Power for each system is regulated from 24V within the module. The sensors are also powered by a different source than the drivetrain. To avoid human error, there are different sized connectors are each end of the motors and motor amplifier, and between different power architectures.

Careful attention was paid during the design stage to make sure that all systems of the robot provided for safe and reliable functionality. One of the most important safety features is the ability to stop the robot immediately, either remotely or locally. The robot is equipped with a large emergency stop button placed prominently on the back. It is also equipped with a remote that has a line of sight range of 300 feet. These features provide enough flexibility to stop the robot in case of any emergency.

Reliability was a key factor in choosing hardware for the robot. Commercially available components were chosen to perform the required functions on the robot, where possible, to increase expected reliability and durability. Employing such equipment also provides recourse in the event of an equipment failure: most hardware components on the robot can be replaced easily through retail channels.

Chapter 3 | SOFTWARE DESIGN

My main task in the team was taking care of the overall structure of the program as well as developing and improving the algorithms related with the path planner, goal point selection and visualization code. Moreover I created a GUI used to generate test maps so we can simulate different cases to test the algorithms in several conditions.

3.1 Software platform

One of our guiding design principles is that code transparency is critical. To enforce this, a unique and innovative software architecture based completely on MATLAB and Simulink was developed.

Functions written in the native MATLAB language handle almost all of the necessary computations for the robot's path planning, map generation, and occupancy structure. Sensor data streaming and robot motion control are accomplished through Simulink block diagrams, with the help of a program called QuaRC.

QuaRC is a real-time control software toolbox developed by Quanser Inc that directly compiles Simulink diagrams to code that can be executed and monitored in real-time. This type of architecture makes the hardware-software interfaces very streamlined, robust, and reliable.

Further the MATLAB-based software architecture allows for powerful tools like MATLAB's Profiler to be used, where the robot's entire code can be analyzed instantly for computation time and function calls in mere seconds. Figure 9 shows the results of profiling the code used on IGVC 09 and you can notice which functions requires to be improved to quickly increase the speed of the program.

Profile Summary

Generated 14-Jul-2009 14:23:01 using cpu time.

Function Name	Calls	Total Time	**Self Time***	Total Time Plot (dark band = self time)
prepareLocalMaps_Structure	663	9.573 s	9.573 s	
sensor_LidarSim	663	7.778 s	7.726 s	
visual_Structure	221	4.716 s	4.505 s	
map_dialateObstacles	662	3.960 s	3.960 s	
occupancyMapFilter	663	3.608 s	3.608 s	
slowSim_occupancy_Structure	663	8.007 s	3.203 s	
path_CalculatePathMethodDStar_v2	2	1.971 s	1.971 s	
...vigationMode_preparePathPlannerInputs	662	5.491 s	1.531 s	
sensor_Lidar2Occupancy	663	1.197 s	1.197 s	
simulationLoop	1	40.018 s	0.792 s	
path_CalculatePathMethodPfield	17	0.563 s	0.563 s	
slowSim_sensor_Structure	663	8.095 s	0.317 s	

Figure 9 | Profile summary generated by MATLAB

Further, the real-time control capabilities provided by QuaRC allow tuning the control gains in real-time and monitor the performance of the robot as the gains are changed. This enables a quick design-test-verify iteration, speeding up the robot deployment schedule. Figure 10 shows one test run where the control gains for orienting the robot according to a specific yaw command were tuned in real-time.

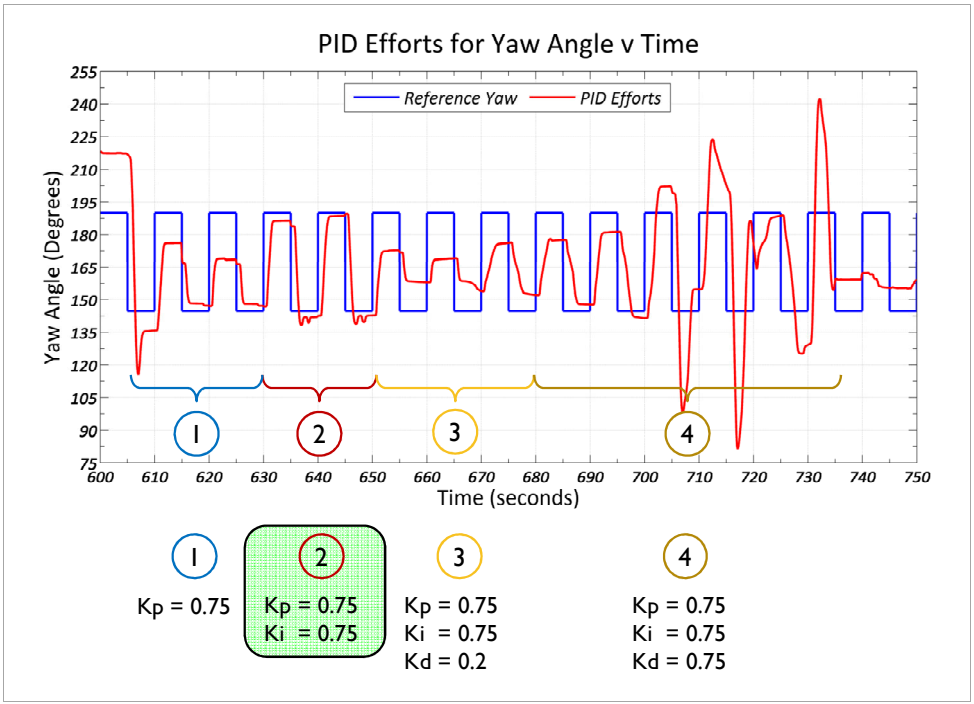

Figure 10 | Profile summary generated by MATLAB

3.2 Program architecture

Developing the program architecture that was going to be used for the IGVC was the first task when this project started on October 2008. First of all some basic definitions were set:

- According to the size of the robot and the different types of obstacles to recognize in the IGVC as well as the minimum free opening of the course configuration we decided 10cm as the smallest partition of the map.
- The program used four types of external data according with the four different sensors (Table 7 chapter 2) with which the robot was equipped: LIDAR to detect obstacles, camera for lines, GPS for position and IMU for orientation. Moreover the code was developed to be able to run if either LIDAR or camera data were missing.
- The occupancy resolution was defined as 10cm so, as shown in Figure 11, if the LIDAR arch interval is 0.5 degrees the maximum LIDAR range to consider is 10m to do not lose resolution (to do not miss any object between 2 consecutive scans); all the LIDAR data beyond this range were dropped.

$$\tan\left(\frac{0.5\cdot\pi}{180}\right) \cong \frac{0.5\cdot\pi}{180} = \frac{0.1}{L} \quad \Rightarrow \quad L = 11.459 \approx 10m$$

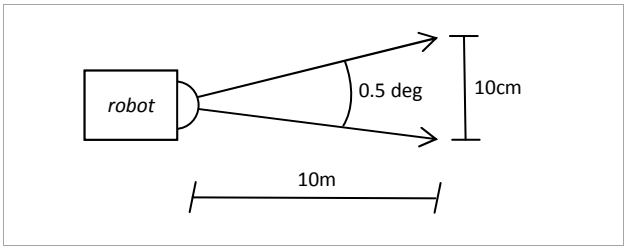

Figure 11 | LIDAR range to consider calculation

- Lines and obstacles (barrels, fences…) where the two types of data that the program used to interact with the environment. Each of them was stored in different maps: lines and obstacles respectively where if there is a -1 means 'not seen' and 1 or 0 for occupied or not. Once the obstacles and lines maps were updated, the information was merged in what we called the "fused map". This map will be used by the path planner algorithm to define a route to the goal point. Special attention was putted on filtering white obstacles appearing in the camera to dose not count as obstacles in the fused map.

Figure 12 | Representation of the three different maps

To sum up, the program managed 3 different types of maps: obstacles (LIDAR), lines (camera) and fused and, at the same time, each of them had two sizes: global and local explained in the next point.

- We defined two sizes of maps to explore the environment: one named global with the same size of the IGVC course (100m by 250m) where each type of sensor data were stored long term and another one smaller named local (30m by 30m – 3 times the LIDAR range) continuously updated to be centered on the robot's position and used by the program to process all the new information acquired and perform the calculus; the smaller size reduces the computational time. Figure 13 shows a diagram with these concepts.

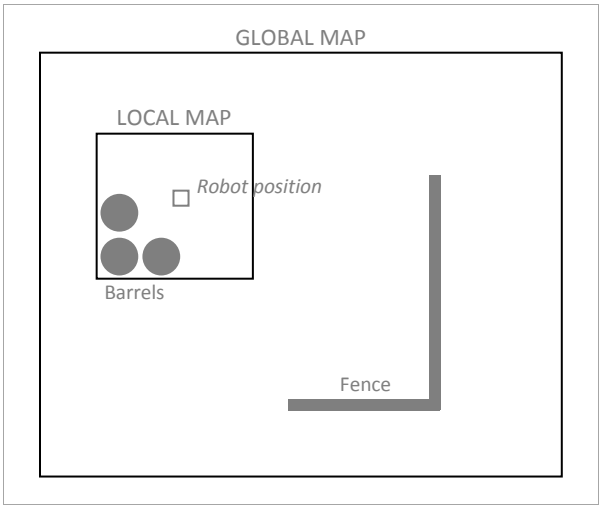

Figure 13 | Diagram of the global and local map

- Objects dilation: Instead of taking into account the robot dimensions when planning the path, the obstacles were dilated so, wherever there is a free point, the program can be sure the robot can go through it. Figure 14 illustrates this process.

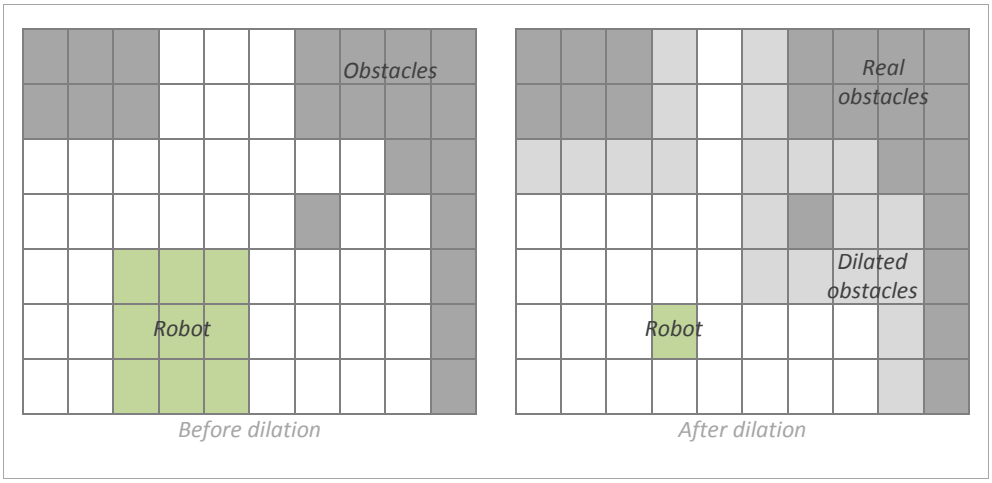

Figure 14 | Dilation process

- Simulation mode: The program should be able to run without being connected to any of the equipment of the robot; this way the algorithms could be easily debugged and tested without having necessarily a robot. Thanks to simulated maps (for lines and for obstacles) the program generates all the external signals needed to run the code: GPS/INS data, LIDAR data and camera data. There were two types of simulation described in Table 8:

TYPE OF SIMULATION	IDEA	WHAT IT DOES?	PURPOSE
FAST SIMULATION	This mode avoids all the sensor data processing.	Revels the 1's and 0's of the simulated maps and put them directly to the occupancy maps.	Avoid the most time consuming functions so you can easily test the rest of the code specially the path planning algorithms.
SLOW SIMULATION	Runs the code going through all the steps as If we were running the robot outside.	Depending on the position and orientation of the robot it looks at the simulated maps and generates a set of data like the senor would do.	Evaluate the whole code before implement it on the robot.

Table 8 | Types of simulation

- Data process: One of the main code differences compared with the last year one is the ability to store the sensor data and use all this previous seen places when you return to a visited place. The data that comes from the LIDAR and camera sensors is processed to generate individual occupancy maps (obstacles or lines) of 1's (objects), 0's (clear spaces) and -1's ('unseen').

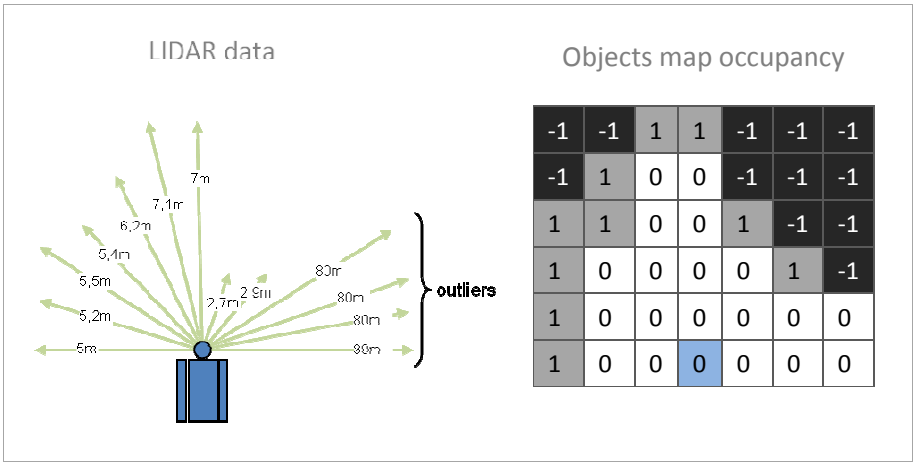

Figure 15 | Mapping LIDAR to occupancy

Using the GPS position of the robot, the occupancy map is merged with its corresponding map (obstacles or lines) using an IIR (Infinite Impulse Response) filter to remove noise. Appling this filter, each value is averaged according to previous readings and a user-defined threshold depending on the speed of the vehicle, level of confidence of the map, and sensor noise.

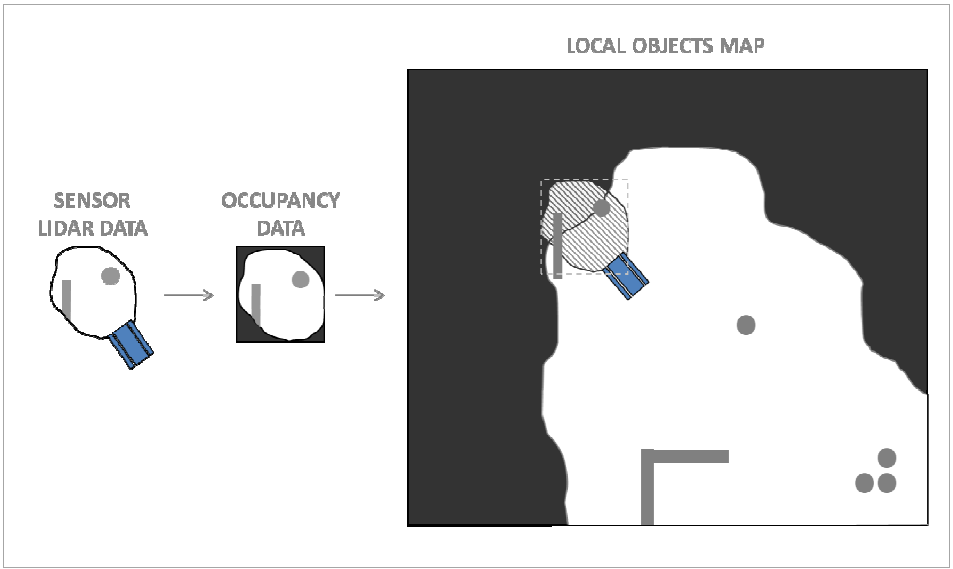

Figure 16 | Converting occupancy map to local map

Once defined all this assumptions, the code was structured in 6 main groups as shown in Figure 17:

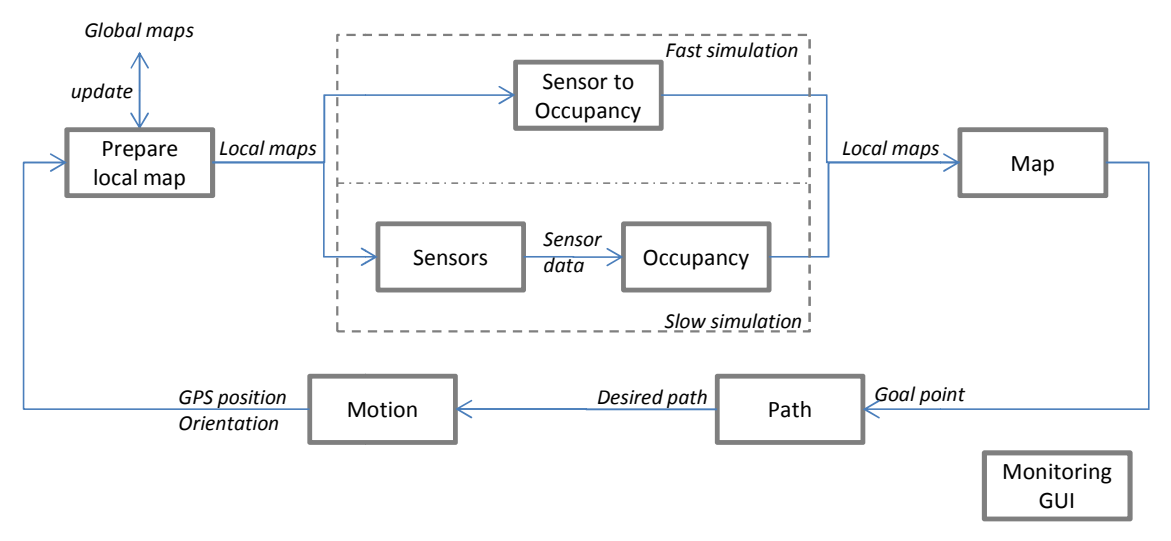

Figure 17 | Software architecture for IGVC

The descriptions of each of these structures are detailed below:

- *Prepare local map*: Updates the 3 global maps (obstacles, lines and fused) with the data acquired by the sensors and pasted on the local maps and extracts 3 new ones (30m by 30m) centered on the robot position according to the GPS coordinates.
- *Fast simulation - Sensors to Occupancy*: As explained before, Sensor to Occupancy is used only on fast simulation and what it does is taking the robot position and orientation and using the simulated maps updates the local maps as if the robot had acquired new data.

- *Slow simulation – Sensors*: In the real mode the robot's sensors will send the data to the computer but to test the program, this function generates a set of it as if the sensors would do. Due to time constraints, only the LIDAR was simulated.
- *Slow simulation – Occupancy*: As shown previously the task of the occupancy structure is pasted all the data acquired in the local maps of objects and lines and merge them to create the fused one.
- *Map*: This structure selects the next point that the robot has to go inside the local map. Depending if we compete in the navigation challenge or the autonomous one the way to decide it is different.
- *Path*: The program manages different algorithms to select the fastest way to get the goal point without hitting any object.
- *Motion*: Give the commands to the robot to move in the desired direction. This code was generated in Simulink and consists of a group of PID controls to command the 4 motors of the robot. In the simulation mode, the GPS signal was generated selecting the next point on the path.
- *Monitoring GUI*: The monitoring GUI is an important tool to debug to code because shows graphically all the data that the program is managing every run: simulated maps, data currently viewed, data previously viewed, visited points by the robot, future path…

Form the structures mentioned before, I programmed the following ones: prepare local maps, fast simulation – sensors to occupancy, map, path and the monitoring GUI. Moreover, as mentioned at the beginning of the chapter, I also developed another GUI to create simulated maps so the program could be tested without having necessarily a robot.

3.3 Prepare local maps

As the Figure 18 shows this structure takes the information of sensor's data mapped into the respectively local maps and save it in the global maps. Once is saved, according to the GPS position, the function pulls out a 30m by 30m map from the global map that will be used for the other structures of the program.

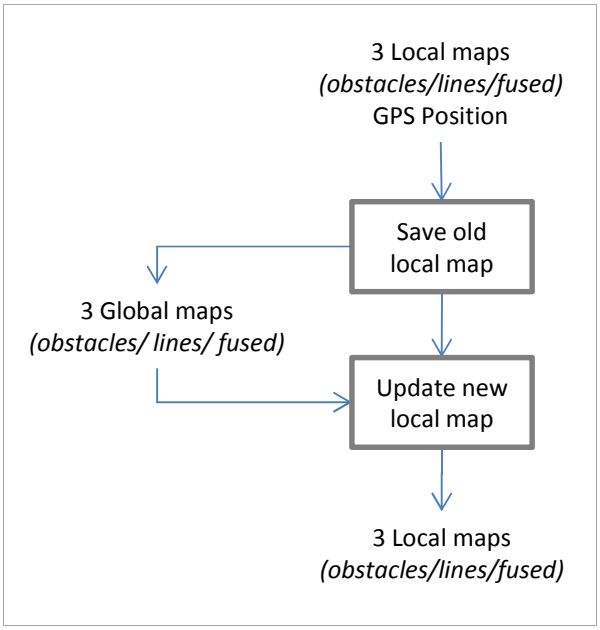

Figure 18 | Prepare local maps structure diagram

The fact of handling a smaller size matrix helps to reduce the computational time for operations such as path planning and goal point selection which will be explained later. The size of the matrix, 30m by 30m, was decided according to the LIDAR range (10m) and the largest obstacles (fences) that the robot will find in the course.

Figure 19 shows the local map around the robot position in different times among the path while the robot is exploring the course to go to the desired GPS position.

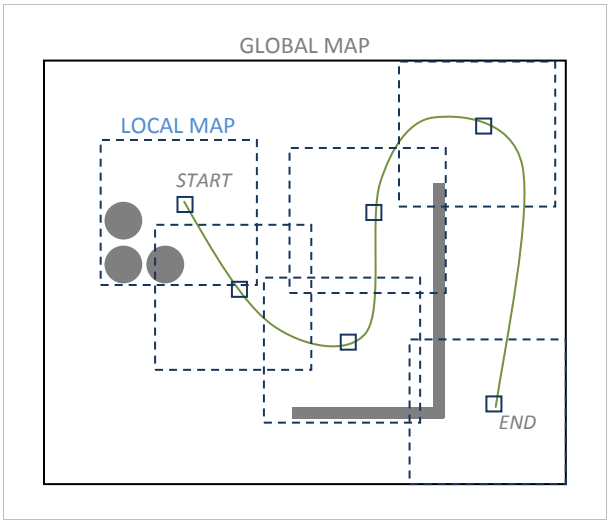

Figure 19 | Local map representation among the course

3.4 Fast simulation – Sensor to occupancy

This structure performs the operations of sensor and occupancy structure of the slow simulation all at once. It was developed to avoid one of the most time consuming parts of the program, the sensor data process.

The inputs and outputs where the same as the slow simulation (sensor and occupancy structures) but instead of generating the sensor data and then process it, the code used the GPS position and the orientation to define the 'seen' area of the sensors and revel the 1's and 0's of the corresponding zone of the simulated maps to put them directly to the local maps (objects for LIDAR and lines for camera). Finally both maps are merged into the fused map.

Figure 20 shows the diagram for this process; note that thanks to a couple of flags defined at the beginning of the code, the program can run just processing one type of data, camera or LIDAR. This is especially useful on the navigation challenge where the course does not have any line so the camera data process could be omitted.

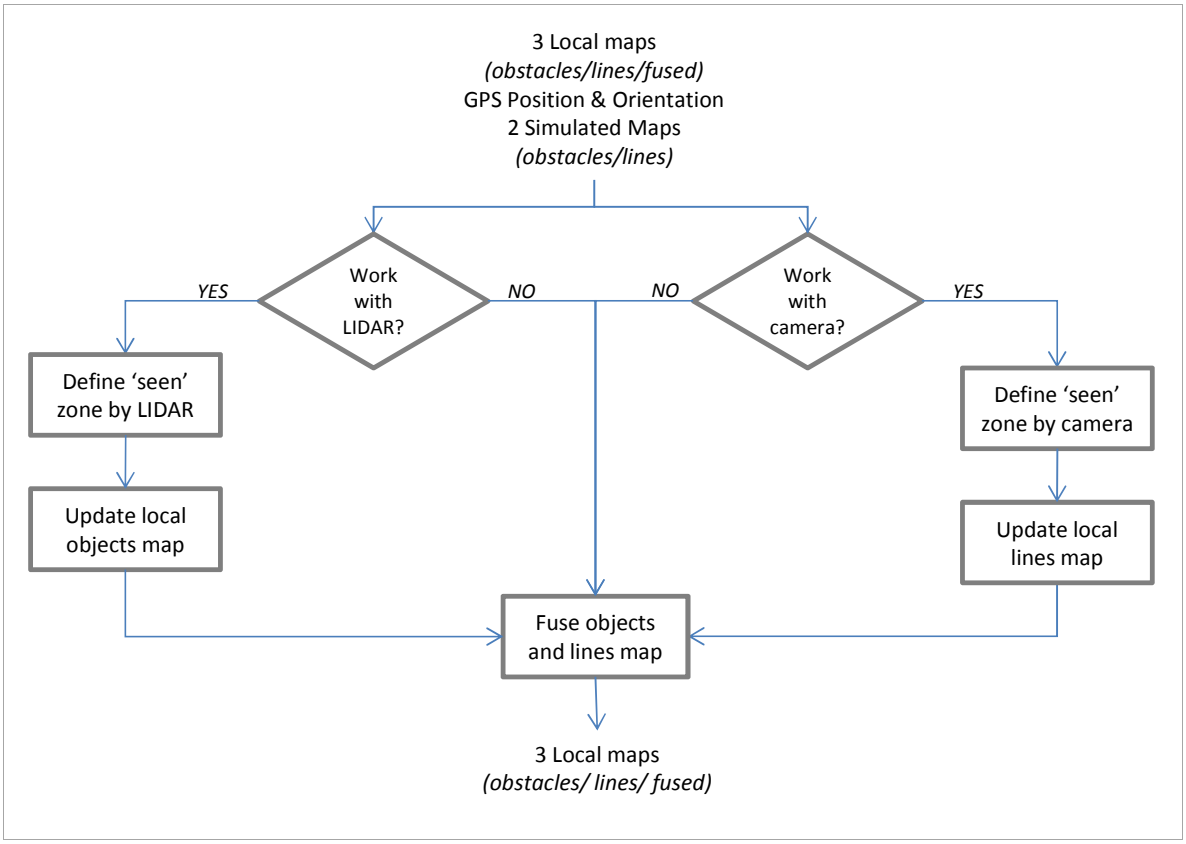

Figure 20 | Fast simulation – Sensor to occupancy structure diagram

The biggest advantage of the fast simulation is that the faster execution of the program enables a faster debugging of other parts of the code.

3.5 Slow simulation – Sensors

Figure 21 shows the initial diagram that we planned to use for this structure where all the data that came from the sensors was filtered and processed to determine the level of thrust of each measure. The process was divided into 4 steps:

- The data that came from the sensor was named as raw: so for the LIDAR, GPS/INS and encoders we had raw LIDAR, raw GPS/INS and raw odometry. About the camera, the data had to be pre-processed before to fix the perspective of the image to convert it into a bird's eye view (2D world) and extract the white parts from the RGB image.
- *Step 1 – Filter*: fix outliers, jitter, noise and bad readings of the data. In this step we use the previous data readings already filtered to determine if there is a big change between measures and if this happens we will use the previous data instead of the new one.
- *Step 2 – Predict and project*: Once the data is filtered, and knowing the values of the previous readings, calculate how the robot has moved for each type of data: Δθ, Δx, Δy (change on the orientation and the 2 dimension movement).
- *Step 3 – Fuse*: Compare the four predictions to get the best assumption of Δθ, Δx, Δy.
- *Step 4 – Determine trust*: Using the best Δθ, Δx, Δy, check each sensor data to determine the trust level of the camera and LIDAR reading and to get the best assumption for the GPS/INS.

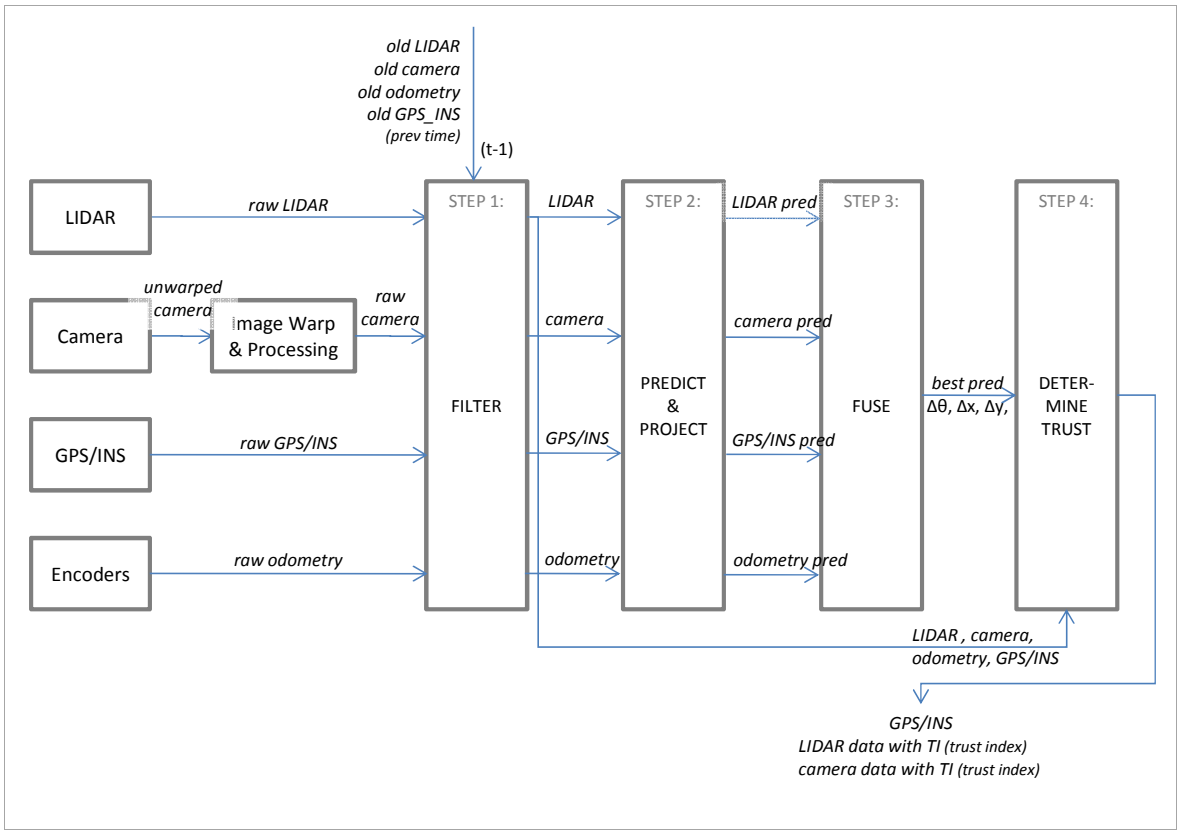

Figure 21 | Slow simulation – Sensor structure diagram

Knowing how the robot has moved enables to predict how the obstacles and lines should have moved so how the data should look like.

This way of filtering the data is especially useful in certain cases like, for example, having a white obstacle that the camera detects as a line. As the robot moves, the bird's eye view of that obstacle considered line changes its place, revealing that that is not a line.

Another useful case is to filter weather conditions like very shiny and sunny days where the sun reflection on the grass shows a white spot in front of the camera. As the robot moves, this white spot always appears in the same place of the image denoting that is neither a line.

By the time of the competition, the code for this structure was not completed and the four steps previously described to filter the data were not included into the final program. Moreover the platform did not include encoders so we didn't have odometry data which would have been useful when we had to work inside a building (no GPS signal) and to perform speed PID controls on the robot.

Finally this structure was simplified to fix the outfitters of the LIDAR data and do the image processing which included: unwrapping the image and detect the white lines in the RGB image. This part, the image processing, was one of the most difficult ones to code and the one that take most of the time to compute and is explained in more detail in the next point.

Mention that due to the simplification of the sensor structure the only data filtering process included in the final program was performed in the occupancy structure.

3.5.1 Vision and image processing

One of the major problems faced during 2008 competition was the inability to adapt to changing lighting conditions such extreme sun versus overcast conditions. The previous algorithm was a simple intensity-based static thresholding algorithm and did not perform well on the course. Consequently, the main focus of the image processing effort was on introducing adaptability into the algorithm.

Various new algorithms were coded and to test them a GUI was created for manually tagging images to represent the ground truth. Comparing both images, the tagged and the processed one, we were able to set a percentage of accuracy for each algorithm. Figure 22, Figure 23 and Figure 24 show the tagging process using the MATLAB GUI:

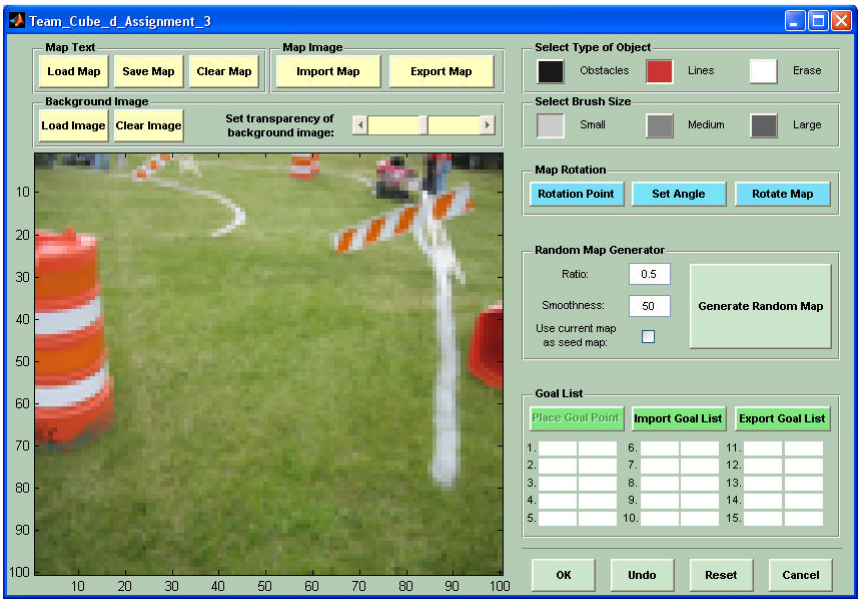

Figure 22 | Image loaded into the GUI

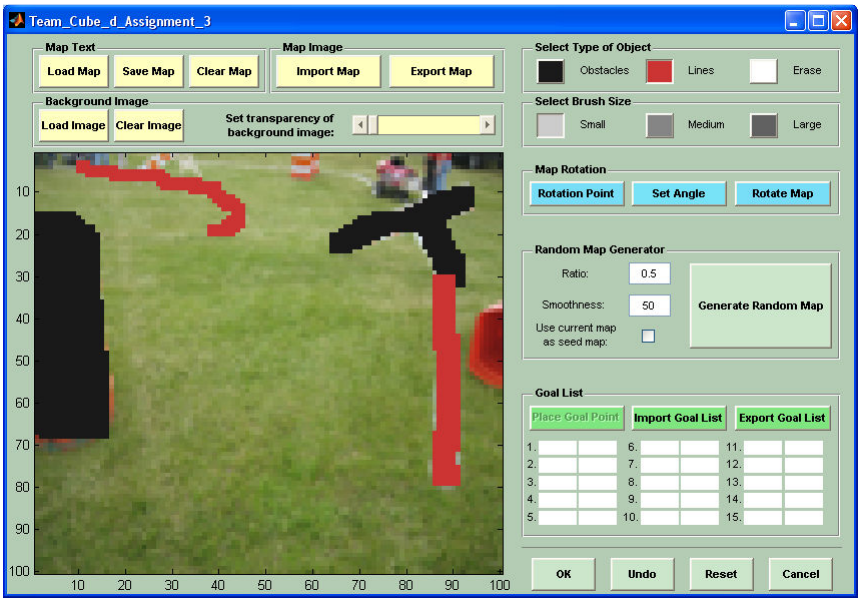

Figure 23 | Lines and obstacles manually tagged in the GUI

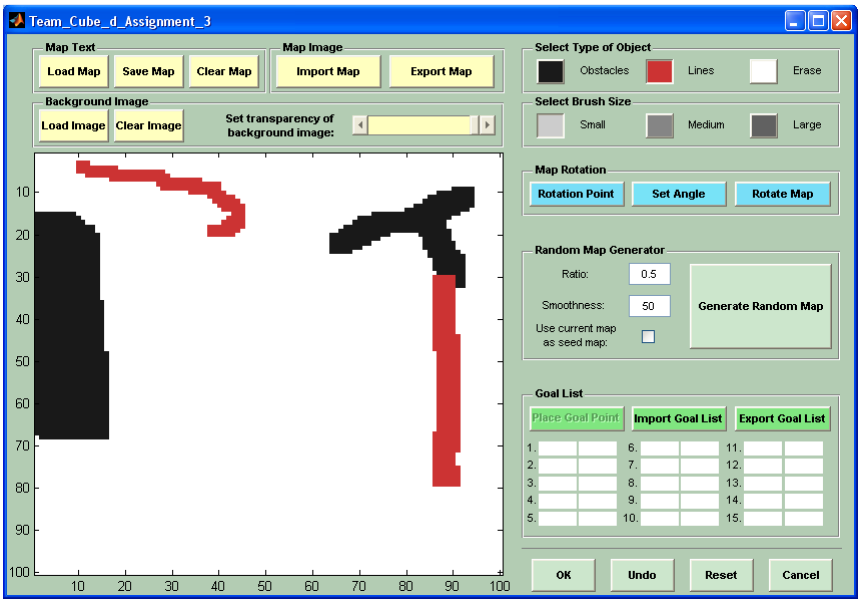

Figure 24 | Final image tagged

Note that in the field the robot will find white obstacles that can confuse the algorithms and process a line where there is no one; this is way we also tagged the white obstacles with a different label on every image.

For IGVC 2009, three different algorithms with varying degrees of adaptability were considered apart from the one used on 2008 that was discarded very soon because failed on many occasions:

- A simple intensity-based algorithm on the RGB color space. Similar to last year's algorithm, except that that thresholds are intensity-scheduled, i.e. change with image intensity.
- An adaptive threshold algorithm based on the R^2GI color space. By trial and error on images saved at IGVC 2008, it was found that the R^2GI color space yields a good color space basis. A new threshold is calculated for each image using averaging, making this algorithm adaptive.
- A fully adaptive algorithm based on use of Principal Component Analysis (PCA) and K-means clustering. The PCA delivers the optimal color space (the principal component) that best partitions grass and non-grass pixels. We then use K-means to quickly calculate thresholds.

The three algorithms were compared against each other for speed and accuracy. It was found that PCA with K-means clustering performs the best in terms of accuracy, and does so at an acceptable speed.

Principal Component Analysis

As lighting and weather conditions change, the color space that helps best distinguish between white lines and grass also changes. Such a color space can be determined on-the-fly, by using only the principal component of the image, i.e. the eigenvector pointing in the direction of maximum change in the image information.

As can be seen from Figure 25, the image data appears scattered. However, by proper orientation, it can be seen that most of the data is tightly clustered along the principal component axis pixels (Figure 26).

The advantages of performing PCA include reduced processing time (by reduction of data dimensionality), adaptability with changing lighting conditions, and the option of extension to higher dimensions (using kernel functions) for higher accuracy

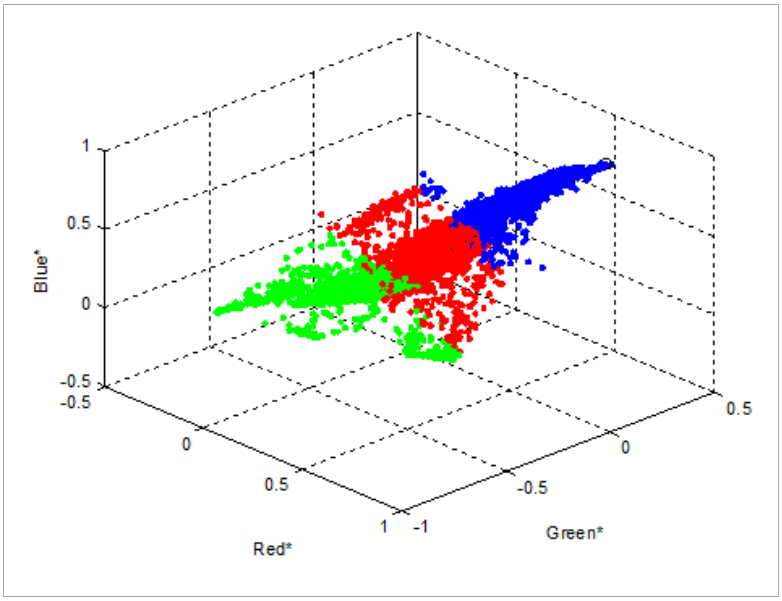

Figure 25 | The image data as seen orthogonal to the principal eigenvector (PE). This direction provides the largest separation of line and grass pixels.

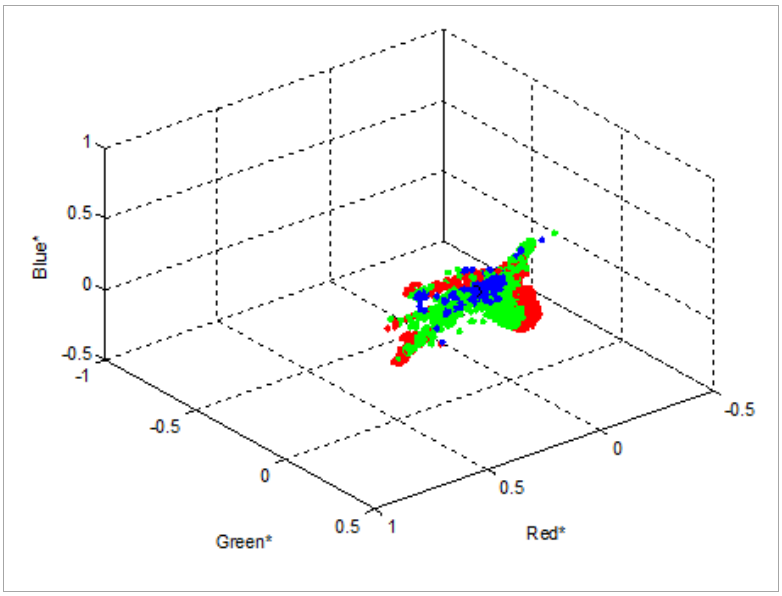

Figure 26 | The image data as seen along the direction of the PE (circle). This view provides minimum separation of line and grass pixels.

K-means clustering

K-means clustering is performed on 1-D data to separate lines from grass. K-means minimizes the sum of distances across all points by putting them into appropriate clusters. Figure 27 shows the clustering for grass and lines.

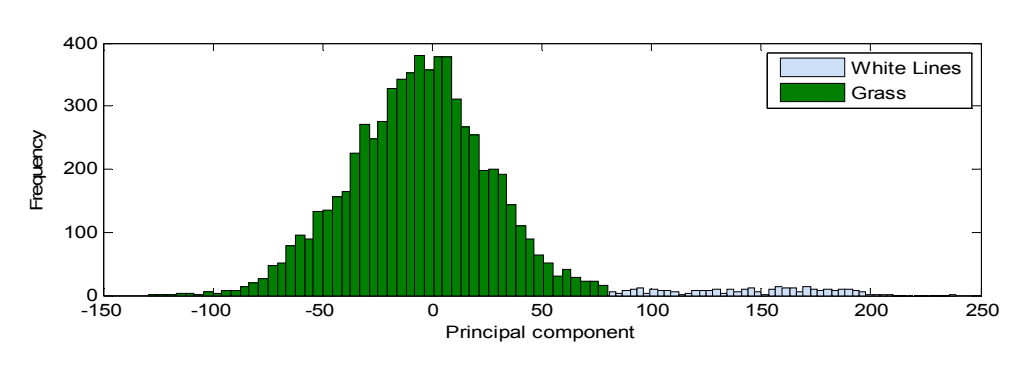

Figure 27 | Clustering on the one-dimensional principal component

After the data points have been clustered in one dimension, the original image is reconstituted and a projective transformation is performed on the image to obtain a birds-eye view. Figure 28, Figure 29 and Figure 30 show the process:

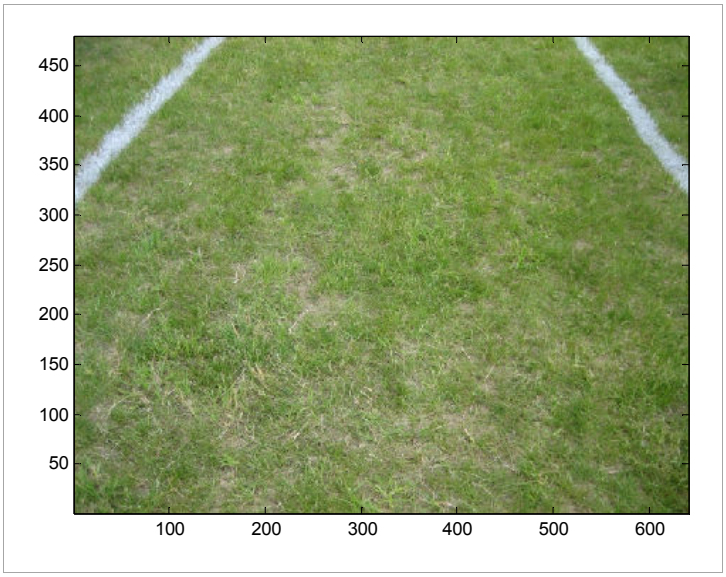

Figure 28 | Original image

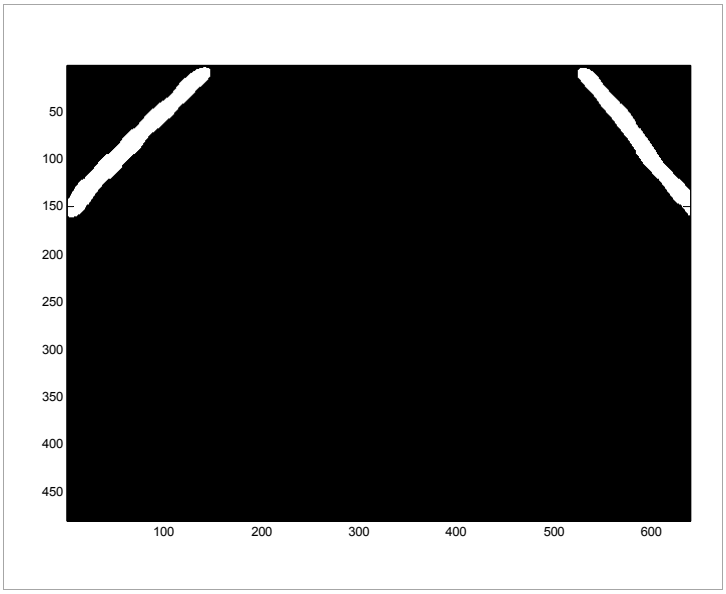

Figure 29 | Processed image with extracted lines

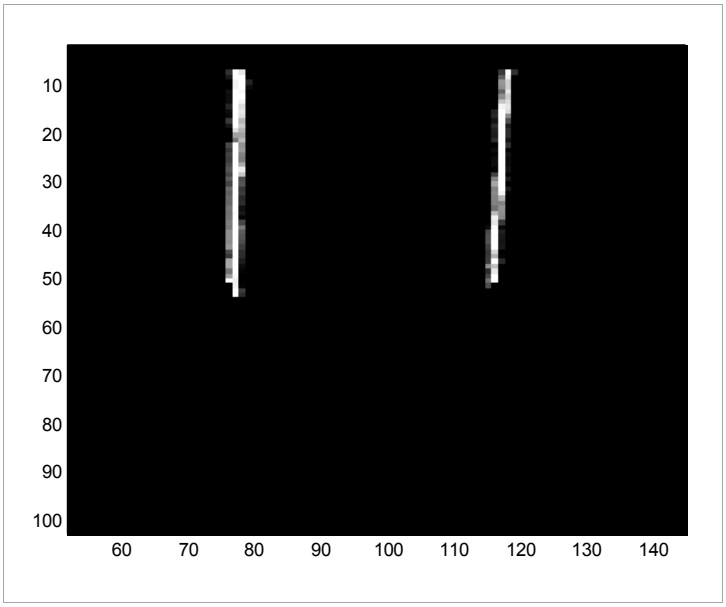

Figure 30 | Processed image with extracted lines

The resultant image processing algorithm coded showed that was adaptive, and that can modify the color space and thresholds on-the-fly, based on the lighting conditions and weather.

3.6 Slow simulation – Occupancy

The objective of the occupancy structure is to map the data that comes from the LIDAR and the camera into a 2D matrix of 1's (for occupied), 0's (for clear) and -1's (for unseen). The size of the local map is a 300-by-300 matrix corresponding to a 30-by-30 meters in the real world with a resolution of 0.1 meters.

Figure 31 shows the diagram process for this structure. Also Figure 15 and Figure 16 under program architecture section explain part of this process:

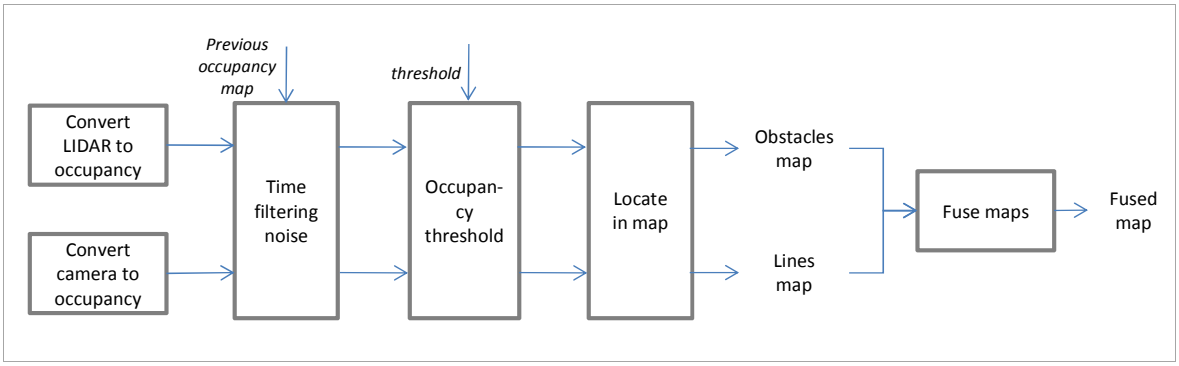

Figure 31 | Slow simulation – Occupancy structure diagram

As mentioned in the previous point, the sensor structure did not perform any kind of filtering process so all the responsibility fails on this section. As described on the previous diagram the first step is processing the LIDAR and camera information to generate individual occupancy maps.

Once the occupancy maps are made, they were merged with its corresponding local maps thanks to the GPS position and the orientation values provided by the GPS/INS using a IIR (Infinite Impulse Response) filter.

The main idea of this filter is averaging the values according to the number of times that we have got a certain reading. So every time we detect that a certain cell is occupied the possibility of having a 1 there increases as well as when we had a clear space the chances gets reduced.

It is as simple as keep adding 1's when its occupied and subtracting 1's when it's free. Then, applying a simple threshold based on the speed of the vehicle, level of confidence of the map and sensor noise the program decides whenever that cell is occupied (1) or clear (0) before pasting the occupancy map onto the local map. The results are similar to a spatial Kalman filter.

Another important function in this structure is the fusion of the obstacles and lines map to obtain the fused map. This map will be used by the path planner algorithm to define a route to the goal point.

Table 9 describes the criteria used on the fusion process:

Obstacles Map	+	Line Map	Fused Map
0	+	0	0
0	+	1	1
1	+	0	1
1	+	1	1
-1	+	0	-1
-1	+	1	-1

Table 9 | True table to fuse obstacles map and lines map

Notice two criteria on this true table that was used to merge both maps: first of all the LIDAR range is much bigger than the camera one so we will not find a combination of a 1 or a 0 of the obstacle map with a -1 on the lines map.

On second place whenever there is a -1 on the obstacle map (an 'unseen' zone) we discard the lines map information. This was done to filter the white obstacles placed on the course like barrels or fences that the camera recognizes as lines. Figure 32 show some of these cases:

Figure 32 | White obstacles placed around the course

Due to the transformation from a perspective view to bird's eye one done to the images captured by the camera the white obstacles are interpreted like white lines painted on the grass in the lines map. Figure 33 shows an example of how would look like the lines map after processing a white barrel:

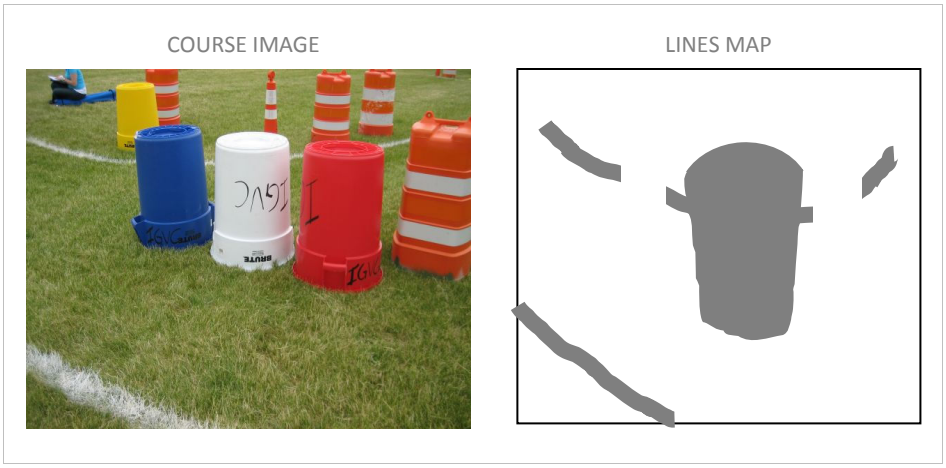

Figure 33 | Lines map after processing a white barrel

An easy way to detect these cases is omitting the lines map information whenever the LIDAR scan has not reached that zone in the course (obstacles map has a -1 on that cells). Another way to describe it is that to discriminate if a white on the picture is a line or not, the obstacles map always should show a 0 on that cell.

This kind of filtering to get a accurate fuse map (with the right occupied/free spaces) was very important due to its relevance on the program; as mentioned before it was used by the path planner algorithm so if we do not filter that type of data we could have blocked possible ways to get to the goal point as shown in Figure 34:

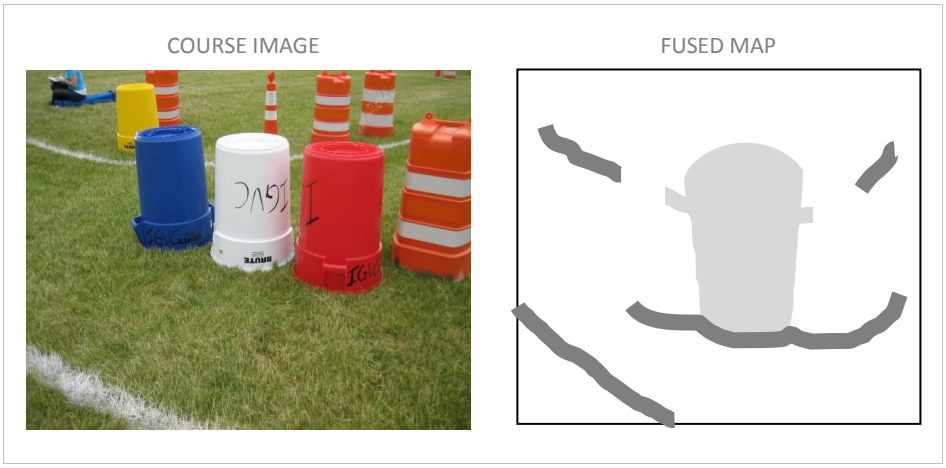

Figure 34 | Fused map after filtering a white barrel

3.7 Map

The idea of the map structure is to prepare the inputs needed for the path planner algorithm. The main tasks are: dilating the fused map and defining a goal point.

Due to the differences between the navigation and the autonomous challenge the way the goal point is defined on each case differ. On the navigation the objective is to get to several GPS point spread around the course and on the autonomous the aim is to discover as much terrain as possible without going out from the lines and hitting any obstacle. This is why we created two different structures depending on which challenge the robot competed.

3.7.1 Navigation challenge

For the navigation challenge the judges provide a list of GPS points that the robot has to get in any order with the minimum time.

The first step is to decide the order in which the given waypoints should be traversed. This problem is essentially the celebrated Traveling Salesman Problem. To solve it three techniques that were analyzed, coded and tested: a heuristic technique (polar sorting around centroid), a modified Monte-Carlo method and a Genetic Algorithm.

The Heuristic Polar Sorting technique was found to be faster than any other algorithm by an order of magnitude, while giving near-optimal routes for small number of waypoints. The technique is explained pictorially in Figure 35.

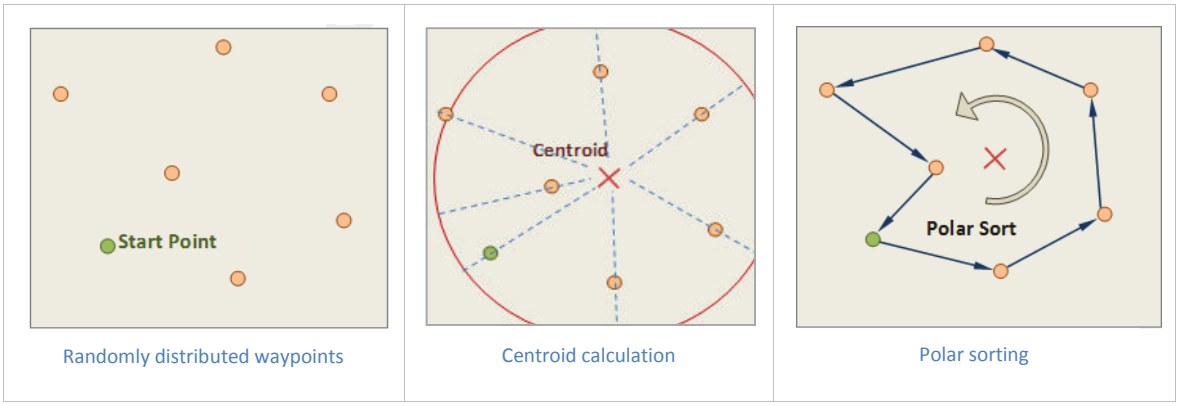

Figure 35 | Heuristic Polar Sorting Algorithm yields optimal route for small number of waypoints

This operation was done on the initialize code so it was only performed once and therefore not included on the structure diagram of this section.

Figure 36 shows the tasks that this structure performs when the robot competes on the navigation challenge:

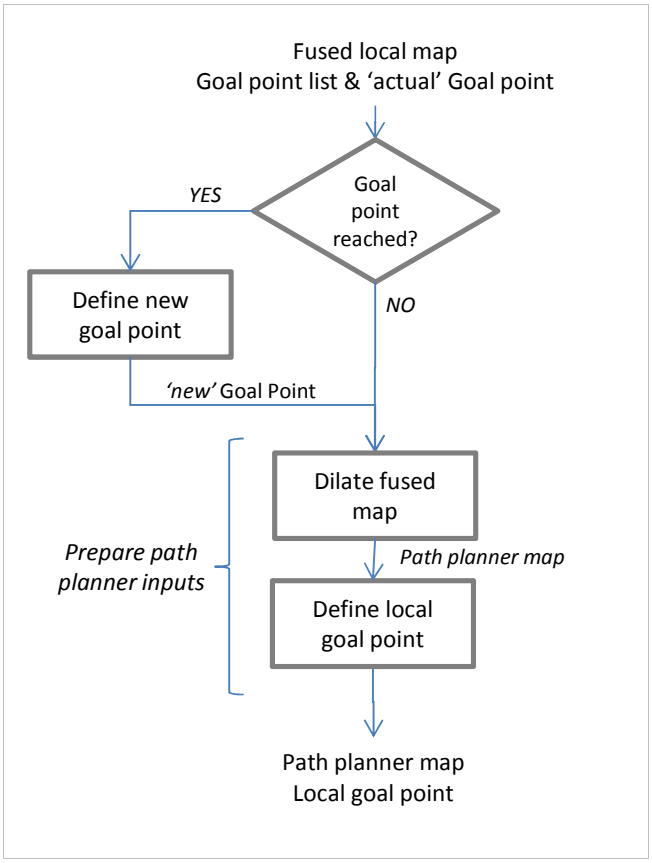

Figure 36 | Map structure – Navigation challenge diagram

The first function determines if the robot was close enough to the 'actual' goal point to assume that it has reached it. The second task is to dilate the fused map a certain number of cells according to the size of the robot so whenever the path planner sees a clear cell on the map can be sure that the robot can fit throw it.

Finally due to the use of a local map to plan a path, usually the desired goal point is placed out of the local map. In order to the path planner algorithm can work in this situation, the goal point has to be projected on the edges local map taking into account that it cannot be above an obstacle. Figure 38 shows the process graphically:

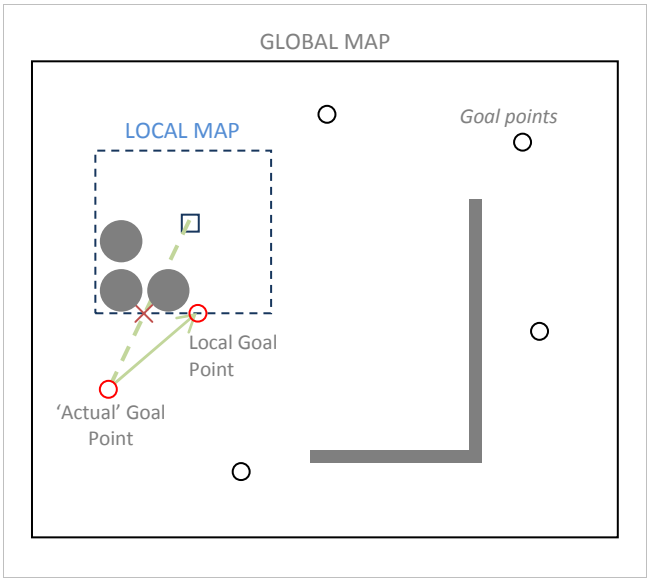

Figure 37 | Goal point projection on the edge of the local map

3.7.2 Autonomous challenge

As shown in the diagram of the Figure 38, the result of this group of functions is the same as the navigation challenge: a path planner map and a local goal point.

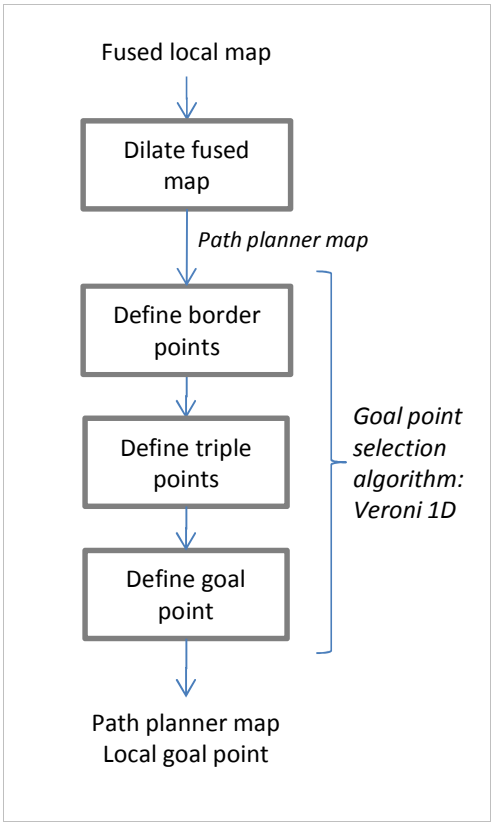

Figure 38 | Map structure – Autonomous challenge diagram

The way to obtain the path planner map is the same as the navigation challenge: dilating the fused map but, on the other hand the method to define the local goal point is significantly different due to the characteristic of this challenge.

The autonomous challenge requires that the robot explore an unknown environment. Thus, there is difficulty in deciding which location to move to next. The classical approach is wall following, but our simulations of this algorithm showed it will fail in common circumstances.

After trying different methods to decide the next location, or local 'goal point', we then investigated seeking points that lie at the intersections of unknown, open and obstacle spaces, areas which we named "triple points". There are only a few triple points in any occupancy map, thus monitoring these points is quite fast. Simulations showed that wall-following is a sub-class of algorithms that seek such triple-points.

For the competition, another type of triple-point algorithm was designed that was motivated by 2D Veroni diagrams, and which we refer to as Veroni 1D. Essentially, the idea of this algorithm seeks to move to the middle of the largest open, unexplored area between triple points on the 1D boundary. The computation process is explained in Figure 39, and is not only robust, but exceptionally fast to compute.

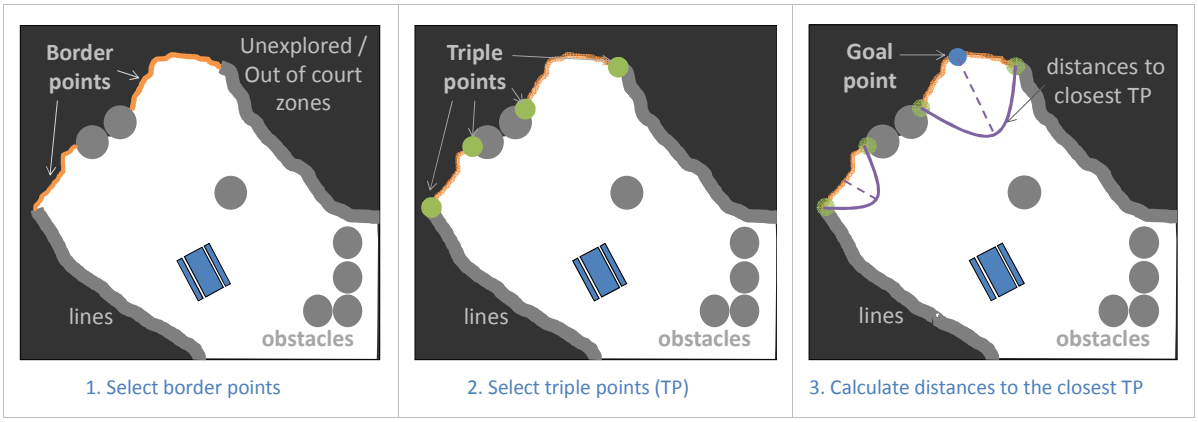

Figure 39 | Goal point generator algorithm for the Autonomous Challenge

The algorithm realizes the following steps:

- Find the border points on the map (orange lines) which are the frontiers between the unexplored zones (-1's) and the clear areas (0's).
- Calculate the triple points presented on the dilated map (green dots): clear cells (0's) that are in contact with obstacles (1's) and unexplored zones (-1's).

- Once it has both elements, calculates the distance of every single cell of the border points to the triple point that is closest to it. The purple line is a graphical representation of these distances.
- Finally the goal point is the border point that has the largest distance to the closest TP and that corresponds to the biggest opening of the map.

3.8 Path

The objective of this structure is to create a path from the current position of the robot to the local goal point sorting the different obstacles mapped on the path planner map. From the experience on previous IGVC competitions, we assumed that any robot traversing the course should be able to plan a path under conditions shown in Figure 40.

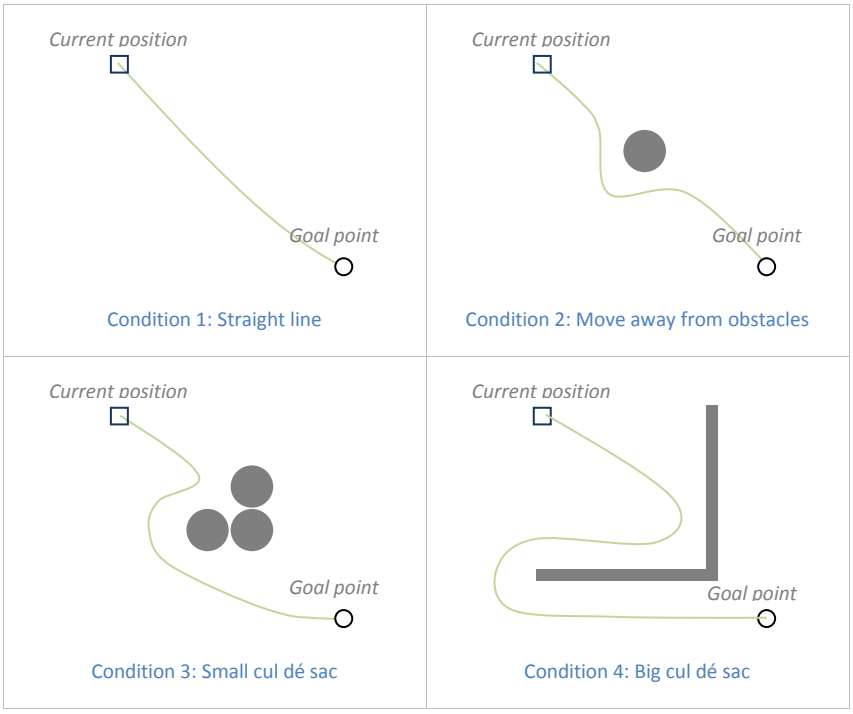

Figure 40 | The four typical conditions on the IGVC course

Knowing the multiple conditions the robot will face on the course, several algorithms were tested thanks to the simulation tool as well as different ways to code them to increase its speed. These are the commonly used path-planning algorithms considered:

- *Straight line*: As its name indicates, this algorithm tries to get the goal point on a straight path. This method is very useful when the robot is moving on an open, free space due to its simplicity that makes it fast to be calculated.

- *Potential Field:* This method assimilates the robot as a negative particle and the goal point as a positive one. Moreover, every object on the map has a negative charge that repels the robot on its way to the goal point. The optimal route to get the goal point is the one that requires less energy to achieve it.

  Another way to describe it is assimilating the robot as a marble that falls down to the goal point, the lowest potential, and every object that finds on its way represented like mountains, high potentials, that repels the ball away from them. Figure 41 shows this concept:

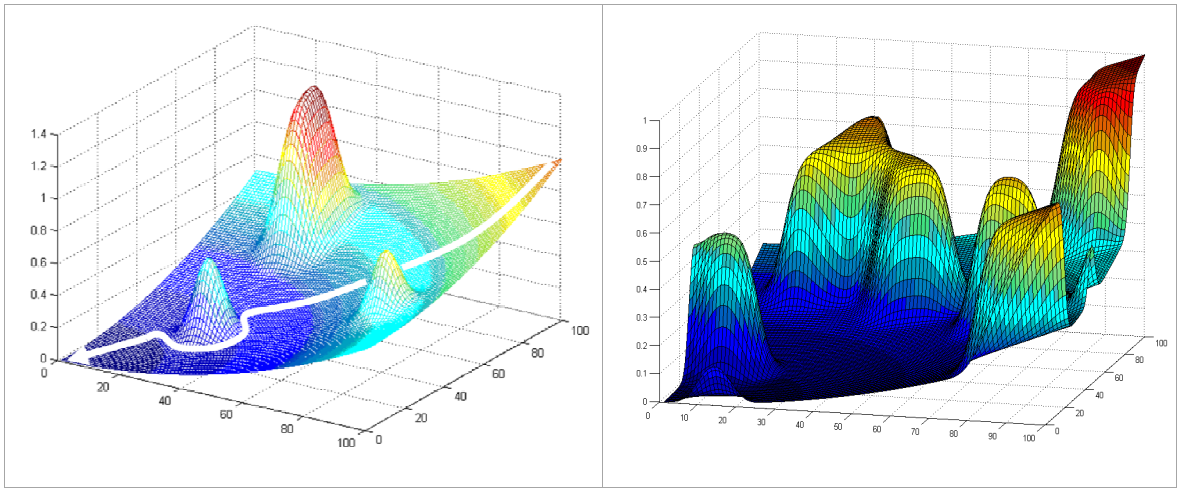

Figure 41 | Potential field map representation

The main problem about this path planner is when the robot reaches a *cul-dé-sac:* the cells around it have bigger potentials so stops falling down although it has not reached the goal point. Figure 42 shows a typical example where the path planner would not succeed:

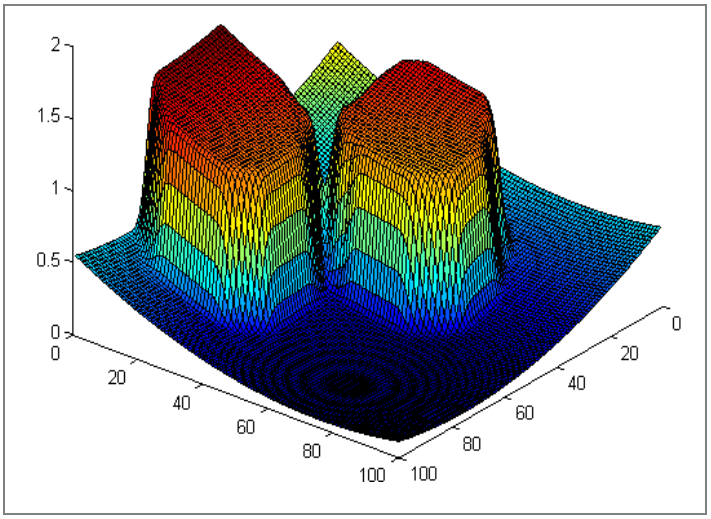

Figure 42 | The robot get trapped on a *cul-dé-sac*

- A star limited: This path planner always finds the goal point. To calculate the path, it uses a heuristic function that assigns to every visited cell and its neighbors a cost resultant to add the cells travelled and the estimated distance to the goal point properly weighted. Figure 43 shows the calculation process on a graphical example:

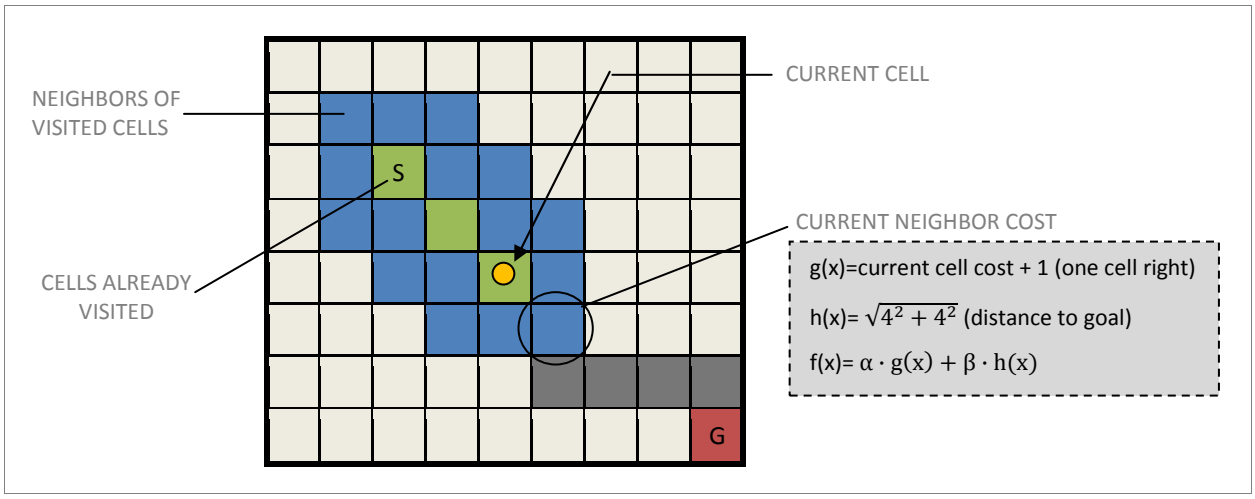

Figure 43 | A star calculation representation

This method is slower than the potential field and it may waste a lot of time to overcome big *cul-dé-sacs*, this is why it was limited to a certain number of iterations. If it passes this limit, the function stops its execution and jumps to the next path planner method.

- D star: This method is pretty similar to the previous one, it assigns to every cell a certain cost and the optimal route is the one that takes less cost but instead of going from the start point to the goal one, it goes reverse. This path planner is usually slower when the robot is moving on an open space or with a reduced number of obstacles but it is very useful when the environment has reduced free spaces and tricky configurations like big *cul-dé-sacs*.

A summary of the performance of the different path-planning algorithms under the conditions shown in Figure 40 is provided in Table 10. There, under processing speed, "N" indicates the distance between the goal and start point, whereas "t" indicates calculations that must be performed at each time interval:

Algorithm	Condition				Processing Speed
	1	2	3	4	
Straight Line	✓	fails	fails	fails	O(N)
Potential Field	✓	✓	fails	fails	$O(N^2) + O(N) \cdot t$
A* Limited	✓	✓	✓	fails	$O(N^2) \cdot t$
D*	✓	✓	✓	✓	$O(N^2) \cdot t$

Table 10 | Capabilities of path planners

As shown in the table, there is clearly a tradeoff between the speed of the algorithm, and the complexity of the planned path. Figure 44 shows the time consumed by these algorithms to sort a basic example of an obstacles map of 10-by-10 meters (100-by-100 cells):

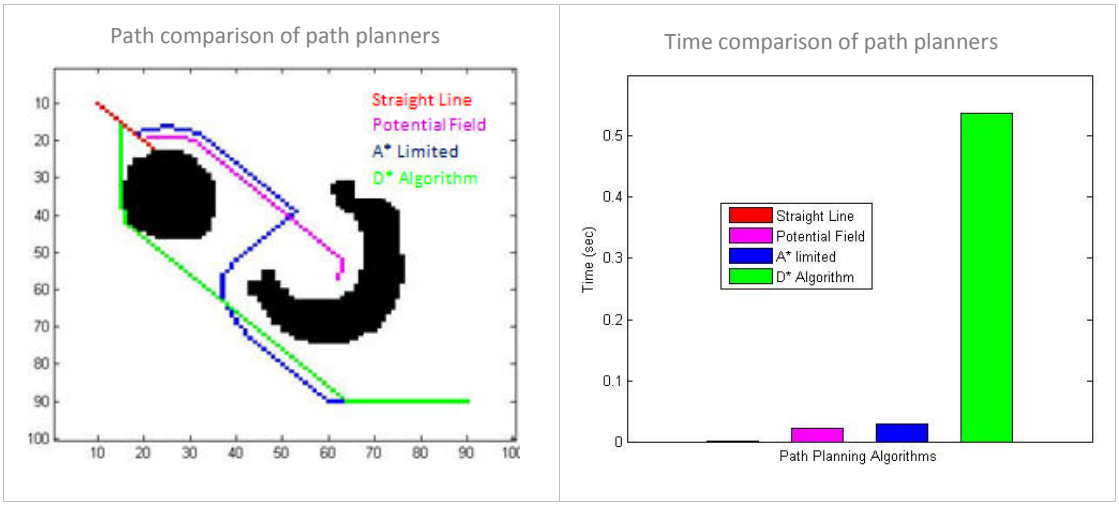

Figure 44 | Comparison of path planner algorithms

It is worth mentioning that on one hand, the algorithms straight line and potential field failed to get the goal point so, although the time consumed is very small, their simplicity makes them unable to succeed in this scenario.

On the other hand, there is the A star limited method that even it succeeded in this case it is not completely reliable as shown in Table 10. Finally the D star proved to be the only one capable to get the goal point in any situation but consuming a lot of resources and decreasing the speed of the overall program, a key aspect to explore an environment in real time.

It is clear enough that there is not a perfect algorithm for all situations that combined sufficient robustness with an acceptable speed to run the code so we decided to use all of them according to the situation.

The path planner switches between all four algorithms to exploit their benefits and still minimize processing time. To achieve this type of dynamic scheduling, the simplest path planners are attempted first, and each algorithm self-monitors progress. In the case that a planner is unsuccessful, a supervisory algorithm switches to a more complex path planning approach. The diagram shown in Figure 46 explains this process.

Another key aspect is how often the path needs to be recalculated. On the IGVC contest, the judges have the chance to move the obstacles so the route to get the goal point may change: an opening on the fence might appear, some barrels could be moved to habilitate a shorter route...

A first approach was to calculate the path at the maximum speed the program could run: approximately 40/50 Hz. But thanks to the simulation tool, we realized we were wasting

resources calculating the path at that speed: while the robot moved a few centimeters, the path was recalculated 8 or 10 times.

To solve this, we simplified the path according to the following strategy: we established a confident radius (about 1 or 2 meters around the robot) where the program attempts to get the furthest point on the path on a straight line without crossing any obstacle. We named that point, intermediate goal point and the straight path to get it: intermediate path. This process is explained on Figure 45.

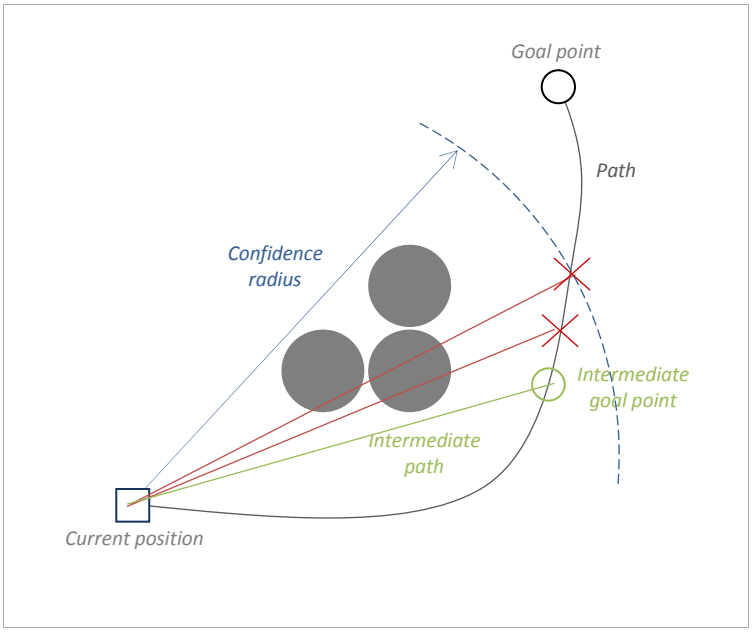

Figure 45 | Intermediate goal point definition

The path is not recalculated until the intermediate goal point is reached or on obstacle appears on the intermediate path. Thanks to this simplifying process we optimized the number of times the path planning was called; the program ran at 20/30 Hz enough for the robot speed.

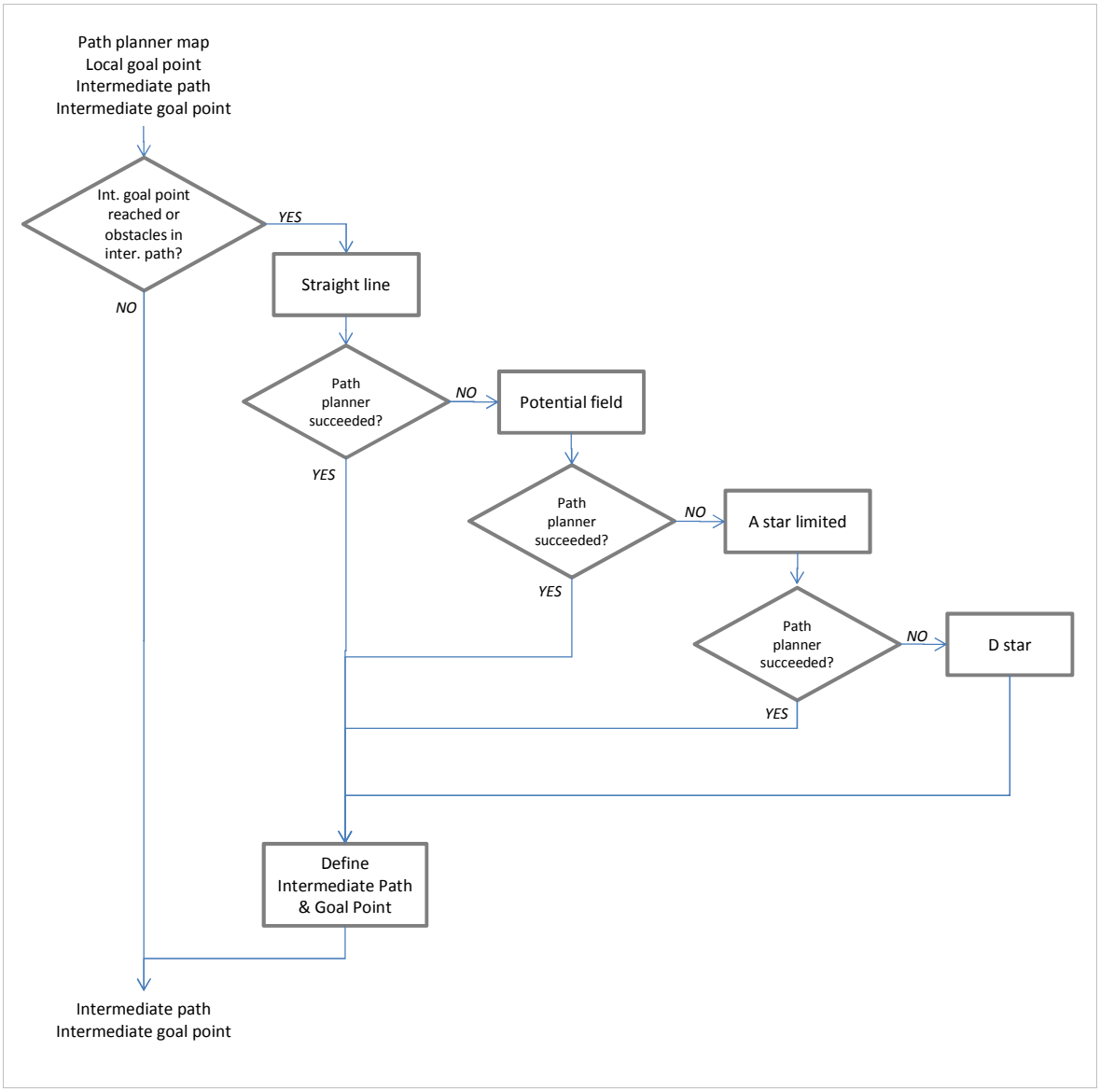

Figure 46 | Path structure diagram

3.9 Motion

Once the path is calculated and simplified into the intermediate goal point, the motion structure is the one in charge to give the commands to the motors to get that point. The code is implemented on Simulink and interacts in real time with the GPS/INS data and the four DC motors thanks to QuaRC using TCP/IP.

There are two basic instructions used to command the robot: turning and speed. As the name indicates, the turning commands are used to face the robot on the right direction and the speed ones to move to it.

On one hand, the diagram, knowing our actual GPS position and the GPS coordinate of the intermediate goal point, estimate which should be the yaw to get that point on a straight line. Then the turning commands are calculated using a PID control previously tuned and this yaw reference. Figure 47 shows the Simulink diagram for the turning commands:

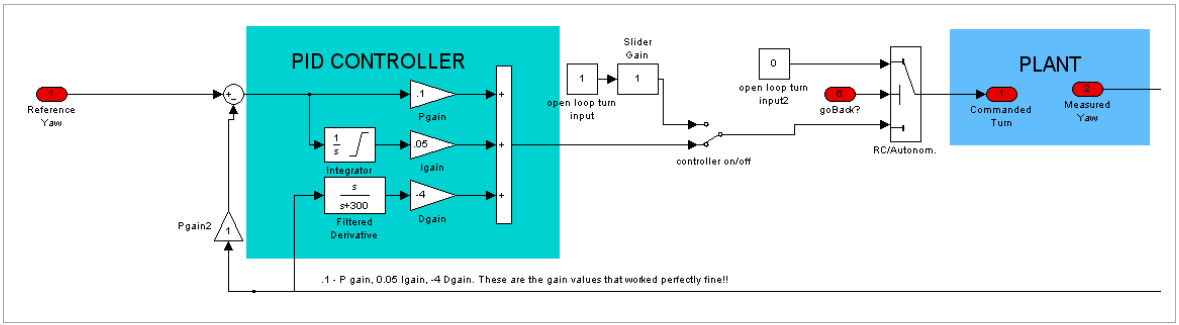

Figure 47 | Simulink diagram for the turning commands

On the other hand, the speed commands are determined regarding to the remaining distance to the specified GPS coordinate. Figure 48 shows the Simulink diagram for the speed commands:

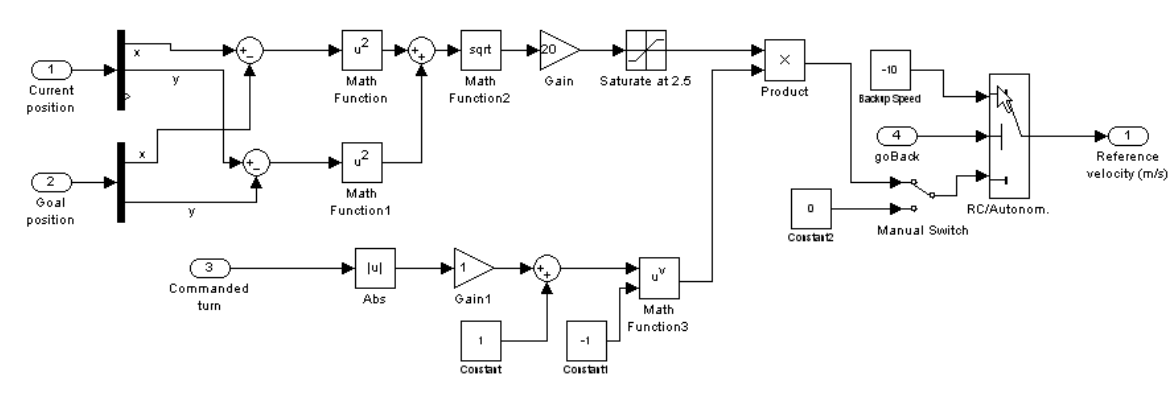

Figure 48 | Simulink diagram for the speed commands

Finally these commands have to be transformed into voltages and combined to two signals: one for the right side motors and one for the left side ones. Moreover a friction compensation term is applied depending on the surface the robot is moving. Figure 49, Figure 50 and Figure 51 show the diagrams for these operations:

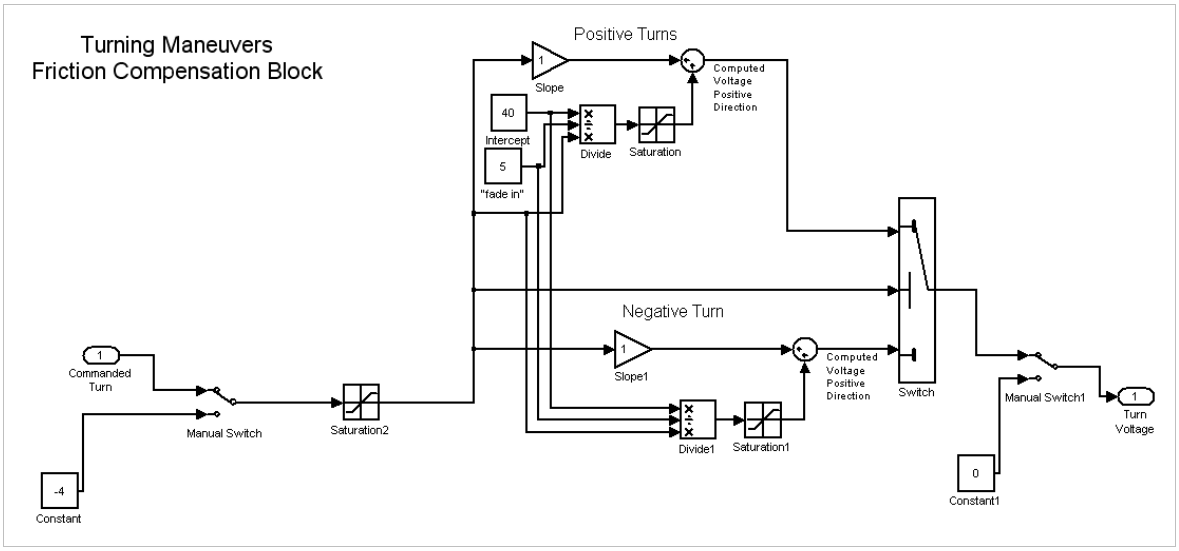

Figure 49 | Turning commands transformation to voltage

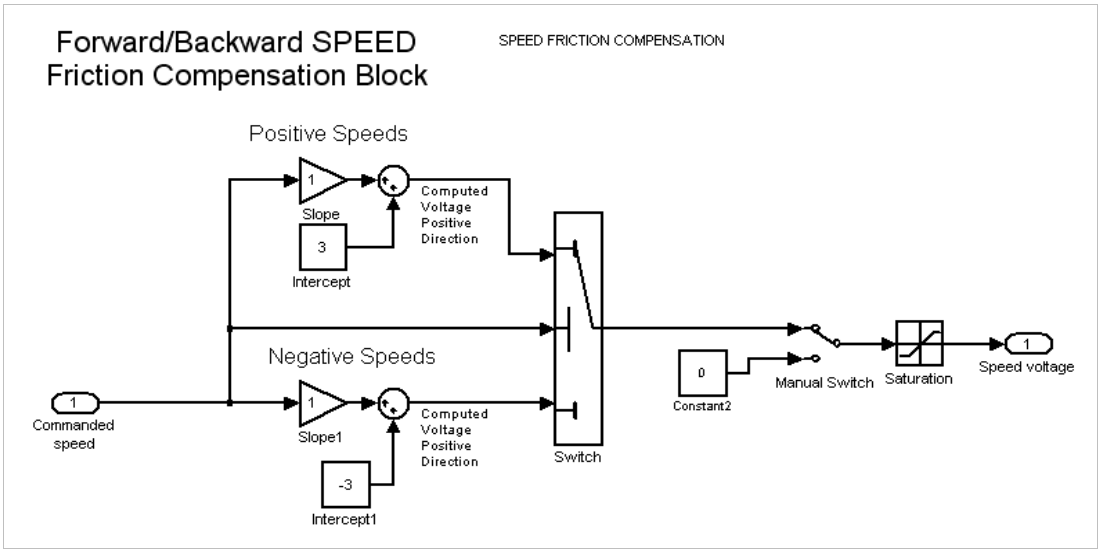

Figure 50 | Speed commands transformation to voltage

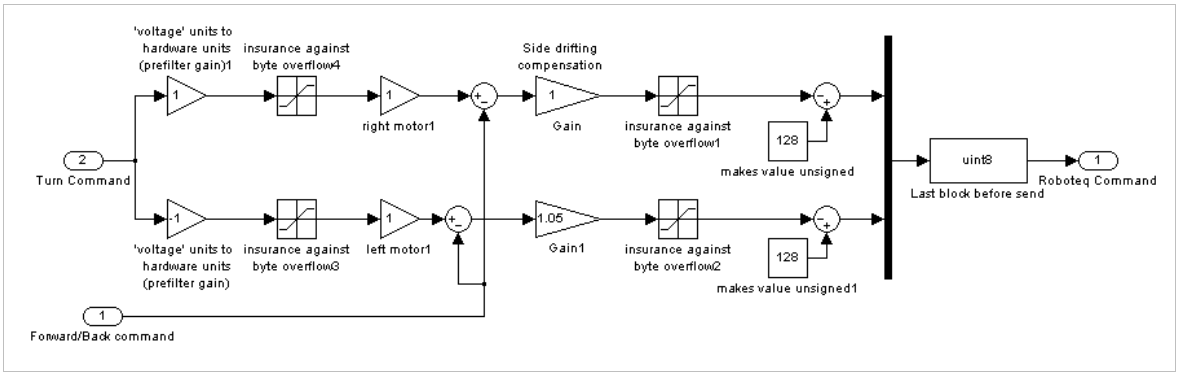

Figure 51 | Right side and left side motors signals calculation

3.10Monitoring GUI

The objective of this structure is to visually show the values of the principal variables in real time to easy check and debug the code. These variables are: obstacles and lines map, currently data seen by LIDAR sensor and camera, simulated obstacles and lines if we are running the simulation mode, the calculated path and the visited points.

All of the data is merged on a 300-by-300 matrix corresponding to the local map and thanks to a true table we defined a space color to show all these data: for obstacles black, for lines red, for the path blue, for the visited points green and if there is any error with the data yellow.

Moreover to show what obstacles or lines are currently seen by the sensors and what was seen in the past we used different color intensity: light colors for seen now, grey ones mean obstacles or lines seen in past and the darker ones are used for the simulated obstacles or lines (if we are running on simulation mode) not seen yet.

As shown in Figure 52, the robot's sensors, like the camera or range finder, are simulated and follow closely the real behavior of these. This simplifies debugging because we can control levels of sensor noise and faults. Further, with this tool, multiple developers can be testing the robot at the same time in different contexts:

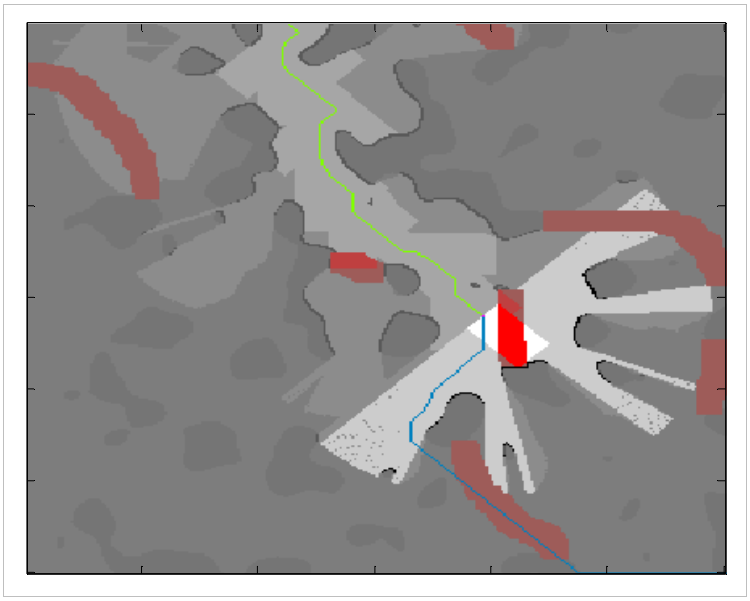

Figure 52 | Example of the monitoring GUI running a simulated environment

3.11 Simulation tool

One of the major improvements this year is the development of a powerful GUI that allows us create simulated environments (objects and lines) to first virtually test the algorithms in different conditions before testing in a physical robot environment.

This is a list of the different features that the developed GUI shown in Figure 53 includes:

- Draw obstacles and lines.
- Specify start point and multiple goal points.
- Erase the objects created
- A map generator to easy create an environment with the possibility to specify the occupancy ratio or the smoothness or the map.
- Different sizes of the brush to paint or erase.
- Opportunity to save the map on a txt file to share it or load a provided one to test the code on previous conditions.
- Run the code

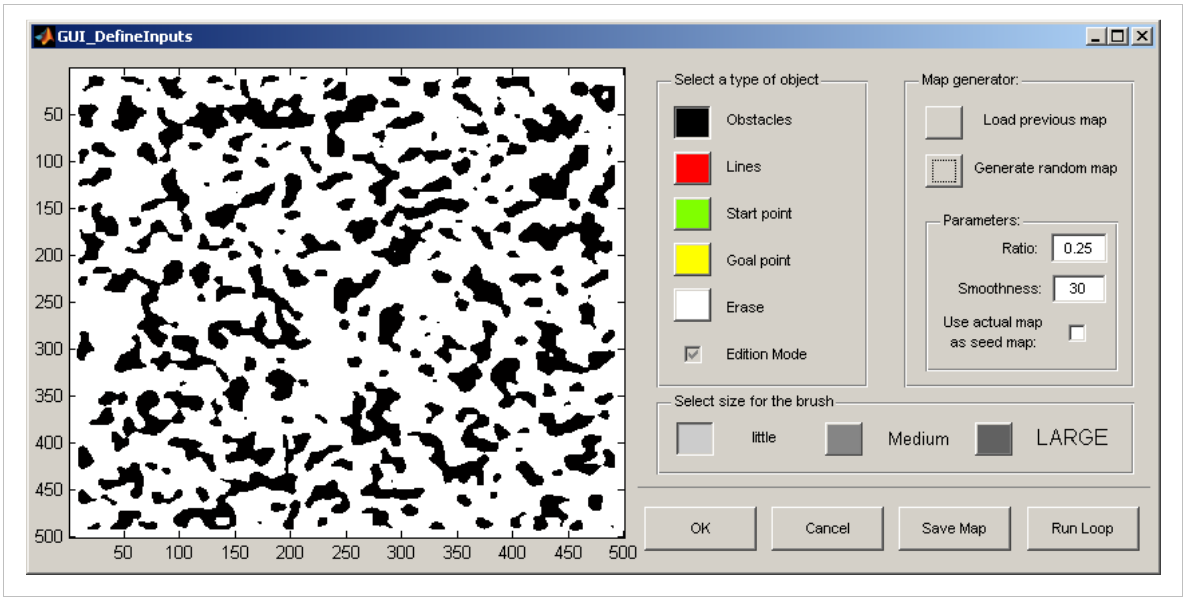

Figure 53 | GUI to simulate different environments

Chapter 4 | SYSTEM INTEGRATION

4.1 Distributed Computing

The innovative software architecture for the robot is built around a foundation of distributed computing and modularized systems. Using UDP commands, suitably modified from MATLAB's TCP/IP toolbox, all functions necessary for the robot are allocated to specific computers depending on their computational and inter-function communication loads. With this method, critical calculations and algorithms for the robot were optimized to run in parallel.

The concept behind distributed computing is quite simple. Each function for the robot is designated as a "foreground" function, with an associated "background" server that runs hidden beneath the main function. Parameters for each background server are specified during robot setup. These parameters govern the UDP communication to and from the server, the variables passed through the server, and their destinations. These background servers can be implemented on computers running MATLAB scripts or computers running Simulink diagrams with QuaRC real-time control. The foreground functions then pass variables to their respective background servers via variable flags; when a variable has been updated in the foreground, the background recognizes this update and passes the new variable to other functions distributed among different computers.

Optimization of the processing speed for various functions running on the robot can be done by allocating the function or diagram to any computer which has the capabilities of a background server. For example, one computer can run an image processing algorithm while another computer plans the robot's path, with the background servers handling any communication between these two processors. As shown in Figure 54 with robust communication and distributed computing, the robot can perform the same functions in less time.

Any number of computers can be declared with background servers, so this type of system integration is versatile, powerful, and scalable. As an added benefit, this method of distributed computing allows for a "supervisor" computer – a computer that logs the robot's performance real-time, monitoring critical information like position on a map, speed, power usage, etc.

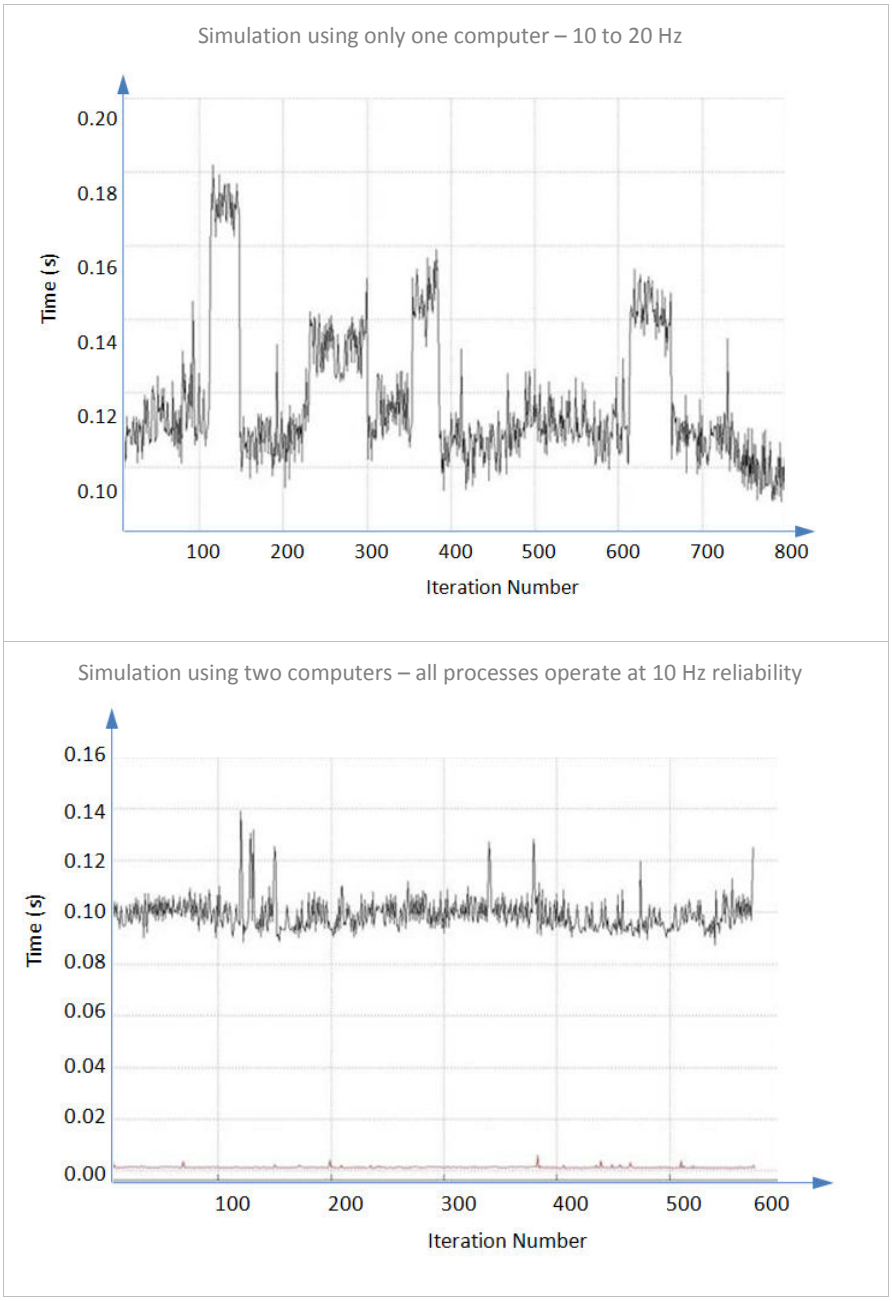

Figure 54 | Speed code comparison using one computer or distributed computing on two machines

Chapter 5 | RESULTS and CONCLUSIONS

The team did not perform well on the IGVC contest: on the design competition we were on the top ten teams of our group but we did not qualify for the final design contest. About the navigation and autonomous challenge we could not pass the qualification round so we could not compete.

Not qualifying for the final round on the design contest was mainly due to our hardware design: although our vehicle had some interesting features like the modularity, a robust power system, excellent communication structure (Arduinos + TCP/IP protocols) or being fast and capable to overcome any obstacle like sand or ramps, the judges were looking for a different approach: light weight, multiple safety measures (a part form the ones established on the rules), new energy sources… and moreover did not take in consideration that our team show up with a new software approach and the different algorithms coded.

The problem about not qualifying for the navigation and autonomous challenge was due to the code, it was not 100% ready. All the structures were done and most of them were assembled and tested on the simulation tool but not implemented on the hardware side with real data. The vehicle was not ready until 48 hours before the competition started so we had a very few time to test and fix some bugs that we could not have foreseen with only simulation and this prevent us to compete.

Despite these difficulties, participating on the IGVC contest was a great experience that showed us a lot of things which I would like to emphasize the following ones:

- The IGVC is an international contest that puts together more than 30 teams; participating is an enriching experience that taught us several things about different ways to solve certain hardware and software problems, how to organize the team members and the work…
- Preparing a robot for the IGVC cannot be done by a single person; participating on this challenge forced us to work in group, cooperate to find strategies, to solve problems and to organize ourselves to accomplish the objectives.
- Moreover our time was limited (only 9 months) so we had to deal with the pressure to have a product ready for the competition.
- Due to the characteristic of the race, it was a great way to learn about multiple interdisciplinary subjects, we had to have a global idea of the final product, to integrate hardware and software on a unique platform. Thanks to the work distribution and team meetings everyone had a minimum idea of every aspect of the product and had a global vision of our local tasks.
- Using MATLAB/Simulink as a new approach enforced us to start form the scratch with all the advantages and disadvantages that this supposes. We faced problems that we did not had

previous experience and we deal with them learning form every mistake and using the powerful tools that MATLAB provided that otherwise we could not have had.

To sum up, mention that even we could not succeed on the contest and the frustration that this involves after 9 months preparing the entry, just participating and being part of the development of the robot teach me so far that compensates the time invested.

It had been a great experience not only on the academic side also in the team work.

Chapter 6 | BIBLIOGRAPHY

Ardunio Company. *Arduino - Home Page.* August 22, 2009. http://www.arduino.cc/ (accessed May 10, 2009).

Attiya, Hagit, and Jennifer Welch. *Distributed Computing: Fundamentals, Simulations, and Advanced Topics.* Wiley-Interscience, 2004.

AUVSI (Association for Unmanned Vehicle Systems International). *IGVC - Intelligent Ground Vehicle Competition.* June 5, 1993. http://www.igvc.org/ (accessed June 20, 2009).

Harkonen, Janne, Matti Mottonen, Pekka Belt, and Harri Haapasalo. "Parallel product alternatives and verification and validation activities." *International Journal of Management and Enterprise Development*, 2009: Vol. 7, No.1 pp. 86 - 97.

Hwang, Y.K., and Ahuja Narendra. "A potential field approach to path planning." *IEEE transactions on robotics and automation*, 1992: vol. 8, nº1, pp. 23-32 (33 ref.).

National Aeronautics and Space Administration. *Systems Engineering Handbook.* Washington D.C.: National Aeronautics and Space Administration, NASA Headquarters, 2009.

Rawlinson, David, and Ray Jarvis. "Ways to tell robots where to go." *IEEE Robotics & Automation Magazine* (IEEE Xplore), June 2008: 27-36.

Roboteq, Inc. *www.roboteq.com - roboteq.com.* July 1, 2009. http://www.roboteq.com/ (accessed May 25, 2009).

Stentz, Anthony. "Optimal and Efficient Path Planning for Partially-Known Environments." *IEEE International Conference on Robotics and Automation*, May 1994: 21-29.

The MathWorks, Inc. *MATLAB Help (R2007a).* June 5, 2007.

Chapter 7 | INDEXES

7.1 Figures index

7.2 Tables index

Chapter 8 | ANNEX