Alma Mater Studiorum – Università di Bologna

DOTTORATO DI RICERCA IN

Ingegneria Biomedica, Elettrica e dei Sistemi

Ciclo 31°

Settore Concorsuale: ING-INF/04 - Automatica

Settore Scientifico Disciplinare: 09/G1 - Automatica

PERCEPTION AND LOCALIZATION TECHNIQUES FOR NAVIGATION IN AGRICULTURAL ENVIRONMENT AND EXPERIMENTAL RESULTS

Presentata da: Flavio Callegati

Coordinatore Dottorato

Prof. Daniele Vigo

Supervisore

Prof. Lorenzo Marconi

Esame finale anno 2019

Abstract

In the last decades automation has had an extraordinary expansion, whose examples are found not only in industry, but anywhere around us in everyday life: cars, automatic machines, robots and drones, both for modeling and supporting humans. The concept of automation, as the name implies, consists in automating certain actions, devices or mechanisms, with the aim of making them faster and less tiring, in perspective less expensive or dangerous for humans. Thanks to the development of new technologies, in mechanics and electronics, which allow to get smarter and more compact solutions, and more powerful computers, automation is constantly expanding in new fields and wherever its application can provide advantages for humans.

In recent years research has led to an expansion of automation, among others, in the *agricultural field*. Notoriously, it is a very hard work environment, where the operator manually carry out any job, often in extreme weather conditions or anyway heat, cold and rain, or simply where the working hours last from dawn to sunset. Due to the high investment of time and energy, considerable benefits can be gained in this field by *automating* certain operations, replacing or simply supporting the human operator. Following these motivations, appeared the first, more or less experimental robots, able to automatically plow a field, by performing a grid by means of GPS receiver, or to sow, using cameras and artificial vision algorithms.

Recently, research in this field is turning towards the development of increasingly autonomous robots, able to take care of different tasks and avoid obstacles, and so flexible to collaborate and interact with human operators. Indeed, expanding perception and computational capacity of these drones, smart robots can be created, capable of locating themselves, accepting whole missions from the opera-

tor, autonomously evaluating the conditions for carrying them out, and collecting data from the surrounding environment. The latter can then be shared with the operator, informing him about the soil moisture rather than the critical health conditions of a single plant. Thus borns the concept of *precision agriculture*, in which the robot performs its tasks according to the environment conditions it detects, distributing fertilizers or water only where necessary, thus optimizing treatments and its energy resources.
The proposed thesis project, which consists in the development of a tractor prototype able to automatically act in agricultural *semi-structured* environment, like orchards organized in rows, and navigating autonomously by means of a laser scanner. In particular, the thesis is divided into three steps. The first consists in design and construction of a tracked rover *prototype*, which has been completely realized in the laboratory, from mechanical, electric and electronic subsystems up to the software structure. The second is the development of a *navigation* and control system, which makes a generic robot able to move autonomously within rows, and from one row to the next, using a *laser scanner* as main sensor. To achieve this goal, an algorithm for rows estimation has been developed, letting the robot to locates itself within the environment. Moreover, a control law has been designed, which regulates the kinematics of the rover and computes desired velocities. Once the navigation algorithm has been defined, it is necessary to validate it. Indeed, third point consists of *experimental tests*, with the aim of testing both robot and developed autonomous navigation algorithm, whose results are presented in the last part of the thesis.

Acknowledgements

Le prime persone che vorrei ringraziare sono i miei genitori, Loris e Teresa, per avermi trasmesso la fiducia in me stesso che mi ha permesso di raggiungere tanti obiettivi e per averla rinforzata nei momenti difficili. Ringrazio di cuore la mia ragazza Chiara, per essermi stata vicino e avermi sempre sostenuto, sia durante gli studi che nel corso del dottorato. Un grazie particolare va a Nicola, per i consigli e l'esperienza che mi ha trasmesso, ma anche per le belle ore passate insieme, in laboratorio e durante i viaggi. Ringrazio anche i ragazzi del Team Rover, Alessandro e Roberto, sia per le risate che per la collaborazione nel raggiungimento di ogni risultato. Un grazie va anche al mio coordinatore, Lorenzo, per la sua guida e per i preziosi consigli che mi ha dato in questi anni. Ultimo ma non ultimo, ringrazio Rob, che mi ha guidato durante la tesi magistrale e ispirato ad iniziare il percorso verso il dottorato.

Contents

List of Figures

Chapter 1

Automation in Agriculture: State of the art

In this chapter concepts of automation and robotics application in agriculture are described in details, both highlighting environment features and related difficulties, and describing last decades progresses, applications and state of the art.

1.1 Introduction

Proposed research work was developed in the context of precision agriculture and concerns, in particular, navigation in agricultural semi-structured environments. Before proceeding with the proposed approaches and the obtained results, it is better to clarify the context and the state of the art concerning this research field. In this chapter a brief report about state of art will be presented, focusing on navigation, control and localization techniques, showing and discussing different examples and gradually motivating methods and techniques applied in this work.

1.2 Motivations

In this section some aspects of the environment and application of this research project will be discussed, in order to better contextualize it, clarify the difficulties and finally motivate approaches and choices.

1.2.1 Navigation for tasks implementation

What is expected by an agricultural robotic system is the ability to carry out different types of tasks in the field, supporting or replacing human operators. This concept holds for any platform, from the simplest one dedicated to a single task (see first example in section 1.3) to more intelligent and modular robotic systems, equipped with advanced user interfaces (see subsection 1.3.2). The main characteristics of such a robotic platform must be the dimensions, the traction capacity, the autonomy and the ability to host onboard or tow different modules or tools. This means that the basic component of an agricultural robotic platform is a tractor, or *rover*, wheeled or tracked, with different features depending on the application. Moreover, any task of the rover can be conceptually splitted into two different actions, to be carried out simultaneously and for the entire mission duration:

- Autonomous **navigation** in the orchard
- Management of the onboard **tools** for task implementation

The latter is related to the specific mission and can range from activation of a camera, for inspections, to ignition of a an air-blast sprayer for treatments distribution or managament of a mower. Navigation, or autonomous movement ability, on the other hand, is a common aspect of any mission, from less to most complex ones, and a *fundamental* requirement for any practical application of automation in the agricultural field.

1.2.2 Peculiarities of the Agricultural Environment

Agriculture environment is characterized by some specific features that make the development of navigation solutions very challenging. Harvesting, pruning, falling branches, missing trees or simply variations due to seasonal changes, cause not uniform conditions in terms of recurrent features. In addition, there is a high variation of the light conditions within rows, related to different coverage of trees and kind of canopy. Moreover, the environment is often affected by the presence of wind, making it (in terms of features) extremely variable. Consequently, the position estimation tends to be inaccurate. Finally, since the object of these project is a land vehicle, characteristics of the terrain become part of the problem. Especially during the cold season, this can be rough and affected by the presence of snow, rather than puddles or mud, with two main consequences:

- After many passages on wet ground deep pools or holes tend to form, where the vehicle risks to get bogged down or stuck (see figure 1.1).
- Vehicle slippage on soft surfaces prejudice the encoders measurements, which tend not to reflect the actual kinematics of the vehicle. In order get a precise localization, it is necessary to correct encoders data or merge with other sensors not affected by ground slippage.

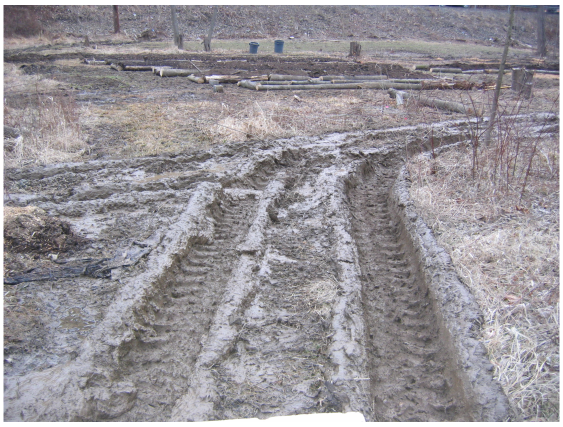


Figure 1.1: Ruts due to tractor passage on muddy terrain

Last but not least, occlusions in GPS are recurrent due to dense vegetation, while ionosphere conditions may cause errors, in absence of differential corrections. All the previous facts make the problem of localization and mapping extremely challenging and, in turn, the development of robust navigation strategies hard to be addressed.

1.2.3 Absolute and relative localization

GPS receivers (see section A.2) have always been a fundamental component of many outdoor navigation approaches (see section 1.3.1). Indeed, absolute position on the earth's surface can be known in any meteorological condition, but provided that satellite coverage is sufficient. However, there are a lot of drawbacks in using GPS technology:

- GPS position is not very precise: for this reason GNSS-RTK receivers (see section A.2) or sensor fusion techniques are employed to achieve better accuracy.
- This kind of solutions are efficient as long as the robot is in open field environments (see figure 1.2), i.e. until there are no constructions, rather than thick vegetation or terrain conformations that cause a decrease or complete lack of satellite coverage.
- GPS, and sensors commonly used for data fusion, do not provide any kind of **perception** about the surrounding environment. Indeed, the GPS provides an absolute position, encoders (subsection A.1) the motors rotation speeds, and IMU gives back attitude, rotations or accelerations.

Figure 1.2: Wheat field, open field environment example

For these reasons, GPS-based navigation requires a perfect knowledge of the surrounding environment, in order to be feasible. This condition, in an outdoor or agricultural environment, is difficult to achieve, as unexpected obstacles can always be present and it is necessary to take into account seasonal changes in vegetation, branch growth or fallen branches due to strong wind, rather than puddles or muddy or icy areas, due to adverse weather conditions. Therefore, it is necessary to use proximity sensors, such as *ultrasonic* sensors (section A.4) or *LIDAR* (section A.5), which allow to evaluate the conformation of the ground or surrounding vegetation and to detect the presence of any obstacle on the path.
Moreover, the necessity of the GPS signal for navigation tends to reduce when working in a *structured* or *semi-structured* environment, like crops organized in

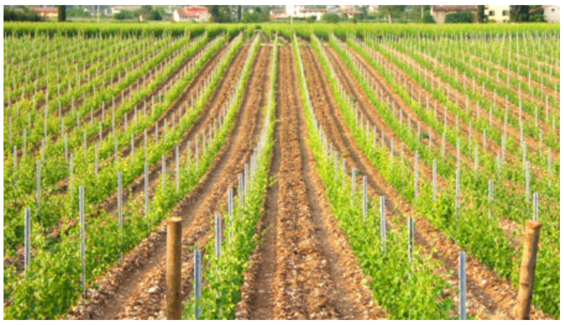
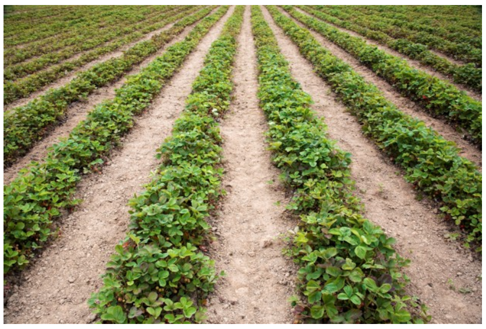



Figure 1.3: Vineyard (left) and crop field (right) organized in row structures

rows (see 1.3). Within this environment, the absolute position is not so important for navigation as the **relative** one respect to the surrounding structure. Indeed, known the lateral distance, the robot can move keeping the row center , while from the front view it is able to assess any obstacle presence. Using LIDAR or cameras (section A.6), and real-time processing architectures, the robot will be able to locate itself within rows, create a map and evaluate its attitude with respect to the lines of trees. In this context, GPS covers a marginal role, being used to have redundant information about rover position in a row or in another one, or know its proximity to the extremities of the rows themselves.
In order to conclude this discussion about localization and sensors and motivate the next projectual choises, a comparison between the two most advanced proximity sensors currently used in agriculture, i.e. cameras and LIDAR, is proposed. The most important advantages and drawbacks of the two devices can be summarized as follows:

- Cameras are lighter and less expensive than laser scanners and can realize all the typical functions of LIDARs, from mapping to localization, supported by colors that characterize objects and surrounding environment.
- On the other hand, the application on a tractor does not involve problems related to sensor size or weight. In addition, LIDAR has a greater range. Moreover, what is foundamental for navigation is the distance with respect to surrounding environment, rather than details or colors of the different objects, which are contained in camera images.
- Working outdoor, the robot should navigate in various weather conditions like rain, wind, darkness or fog, while the sensor could get dirty due to wind

or dust generated during work. For this point of view, LIDARs seems to behave better than cameras, since they are tested for outdoor environments and their performances are not affected by light conditions.

In view of these motivations, LIDARs and cameras seem to be both suitable for providing environment perception and supporting outdoor autonomous navigation. However, in the proposed agricultural application a 3D laser scanner has been chosen, expecially for its capacity of working in adverse weather conditions, but with the knowledge of its high cost and drawbacks.
Next experiments will show if this choice is actually correct, or further evaluations are needed. For example, it could emerge that the proposed navigation algorithms do not need so high sensor performances, and maybe just a less expensive 2D laser scanner will be employed. Of course, also a stereo camera could be a suitable choice, expecially if some fruit or plant deseases detection algorithms will be implemented onboard, in which evironment color details are foundamental, and the whole surroundings perception can be provided by means of a singe sensor.

1.3 State of the art

1.3.1 Navigation in agricultural environment

In the last decades automation and robotics applications are constantly expanding in many fields, leading to considerable savings in terms of cost, as well as fatigue or time for the human operator. Agriculture has followed this trend and has been subject of many research projects, which seek new solutions to automate operations like inspection, planting, plowing or harvesting. These approaches vary widely, ranging, for example, from the simplest ones based on *single* sensors, as in the case of GPS-based solutions used for automating plowing or threshing over large areas, to the one using entire sensor suites, whose data are merged to obtain more and more precise and reliable localization in the environment. Depending on the applications, inexpensive solutions can be provided, taking advantage of cheap sensors, up to more expensive and performant ones, using for example modern LIDAR sensors, through which visual odometry, mapping and obstacle detection can be implemented, or GNSS-RTK receivers, capable of providing centimetre ac-

curacy.
One of the first applications is an automatic control system for melon harvesting [25], developed about twenty years ago by an Israeli research group. The experimental platform was composed by a cartesian robot manipulator, having the task of picking fruits, mounted on a mobile platform pulled by a tractor along the rows (see figure 1.4). In this application, the fundamental component for melon detection was a camera, whose images were then filtered and enhanced using computer vision algorithms. The result of the experimental tests was a success rate of 85% in melons detection and picking. In this work the robotic harvester control is independent from the movement along the rows, provided by the tractor, meaning that problems like navigation or localization in the field are not taken into account. The work is indeed focused on fruit detection and cartesian harvester motion control .
Differently, in order to automate operations like plowing or fertilizer distribution,

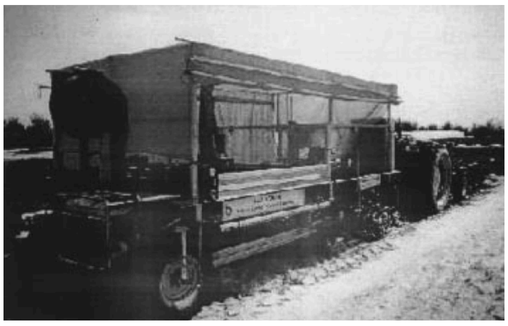


Figure 1.4: Robotic melon harvester pulled by tractor

the movement of the platform has also to be managed and the control problem becomes more complicated. In fact, the system must be able to autonomously estimate its position in the environment, as well as evaluating slope and consistency of the terrain and detect and manage any obstacles presence. The problems are therefore both at the level of *environment perception* and trajectory tracking, and of *safety*, for people, surrounding environment or the robot itself, due to its size and traction capacity.
An example of research about *ground prediction* for safety reasons is given in [24].This work deals with the problem of navigating in vegetation or farm conditions, where terrain characteristics change and vegetation itself can hide holes or obstacles. Here, an online adaptive approach to automatically predict ground con-

dition is presented. Pose estimation is carried out by means of a GPS unit, a 3-axis gyroscope, a doppler radar and encoders on wheels and steering unit. Differently, the ground prediction is based on a couple of stereo cameras and two LADAR sensors. The presented algorithm is not model-based, but relies on a learning approach: the predictions are developed looking forward, with the sensors, and using the vehicle past experience.
In 2000 John Deere company released the *Autotrac System*, a commercial GPS guidance system available for different models of its tractors and agricultural machines. Such a system helps the human operator in the steering action, expecially in low visibility conditions, defining a more efficient trajectory and lowering production costs.
In 2005 a guidance system based only on *GPS* technology was developed and tested on an autonomous mobile platform [12]. The robot receives *waypoints* from the user through an RF communication system, by which the UGV position can be monitored on a remote PC. This generic results can be applied in various fields, such as transports, agriculture, and so on. The most important limitation of the approach is the missing of informations about*relative* position between robot and surrounding environment. Indeed, the vehicle can move anywhere on the earth's surface, but it is *blind* to obstacles, terrain conditions, holes and vegetation.
Moreover, such a system is efficient only in open or wide spaces, since the precision of a common GPS receiver is about one meter. This limits the possibility to act in narrow environments or performing strict maneuvers, needed to move in a street or in a row of trees. The solutions to that problem could be the adoption of more performant sensors, such as a differential *GNSS-RTK* receiver. That kind of device can ensure an accuracy of centimeters, but, as a main drawback, has a very high cost. Other approaches could be the installation of distance sensors, by which the system can percept the surrounding environment, or using sensor *fusion* and combining for example GPS and IMU. The approach proposed in [21] consists in fusing data from a common GPS receiver and proximity sensors. Actually, the latter are not mounted onboard, like lasers or sonars, but *landmarks* installed in the field and broadcasting their position. When the vehicle approaches those sensors, its position is corrected or calibrated. The main advantage of this system is that the position correction is exact, since the landmark location is perfectly known.

The drawbacks are the position estimation drift, which tends to appear in the path between sensors, and the necessity to prepare or *adapt* the work environment to the vehicle, precluding its application fields and its flexibility.
In a different way [15], GPS performances can be improved using multiple, i.e. two or three, GPS receivers placed over the vehicle at a certain mutual distance (see figure 1.5). Through data fusion, for example by means of a Kalman filter,

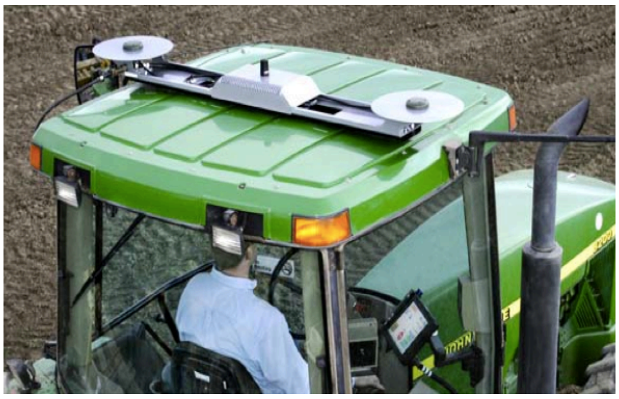


Figure 1.5: Dual GNSS-RTK receiver mounted on the tractor roof

the position error can be reduced and also the yaw angle (see section 3.2) can be estimated. Since the distance between receivers is obviously limited by the vehicle dimensions, the use of low-cost GPS receivers, with one meter precision, does not produce appreciable results and leads to the necessity of expensive GNSS-RTK sensors. So, even if this approach gives good results in terms of absolute position, has the side effect of being really expensive.
In 2011 a Japanese research group presented an autonomous tractor able to harvest and move in a rice field [26]. The problem was basically to control the vehicle through a row of plants, then turn to the next one and so on, moving along a sort of grid. The proposed guidance law is based on GPS receiver and IMU, giving respectively global position and heading of the robot.
Another example of GPS-based navigation in agricultural environment is given in [22]. Here a GPS based guidance law is presented, which allows a robot to autonomously navigate in a farm environment. The vehicle heading is provided by a digital compass, which provides the capability of steering following a predefined path.
Exploiting the precision of a RTK GPS, an autonomous tractor can also navigate

successfully in a narrow environment, such as orchard rows. In [11] an example of this approach is described. Known the exact GPS location of the field and the single rows, a predefined trajectory for the robot can be computed. Then,using a RTK GPS receiver and an IMU, which gives the heading, the system calculates and controls the position error, providing the capability to navigate within the rows of a generic field following the desired path. This kind of approach is nominally correct, but based on a *perfect knowledge* of the field and on the hypothesis of obstacles absence and no variations in vegetation or ground conformation.
The position estimation can be improved fusing data from GPS and a camera. In [20] a small automatic robot is developed, with the purpose of sowing seeds. The vision system builds a local map, identifying the path to follow, and these data si correlated to the GPS coordinates, in order to localize the robot. Thanks to an ultrasonic sensor placed in the front of the vehicle, is also implemented a simple obstacle avoidance task, which stops the robot if some object is detected on its path.
Some examples of robotics and automation applications in the agricultural field have been discussed so far, which are generally based on the specific application, such as rice or melons harvesting, sowing, or polarized towards a navigation idea based on GPS technology. Fasting forward a few years, more recent developments and improvements in this approaches can be observed. This evolution will be discussed in the two following subsections (1.3.2 and 1.3.3), which describes a more conceptual aspect, related to *human machine interaction*, and an improvement related to *environment perception* and new generation sensors, respectively.

1.3.2 Smart agricultural robotic platforms

Up to now different solutions of agricultural robots were presented, basically designed just to navigate in outdoor environment and performing well delimited tasks. The main purpose was to move in the orchard, with a poor or absent *interaction* between robot and human operator, and no particular smart functions. Over time the idea of agricultural robots has evolved and now tends to mean a *robotic platform, equipped with a suite of sensors by which it can navigate autonomously in different kinds of crops, collect data from the field, monitor the health of plants and interact with the user.* The described robotic system is generally composed

by a team of robots, i.e. UGVs for the heavy tasks and UAVs for monitor and inspection functions, a user interface, throw which the operator can both monitor farm state and assign missions to the system and finally a dynamic database, containing data collected during works and inspections and shared with the user. The last component is a wireless network which covers all the farm and provide communications between robots, database and human operator. So, in this vision of automated farm, the rover, or automatic tractor, becomes just a part of the whole robotic system.
One of the first examples (2015) of this concept is given in [10]. In this work is described "Agrobot" architecture (see figure 1.6), a smart agricultural robot able to execute many tasks and monitor field conditions, from disease diagnosis to soil analysis and so on. Moreover, part of the project is also the implementation of a *cloud service*, containing and sharing weather informations, data collected by the robot or treatments parameters. In addition to the basic mechanical and electronic components, the rover is equipped with a wireless communication board, camera and live video transmitter, in order to provide interaction and communication with user or other devices. Moreover, thanks to the presence of distance sensors, camera, IMU and GPS receiver, Agrobot can ensure the navigation capabilities needed to autonomously perform tasks and missions in the field.

During the same year, Bergerman and his team presents an agricultural robotic

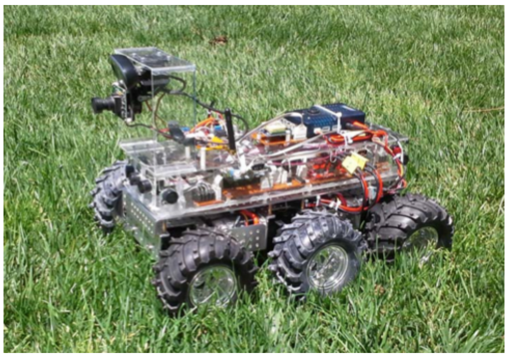


Figure 1.6: Agrobot operating in the field

system able to navigate autonomously whithin rown of trees and equipped with a *remote user interface* [17]. The platform is a car-like robot (see figure 1.7), equipped with a 2D LIDAR sensor as foundamental component of the environment perception system. Indeed, navigation is essentially based on encoders and

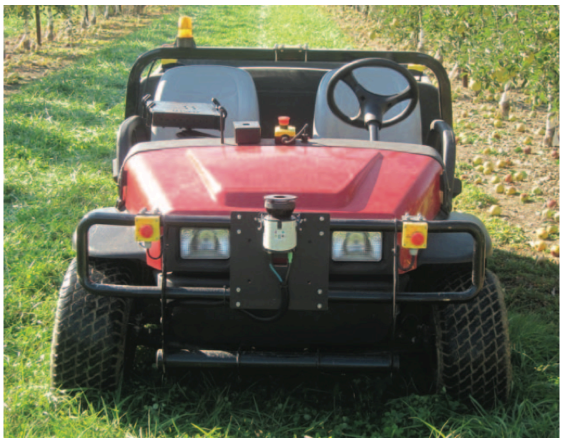
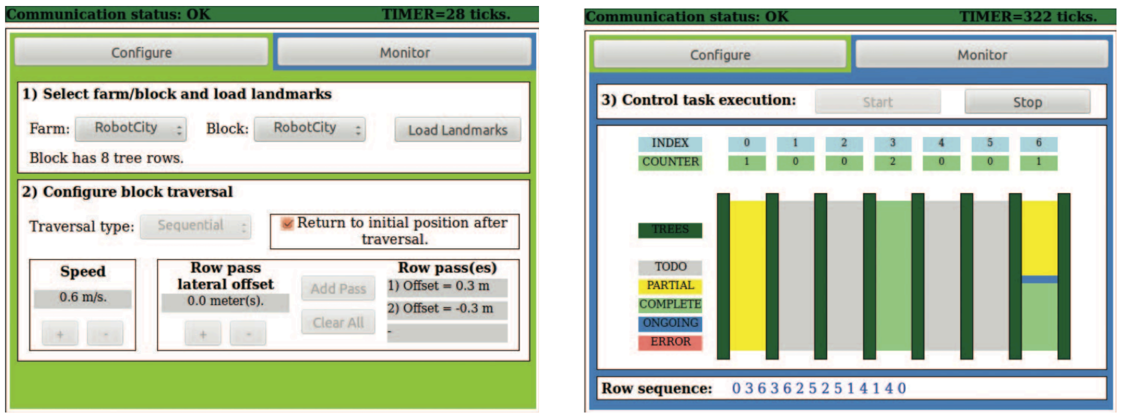


Figure 1.7: Experimental robot farmer

laser scanners, with no use of GPS. From human machine interfacement point of view, the most interesting progress is the development of a remote interface, running on tablet or laptop, and communicating with the robot onboard computer by means of a wireless network. Through a first screen (see figure 1.8) the user can assign missions to the robot, deciding the field where to work and specifying the rows to go through rather that to skip and defining the desired lateral distance from the trees.
In a second screen (see figure 1.8) the operator can start the mission and monitor its advancements on the different rows, i.e. see if the work in the single row is completed, has already to start or is in progress.
These interesting examples show the possibility to realize agricultural robotis plat-


Communication status: OK TIMER=28 ticks.

1) Select farm/block and load landmarks
Farm: RobotCity Block: RobotCity
Block has 8 tree rows.

2) Configure block traversal
Traversal type: Sequential
Return to initial position after traversal.
Speed: 0.6 m/s.
Row pass lateral offset: 0.0 meter(s).
Row pass(es):
1) Offset = 0.3 m
2) Offset = -0.3 m

Communication status: OK TIMER=322 ticks.

3) Control task execution:

INDEX	0	1	2	3	4	5	6
COUNTER	1	0	0	2	0	0	1

Row sequence: 0 3 6 3 6 2 5 2 5 1 4 1 4 0

Figure 1.8: HMI Mission assignment (left) and monitoring (right) screens

forms able to navigate in the farm environment, while autonomously performing different operations, and even more to interact and communicate with the human operator. This conceptually opens the doors towards a future in which agriculture will be completely automated and the farmers have a team of robots, can easily assign missions and monitor their progresses from a remote position. The advanced user interface and the sensor suite whereby robots will be equipped will allow the operator to monitor plants health, detect malfunctions of faults, rather than directly observe the robot at work, thanks to onboard cameras and possibility to streaming data. In this vision all the work is demanded to the robots and the operator can remotely monitor his farm and robot team as if he were present in the field.

1.3.3 Navigation in structured agricultural environment

Up to now (see subsection 1.3.1) some esamples of navigation tecniques in agriculture field have been discussed, were proximity sensors assumed a marginal role, thus giving little importance to the surrounding environment perception. When operating in a structured or semi-structured environment, such as an orchard organized in rows, knowledge of the absolute position takes on a less significant role in navigation, while the concept of *environment perception* and *relative position* respect to the structure becomes more important. Indeed, once a row has been taken, the goal of the robot is just to navigate within the lines of trees and then move to the next rows. This means that the robot only needs to know its relative position within rows, which can be obtained through advanced proximity sensors, combined with *computer vision algorithms.* Over time, technological progress has made available high-performance sensors, which allow a visual perception of the environment that tends to get closer to the human one: cameras and LIDAR.
These sensors returns images or pointclouds (see section A.5) of the surrounding environment. Through computer vision algorithms, sensor data can be enriched, extracting models or features. In agriculture this allows to estimate trees position or row lines, detect rows entry or exit and navigate inside it, and recognize different types of obstacles. Exploiting that relative position feedback, a control loop can be designed, in order to keep the tractor in the middle of the corridor environment and, known the rows position, discriminate obstacles not belonging

to the structure. On this basis an obstacle avoidance system can be implemented, with recognition of the obstacle type and precise recovery actions.
Assuming a marginal role, GPS does not necessarily have to guarantee high performances. In fact, a common GPS receiver can be employed, whose precision is sufficient to discriminate the row where the robot is moving or to get an idea of proximity to the row extremities. Moreover, since navigation is no longer based on GPS, the risks related to low satellites coverage or disturbances due to particularly thick vegetation, which would tend to mask the satellite signals, are reduced. Obviously, other sensors, like encoders or IMU, continue to be installed onboard. The idea is just to *base* navigation on LIDAR or camera, while reading and using other sensors for safety and redundancy reasons, in order to have a greater accuracy in position estimation, on one hand, and to be able to continue mission or autonomous movement as well as possible, in spite of LIDAR failures.
One of the first applications of these techniques dates back to 1997, when a video guidance system for vegetable gardens structured in rows was presented [27]. The environment perception system was based on a color video camera, whose images were then filtered using a series of innovative algorithms, in order to extract the plant lines. Given this feedback, a control loop for distance regulation respect to plants was closed, using the robot's steering as control variable. After a decade, in 2005, the results of a research project are published, presenting an autonomous driving system in the agricultural environment based on stereocamera and row detection algorithm [16]. The latter was divided in three steps: processing of the stereo image, creation of a 3D elevation map (see figure 1.9) and navigation points definition. This process outputs a steering signal for the robot, in order to guide

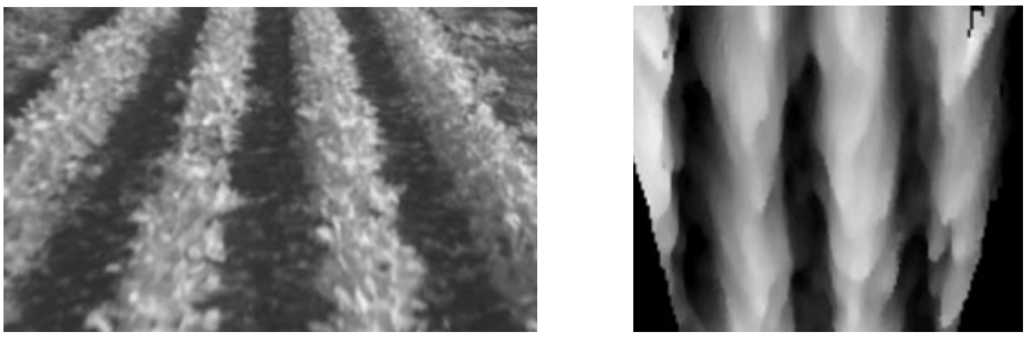
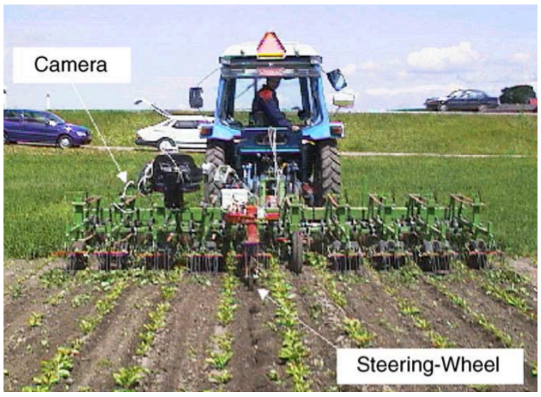


Figure 1.9: Raw (left) grey-scale image and developed elevation map (grey) of the rows

it between the rows of plants. The described experimental tests were carried out in soybean fields (see figure 1.10), i.e. an environment where the robot moves over the lines of plants and, thanks to the camera, has a top view of the cultivated area. In the same year, a Swedish research team develop a camera-based system

Figure 1.10: Picture of soybean field

able to estimate the lines of plants within garden environments [2]. The system is equipped with a grey-scale front camera pointing downward and providing an image of the ground, which contains rows of plants and various disturbances, such as other plants or grass. The ground is modeled as a plane, and the rows of plants as parallel lines. Distance and attitude estimation of the camera with respect to those lines is carried out through post-processing algorithms, and in particular using the *Hough transform* (see section B.3). Since the camera frames two rows of plants, the algorithm returns a double measure of angle and distance, thus allowing redundancy and providing a more robust and precise result. Known the plants position, a control loop is closed in order to regulate the robot steering and make it able to follow the estimated lines. The proposed approach [2] is tested using two platforms: an inner-row cultivator pulled by a tractor that the farmer drives and a wheeled autonomous robot (see figure 1.11). In the first case the control input is a steering device present on the cultivator, which allows to adjust the transversal position with respect to the plants, while in the second case consists directly the steering unit of the robot. In both cases the experiments confirm excellent performances, with centimetric errors compared to the ideal rows lines.
In the same period laser scanners, or LIDAR, start to be applied also in agricultural field and new applications are born, in which these sensors are used in place of cameras for environment perception and rows estimation.
In 2010, a Pittsburg research group presents a LIDAR-based guidance system that



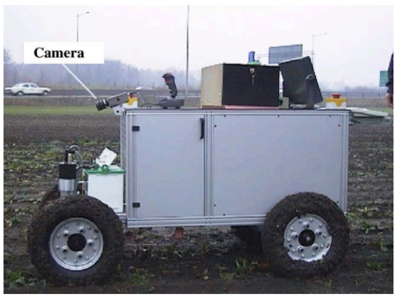


Figure 1.11: Experimental platforms: tractor-pulled cultivator (left) and autonomous robot (right)

allows an agricultural rover to navigate autonomously through orchard rows [3]. The system uses two *planar* laser scanners, installed in the front of the experimental vehicle and facing not forward but slightly (30°) towards right left directions, respectively (see figure 1.12) LIDAR data post-processing algorithm is based on

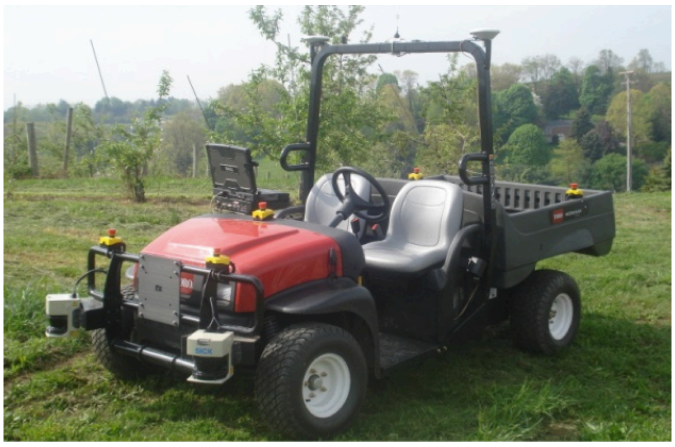


Figure 1.12: Experimental platform, equipped with a couple of 2D laser scanners

the Hough transform, which allows the estimation of row lines (see figure 1.13) within the rover is navigating in the middle of. The navigation system is validated through an intensive series of experimental tests, during which the rover has been driven autonomously for a total of 130 km. In addition, a guidance law for row change, i.e. from the present to the desired one, is also proposed. This is done by describing a predefined semicircle by means of just encoders feedback, until the detection algorithm recongnizes the new row and an entrance trajectory is gener-

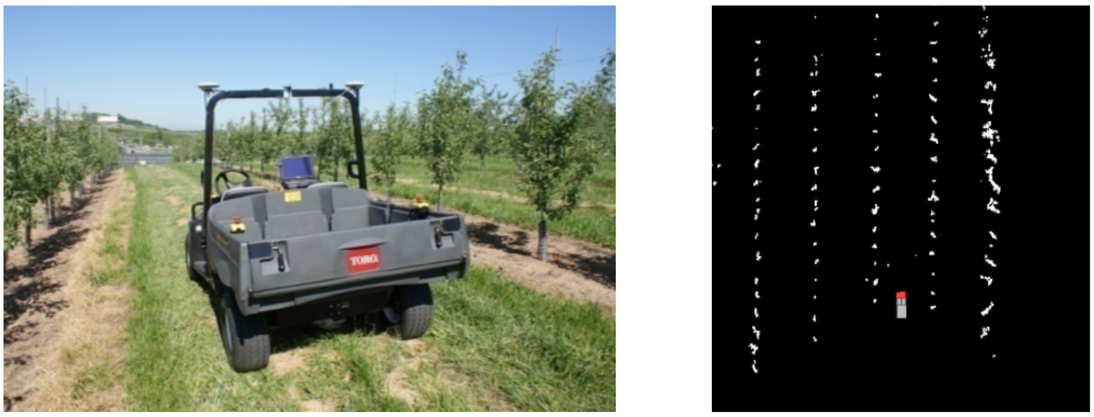


Figure 1.13: Experimental car moving within rows (left) and LIDAR environment perception (right)

ated.
In 2015, another guidance system for agricultural robots based on laser scanners is presented, which allows to navigate autonomously within rows and from one row to the next [8]. What is new is that here the focus is not on the rows estimation, but on trajectory computation and tracking. In particular, an asymptotically stable path tracking controller is presented, combined with a trajectory generator. The position within rows is estimated by integration and filtering of encoders feedback, as regards the longitudinal position, and by means of laser scanner and a simple tree lines estimation, as regards the transverse position and the relative orientation. Finally, about the row change, some trajectories are proposed (see figure

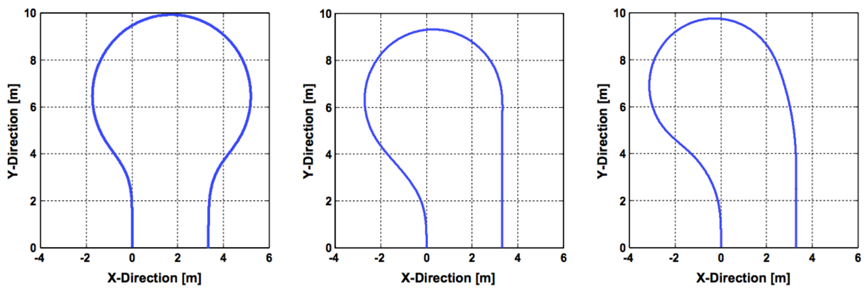


Figure 1.14: Row change proposed trajectories: lamp (left), clothoid(center) and thumb(right)

1.14) in order to

- reflect as much as possible the path described by a human operator
- guarantee, at the end of the maneuver, a position as central and parallel as

possible with respect to the new row

In this way, the visibility of the new row is maximized, while the noise caused by the other rows is minimized, guaranteeing a more effective and safe maneuver.
During the same year an Australian research group presents a localization system for agricultural applications using a 2D laser scanner, based on single trees recognition [13]. The experiments, conducted in different seasons, show the possibility to use the proposed approach in order to navigate in the field, in presence of vegetation seasonal changes and without using GPS.
In 2016, Bergerman and his team present an improvement in their previous work, with the addition of a model and a driving system that compensate for *wheel slippage* [9].
The novelty is the development of a kinematic model which takes into account wheel slippage and a state observer estimating the position error, exploiting an RTK-GPS. The latter is employed only for this purpose, while environment perception and vehicle control are based on a LIDAR sensor. The experimental tests are carried out both in open field, in presence of *snow*, and within orchard rows, in muddy terrain conditions (see figure 1.15). The results show a marked improvement in trajectory tracking respect to the case in which slippage was not taken into account. Another interesting example of LIDAR data post processing and

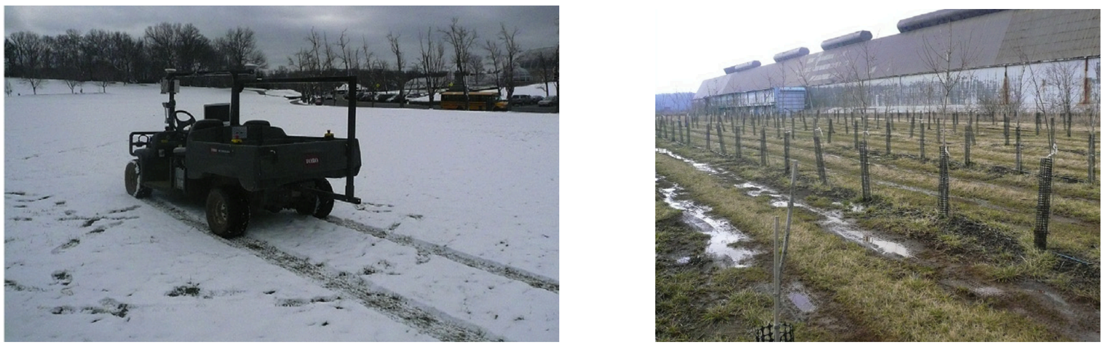


Figure 1.15: Eperimental tests environments, with presence of snow (left) and mud (right)

environment perception is presented in [1]. The application field is autonomous navigation in road environment, differently from the literature introduced so far. In the proposed approach, RANSAC algorithm (see section B.2) is applied for obstacle detection, both for fixed or moving objects. Obstacle detection system is based on 3D LIDAR sensor: most important environment features, such as road plane, are extracted from the pointcloud and then the different obstacles, which emerge from that surfaces, are discriminated. This work reveals the potential of RANSAC in detecting different kinds of features from 3D pointclouds, even in presence of noise, complex environments and when dealing with moving objects. Thanks to its good performances and robusteness respect to disturbances, RANSAC algorithm could maybe be applied also in agricultural environment. The idea could be to estimate the rows of trees, in some way, and the ground, modeled as a plane, using RANSAC. At this point, anything which compares in the pointloud outside from trees and ground is a potential obstacle and becomes easy to be detected.

1.4 Conclusions

This chapter gave a brief introduction to problems and solution approaches concerning navigation in the agricultural environment and to the relating state of the art, obviously focused on the most promising and innovative techniques. In particular, the solutions based on LIDAR sensors have been analyzed in more detail, highlighting potential and advantages related to their use for navigating within orchard rows. Next chapter deals with development and construction of an agricultural robotic platform, foundamental for experimental tests (see chapter 5) and navigation algorithm validation (chapter 3)

Chapter 2

Platform description

This chapter describes the agricultural platform developed during the research project and employed in carried out experimental tests, focusing on design choises and physical structure, composed of mechanics, electrics, electronics and sensors.

2.1 Introduction

In the last chapter, an overview about the application of automation and robotics in agriculture has been given and many examples of last decades research results in this field have been discussed. After a proper introduction, this chapter starts the presentation of the real carried out research work. In the light of discussions reported in chapter 1, expecially about orchard environment features and proposed agricultural robots, during last months an experimental robotic platform has been designed and constructed. Of course, the idea was to create a rover able to move and work in the agricultural environment (described in chapter 1). For this reason, during robot development, all its features have been curated in order to overcome those environment difficulties. The result of this work is showed in figure 2.1. In the next sections, the developed robotic platform will be described in all its features. The discussion is organized in three different sections, corresponding to the mechanical structure (2.2), electric subsystem (2.3) and electronics and sensors (2.4).

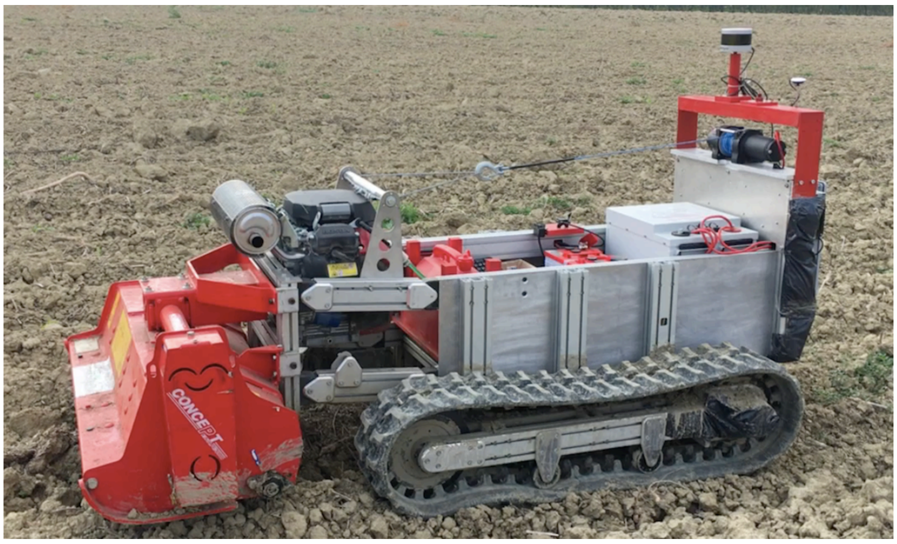


Figure 2.1: Picture of the developed agricultural robot.

2.2 Mechanical description

In this section, an exposition of the main mechanical components will be given, highlighting key features and motivations behind the choices taken in the different design steps. In particular, most relevant aspects, which will be discussed later, are presented here below.

- **Frame**, heart of the mechanical structure
- **Tracks**, which provides motion capabilities
- **Tools**, foundamental for implementing different kind of agricultural tasks

2.2.1 Frame and mechanical structure

In figure 2.2 a view of the vehicle frame is proposed. As showed in the picture, the strucutre ("skeleton") of the vehicle is composed by pre-shaped aluminum beams. This choice is motivated by **modularity**, since they allow the designer to easily modify the structure without the need of any soldering or complex procedures. This feature is very important, since the robot is an experimental platform, and improvements or modifications have to be frequently carried out. Beside granting an high level of modularity, these particular beams can be used to host the wiring (see figure 2.2, where cables goes out from the robot frame), solving in a

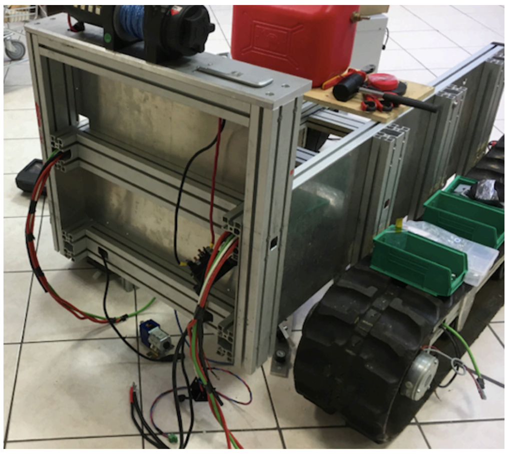


Figure 2.2: Picture of the robot construction, showing the aluminium frame.

compact and tidy way the problem of routing wirings. The frame has the purpose of withstanding the mechanical stresses generated by motion and presence of active farming tools. Moreover, it has to support different heavy payloads, like tools themselves and batteries, which needs to be protected and fixed in a safe way, possibly damping vibrations and oscillations generated by robot motion and uneven ground.

2.2.2 Tracks

Motion capability for the robot is provided by means of continuous rubber tracks. This kind of solution has been preferred to *wheels*, since it guarantee an higher level of traction, thanks to a wider contact surface with the ground, and an higher stability in case of bumpy environment, endowing the robot with an enhanced robustness during motion. As figure 2.3 shows, the track structure presents

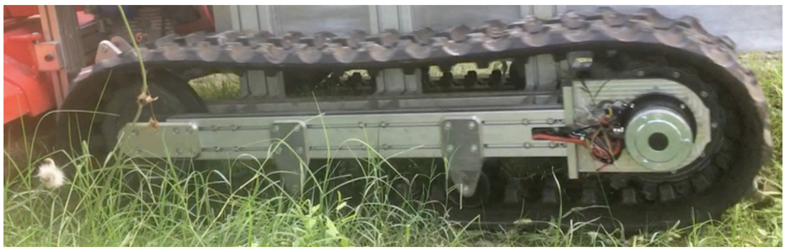


Figure 2.3: Picture of the left robot track.

two bigger wheels. The one on the front is a castor wheel, used to stretch the track itself and keeping a correct shape, while the rear one, connected to a motor through a gearbox, transmits the motion. This driving chain (motor and gearbox) have been embedded inside the track structure, in order to save space and create a *module* which can be easily removed and substituted. Between these two bigger wheels there are two *road-wheels*, which have the aim of increasing the uniformity of normal *pressure distribution* on the contact surface. A planned future development is the addition of more road-wheels in order to make the pressure distribution more uniform and therefore to reduce the resistive torque during turning maneuvers. Finally, in the picture can be seen the motor and the relative brake, which consists in the last part of the cylindrical body of the electric machine.

2.2.3 Tool Modules

The vehicle is designed for acting in farming environment, performing different tasks while navigating autonomously. Consequently, for any kind of mission, the robot have to be equipped with proper tools, which, for the sake of modularity, need to be easily plugged or unplugged into the vehicle frame. For this reason, a particular quadrilateral structure have been built in the front of the rover. The latter is composed by two four-bar linkages (see right side of figure 2.4) and has the main feature of having four connecting pins that create the junction between tool and frame. So, the bars composing this structure are not fixed, but can rotate

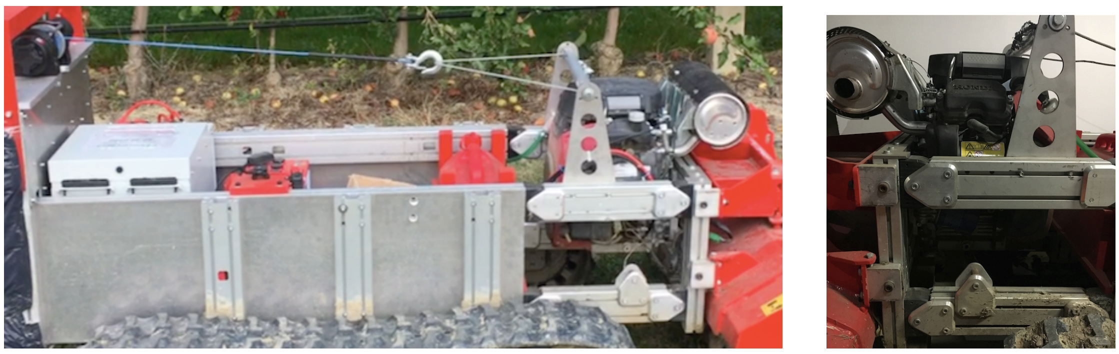


Figure 2.4: Mechanical system for plugging and managing tools (left) and detail of the quadrilater structure (left).

around their fixing points on the frame, giving the possibility to lift or lower the attached tool. In this way, the tool height can be adjusted to the particular task or work conditions, also helping the robot during curves or stricht maneuvers, lifting the structure and reducing the ground friction. This mechanism is actuated by an electric winch (see left side of figure 2.4) that, winding or unwinding, makes the four-bar linkage move along vertical direction. Within the above-mentioned mechanism a combustion engine is accommodated, in order to provide mechanical power to the tools mounted onboard. This engine is fed with gasoline (the gas can is visible in figure kjsndf) and generates a power of 40 HP. The mechanical interface between motor and tool is the power take-off used for tractors, meaning that the robot can manage standard agricultural tools, with no need of adaptations or additional components.

2.3 Electrical description

This section describes the electric system of the robot, foundamental for providing motion capacity, winch movements and feeding onboard electronic devices and sensors. The most important components of this subsystem are:

- Lithium batteries
- Motors drivers
- Motors
- Brakes
- Lead battery
- Winch power circuit
- *DC-DC* converters

Batteries are placed in the central part of the robot frame (see figure 2.1), motors and brakes are installed inside tracks structure, while all the other electronic components are mounted on a vertical *aluminium plate* (see figure 2.6) at the rear of the vehicle, in order to save space and create some kind of heat sink, making it easy for electronic components to disperse heat. Moreover, in this way electric devices are close to the electronic box and wirings and connections are easyer to implement. In the next subsection electric components are described one by one, clarifying both their technical features and functions.

2.3.1 Lithium Batteries

Robot locomotion, electric subsystem and electronics are fed by means of high capacity lithium batteries, which are placed in the central part of the robot frame (see figure 2.5). Most important parameters of these devices are summarized in

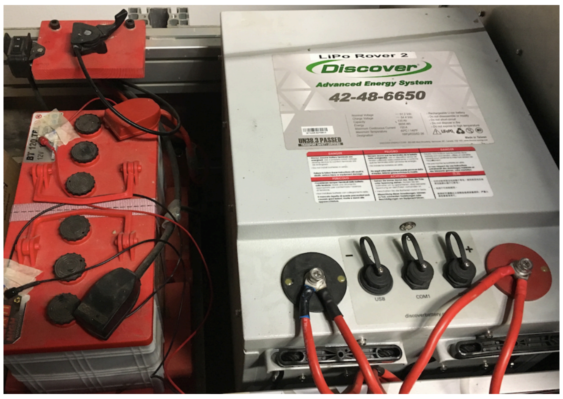


Figure 2.5: Detail of lithium and lead batteries installed on the robot.

table 2.1. In the current configuration, the robot is equipped by just one battery, in such a way to test the system in this conditions. After a first experiment session, at least one battery will be added, in order to get an higher autonomy and giving the possibility to feed also the tools and remove the endothermic engine, towards a full electric solution (see chapter 6). From the battery, two parallel connections (two

Parameter	Value
Nominal voltage	51.2 v
Charge voltage	54.4 v
Capacity	130 Ah
Energy	6656 Wh
Max. cont. current	130 A

Table 2.1: Lithium batteries parameters

negatives and positive poles, for an amount of four cables) goes towards the rear of the vehicle to the aluminium plate, giving 48 v power to motor drivers contactors, to DC-DC converters, and finally to the ignition key. Turning the latter, drivers logic circuit is also fed (see subsection 2.4.2 for further details).

2.3.2 Drivers

This devices are placed on the aluminum plate and have the aim of providing power to the motors, depending on the signals received from the robot controller through the CAN bus connections. Drivers, each one dedicated to a single motor, are fed, in terms of battery power, through the contactors. Figure 2.6 shows a detail of the aluminium plate and components installed on it, during robot construction. In the left upper part is visible the $24\,v$ to $48\,v$ DC-DC, while the other one is already missing, in the central part there are drivers and winch contactors, and in the lower part the two drivers are visible behind wirings. Most important

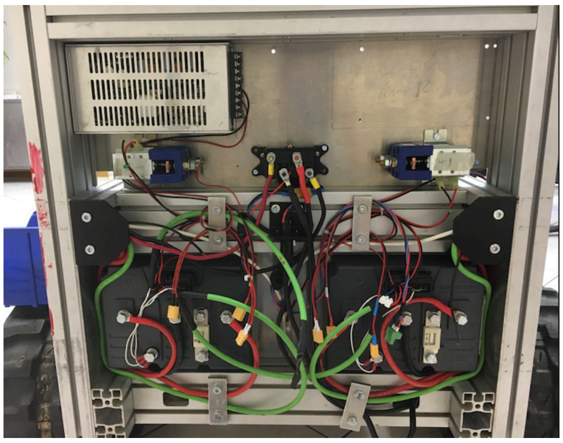


Figure 2.6: Aluminium plate on which drivers, converters and contactors are installed.

parameters of this devices are summarized in table 2.2. In case of emergency, the

Parameter	Value
Model	Sevcon Size 4
Max. cont. current	180 A (for 60 min.)
Spike current	450 A (for 2 min.)

Table 2.2: Drivers parameters

latter can always be switched of from the emergency button (see section 2.4.2), while in nominal situation they are managed by the drivers. Drivers communicate both with the onboard controller, by means of CAN bus and USB-CAN interface (see subsection 2.4.1), and with motors, in terms of power feeding them, and feedback, consisting in motor temperature and encoder signals.

2.3.3 Motors

Motors provide mechanical power to the robot tracks. They are fed, in terms of electric power, by the drivers, and gives back temperature sensor and encoder feedback, each one to the corresponding controller. Motors most important parameters are summarized in table 2.3.

Parameter	Value
Model	Metalrota PSFBL72
Encoders type	$sin - cos$
Rated speed	$3000\,rpm$
Nominal power	$1500\,W$ (for 60 min.)
Nominal torque	$4.77\,Nm$
Nominal current	$45\,A$ rms
Max torque	$16.8\,Nm$ (for 25 min.)
Max current	$135\,A$ (for 5 min.)

Table 2.3: Drivers parameters

2.3.4 Brakes

Brakes are foundamental for safety, in order to avoid risks when operating in robot proximity or to stop it in case of emergency. Brakes are fed at $24\,v$ power, coming from the $48\,v$ to $24\,v$ DC-DC and controlled by ignition key and emergency button (see section 2.4.2). Of course, when the circuit, namely a coil, is switched off, brakes are active, in the sense that they keep the robot tracks *stuck*, and viceversa if the circuit is *alive*.

2.3.5 DC-DC Converters

A couple of DC-DC converters is placed on the aluminium plate (see figure 2.6), with the aim of providing proper voltage power to sensors, brakes and electronics, which works at a lower voltage level than the batteries one. In particular, a $48\,v$ to $24\,v$ converter feed brakes and microcontrollers, through a small $24\,v$ to $5\,v$ device, while a $48\,v$ to $12\,v$ one provides power to sensors and onboard computer. The first converter gives a maximum current of $8.3\,A$, at $24\,v$, while the second can provide $16\,A$, of course at $12\,v$.

2.3.6 Lead Battery

Close to the lithium battery, also a lead one is installed (see left side of figure 2.5), with the aim of:

- Switch on the endothermic engine (see subsection 2.2.3)
- Feed the winch, for tools height regulation (2.2.3)

This device is actually a common *car* lead battery, working at a nominal voltage of $12\,v$ and $120\,Ah$ stored energy.

2.3.7 Winch Circuit

The winch mounted at the rear of the vehicle works at $12\,v$ and absorbs, as maximum value, a current of $130\,A$. For this reason, it has not been possible to feed this device through a DC-DC, exploiting lithium batteries already installed onboard, and in the current robot configuration it is connected to the lead battery. The winch is managed by means of a contactor, which makes it lift or lower based on its control switch position (visible in the upper left corner of figure 2.5). Actually, the winch can be manually operated by means of a switch placed close to the lead battery, but also remotely controlled. In this case, joypad commands (see subsection 2.4.4) triggers two relays of the electronic box (through NUC and arduino Mini Pro microcontroller), which in turns act on the winch contactor. This funtion is useful for adapting the tool height while manually driving the robot, without breaks or need to reach the manual switch.

2.4 Electronic Box and Sensors

This section describes the third subsystem composing the robot structure, i.e. electronics and onboard sensors. In the next subsection (2.4.1) the electronic box is presented, which contains all components providing for robot control and sensors reading. Finally, in subsection 2.4.3, sensors installed on the robot are described.

2.4.1 Electronic Box

The *electronic box* has the aim of controlling the robot and contains all the components necessary for providing this service. The most important box functions are summarized in the list below.

- Manage low voltage power, i.e. $12\,v$ and $24\,v$, which must be provided to the different electronic components.
- Sensor management, in terms of power supply and reading
- Brakes management
- Run onboard computer applications, i.e. user interface and autonomous navigation algorithm
- Deal with the manual controller
- Communicate with motors drivers, sending references and receiving feedback
- Guarantee a certain safety level for humans dealing with the robot, by means of manual buttons

Figure 2.7 shows a view of the panels and components of the electronic box. The lower panel is quite hidden, while on the upper one are visible an ethernet switch, the LIDAR board, an Arduino Mini Pro and a Mega 2560 microcontrollers. Here

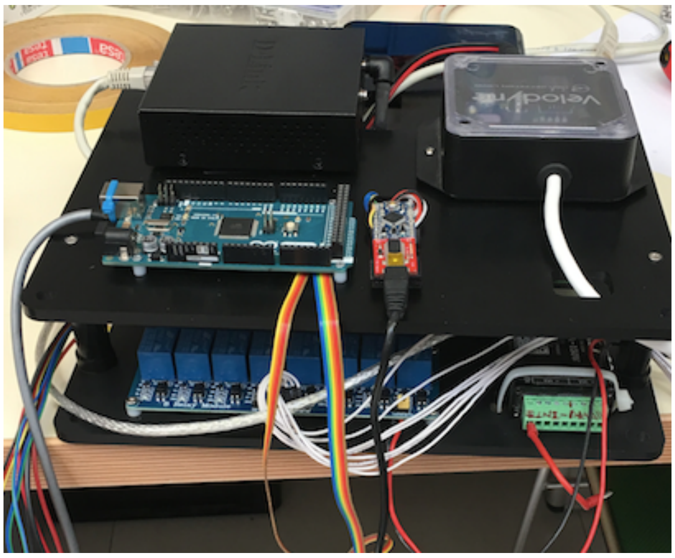


Figure 2.7: View of the whole content of the electronic box.

below the electronic box components are presented and described, focusing on their function, in such a way to give a complete discussion about both electronic devices contained in the box and related service they supply.

Intel NUC

This is the onboard controller of the robot and runs all the code which provides user interfacement and autonomous navigation capabilities. For thi function, an Intel NUC has been chosen, a compact mini-PC highly customizable that ensures good computational power in a quite limited footprint. For what concerns the application, among the countless models available on the market it has been chosen the *D54250WYKH*, this choice is motivated by the fact that, at the beginning of the project it was one of the best option with respect to the price/performance ratio. The key advantage of having a PC onboard is the possibility of having

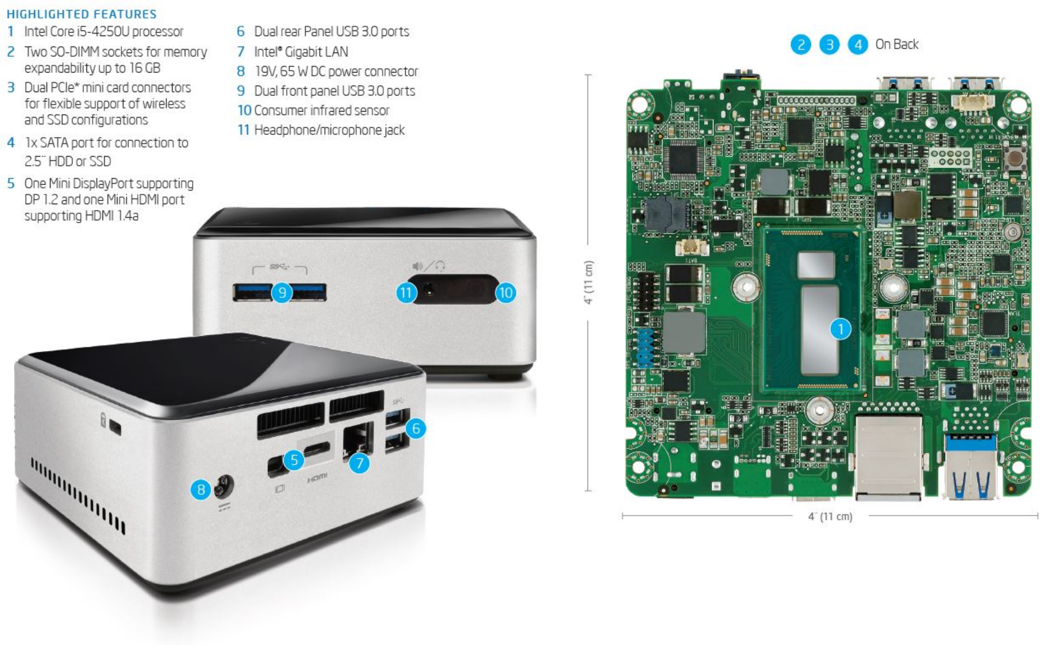

HIGHLIGHTED FEATURES
1 Intel Core i5-4250U processor
2 Two SO-DIMM sockets for memory expandability up to 16 GB
3 Dual PCIe* mini card connectors for flexible support of wireless and SSD configurations
4 1x SATA port for connection to 2.5" HDD or SSD
5 One Mini DisplayPort supporting DP 1.2 and one Mini HDMI port supporting HDMI 1.4a
6 Dual rear Panel USB 3.0 ports
7 Intel® Gigabit LAN
8 19V, 65 W DC power connector
9 Dual front panel USB 3.0 ports
10 Consumer infrared sensor
11 Headphone/microphone jack
2 3 4 On Back
4" (11 cm)
4" (11 cm)

Figure 2.8: Intel NUC model D54250WYKH, external and inner view.

Operating Systems running onboard, endowing the system with an extremely high versatility. Furthermore, by means of virtual machines it's possible to use different OSs. Indeed, in the developed software structure the operating systems installed onboard are:

- Windows 7, for user interfacement features
- Linux Ubuntu, for autonomous navigation implementation

In chapter 4 a detailed view of the software structure inside the onboard pc is given, but it is important to highlight some aspects also in this section, in order

to better understand the presence of some hardware items.
The part running Windows is devoted to interface the user with the machine and to receive commands and feedbacks from the navigation algorithm and in turn send them to the motor drivers.
The Windows-running machine is connected, via USB, to the following devices:

- Arduino Pro Mini
- IXXAT Can Interface

On the other hand, software running under the Ubuntu Operating System permits to communicate with sensors. Therefore the Linux Ubuntu machine is connected, both via USB and Ethernet port, to:

- Arduino Mega 2560 (USB)
- Velodyne LIDAR PUCK VLP-16 (Ethernet)
- Trimble R8s GNSS System (USB)

As can be seen in the lists above, the two operating systems are connected to their own Arduino board. This choice has been taken in order to create an organized communication structure, in which an Arduino, plugged to the Windows part, is aimed to control peripheral devices or functionalities directly from HMI commands, and the other one, connected to Ubuntu, with the task of reading data from sensors. Of course the NUC is connected to a power source, at $12\,v$, and to a screen, a 10,1 inches screen with touch-screen features. This characteristic ensures an higher level of convenience than using mouse/keyboard away from the desktop and make it easy to deal with the pc even during experiments within orchards.

Arduino Pro Mini

Arduino Pro Mini microcontroller, showed in figure 2.9, communicates with the robot HMI, receiving user commands and sending him some feedback. In particular, it controls:

- Array of relays, in order to provide power and activate sensors or other components
- Electronic box cooling fans

On the other side, it provides feedback about relays conditions, indicating if a device is active or not, and temperature inside the electronic box. This latter information is also used, by the microcontroller, for regulating the cooling fans and keeping it within a safety threshold. These feedback are proposed to the user in the first screen of the robot HMI (see figure 4.2).

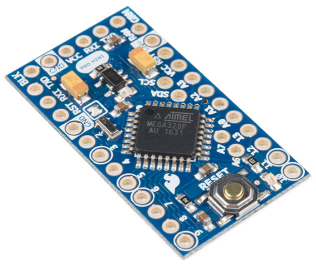


Figure 2.9: Arduino Pro Mini microcontroller.

Relays array

This Arduino module is composed by an array of eight relays whose coils are opto-isolated (see figure 2.10). Therefore, it can be directly connected to the Arduino board without any worry about safety. In order to avoid malfunctions or saturations, coils activation currents are provided by the power supply, at a $5\,v$ level, and do not come from the Arduino pins, whose capacity is limited to about $100\,mA$.

This module is used to supply power to several devices, which are not required

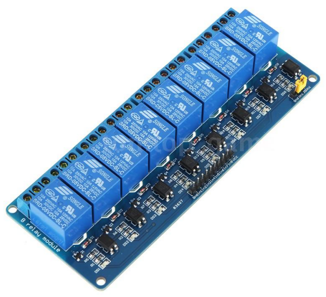


Figure 2.10: Arduino relay module.

to keep enabled if not used:

- *Ethernet Switch*, used to increase the number of possible Ethernet connections, since the NUC has just one RJ45 port, and it needs to be connected to the LIDAR, to the virtual machine and to the internet network.

- *LIDAR VLP-16*, which is foundamental for autonomus navigation experiments, but can be switched off, for saving batteries, during debug or when the robot is manually driven.
- *Motor Brakes*, which needs to be armed and disarmed depending whether the robot has to move or not. Brakes managements through the microcontroller gives the possibility to remotely activate or deactivate brakes, from the Arduino or the HMI.

Cooling Fans and Temperature Sensor

Within the electronic box there are many components (mainly the NUC mini PC) that emits a lot of heat, which is not healthy for the electronics functioning. Therefore, a system for controlling the box temperature has been designed. A *DTH22* temperature sensor (see right side of figure 2.11) is connected to the

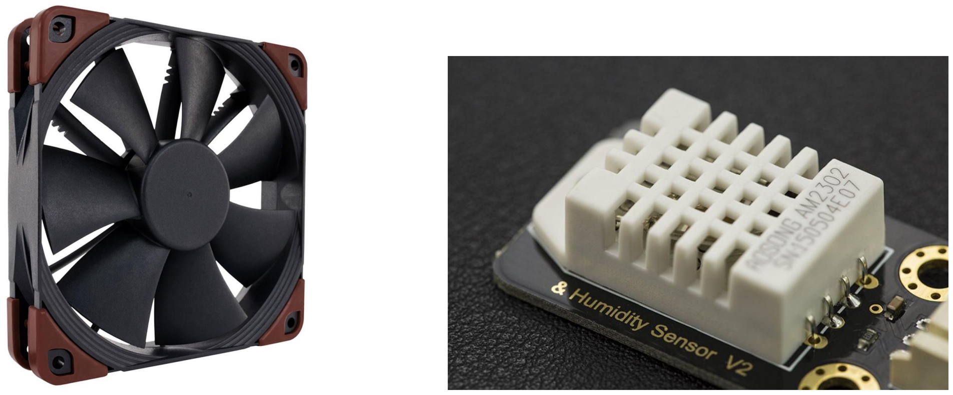


Figure 2.11: Cooling system of the electronic box, composed by cooling fans (left side) and temperature sensor (right side)

Arduino Pro Mini. On the basis of sensor data, the microcontroller regulates the spinning velocity of two cooling fans (left side of figure 2.11), in order to keep the temperature box within a safety range. The two cooling fans are attached onto the box in such a way to have one blowing inside cooler air, and the other one pushing out the hotter air. Moreover, powder filters has been mounted both onto the inlet fan, in order to avoid aspiration of undesired and potentially dangerous elements, and onto the outlet fan, to avoid the entrance of particles when the fan is standing still).

Arduino Mega 2650

The Arduino Mega microcontroller (see figure 2.12) is connected to the Ubuntu operating system and reads data coming from low-cost GPS, with a built-in magnetometer, and exchanges data through a serial communication channel with another Arduino Pro Mini. The latter is directly connected with an Inertial Measurement

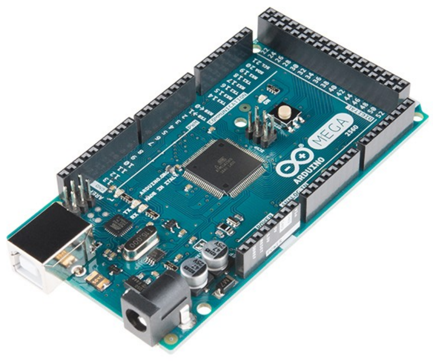


Figure 2.12: Arduino Mega 2650.

Unit (IMU) through an IIC (I^2C) bus. This solution could appear quite strage, but it is motivated by two factors:

- The IMU has been placed really close to the center of gravity, in order to make it easy to read and interpret its measurements, even if it means to mount it far from the electronic box, and close to batteries and power cables, i.e. expose it to high electromagnetic fields.
- The IMU interface consists in a I^2C bus, which is really sensitive to *EM* disturbances.

For this reason, the IMU has been connected to an Arduino Mini, placed really close to it, in order to keep I^2C wires as short as possible and be less sensitive to environment disturbances. Sensor measurents are sent via serial communication, which is more robust, to the Arduino Mega board, which transmit these data to the onboard pc.

IXXAT Can Interface

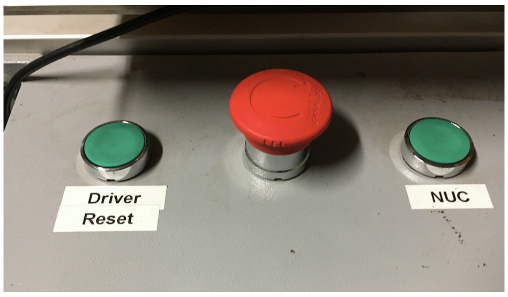
In section 2.3 motors electric drives has been described, discussing about their *CAN* communication protocol, used for comminicating with external devices. Of course an interface is needed, in order to provide communication between onboard pc and CAN side. The chosen *USB-CAN* interface (see figure 2.13) has the burden of converting signals from an USB port that is plugged into the NUC and a 9-pin connector which implements the CAN-bus, and viceversa.

Figure 2.13: IXXAT USB-CAN Interface.

2.4.2 Buttons Panel

On the external part of the electronic box are installed some manual buttons or switches, which provide to the system additional functionalities. Figure 2.14 shows a view of top and right side of the box (at left and right sides of the picture, respectively), on which those items are placed. Here below these buttons are listed and described, focusing on the features or safety function they provide

- **Ignition key**
  Placed on the right side of the electronic box, has the function of switching on, or off, the whole electronic subsystem of the robot. In particular, the key interrupts:
  - 12 v power, coming from the 48 v to 12 v *DC-DC*, which supply the onboard computer, sensors and microcontrollers.



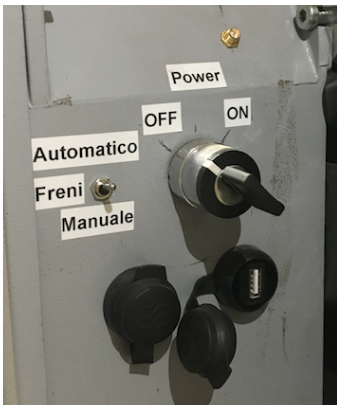


Figure 2.14: Detail of the electric box external buttons and switches.

- 24 v power, which comes from the 48 v to 24 v *DC-DC* and supply brakes and cooling fan.
- 48 v power, coming from the battery and supplying the logic part of the motors drivers.

- **Driver Reset**

  48 v power supplying the logic of motor drivers goes through the ignition key, as already stated, and the *normally closed* Driver Reset button. Pressing it the drivers are switched off for a while, implementing a driver reset function, useful in case of problems.

- **NUC Switch ON**

  Through this *normally open* button the onboard computer can be easily switched on, without opening the electronic box.

- **Emergency button**

  In case of emergency, the user can push this button, placed on top of the electronic box. This button interrupts:

  - Brakes power supply (24 v), in series with the ignition key
  - Contactors power supply (48 v), through which battery power reaches the motors drivers.

  In this way, as the button is pushed, the tracks are stucked and the motors do not receive battery power anymore.

- **Brakes Switch**
  This switch let the braking system behave in two different ways:
  - *Automatic.* Brakes power comes through ignition key and it its managed by the first relay of the array driven by the Arduino Mini Pro, letting the user activate or deactivate brakes from remote controller or HMI.
  - *Manual.* Make it possible to directly supply brakes, bypassing the ignition key and the Arduino microcontroller. This feature is really useful in case of emergency, in order to move the robot even if there are problems in the electronics functioning.

2.4.3 Sensors

This subsection has the aim of presenting the sensors installed onboard and describing their most important function and parameters. Figure 2.15 shows the

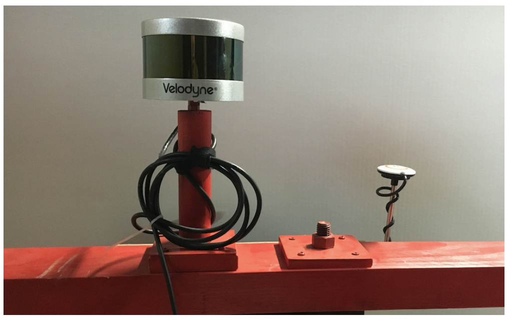


Figure 2.15: Detail of laser scanner and gps installed on the sensors bar.

LIDAR sensor and low-cost GPS receiver installed on the robot, on a dedicated bar. In the central part there is an empty space designed for the RTK-GNSS receiver, which, in the current configuration, is not installed yet.

IMU

An interoduced in subsection 2.4.1, an IMU sensor (see section A.3) is mounted in the center of gravity of the robot. The model is the *HMC5883L* (see figure 2.16), an IMU composed of:
- Magnetometer, which returns the three dimensional earth magnetic field
- Gyroscope, providing rotation velocities along the *body* frame axes

- Accelerometer, giving accelerations along the *body* axes

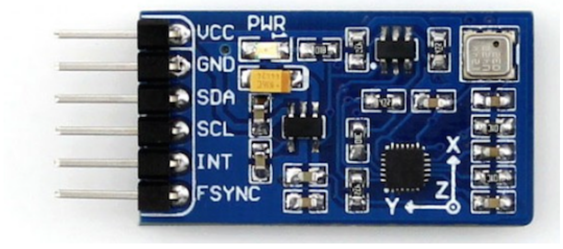


Figure 2.16: 10 degrees of freedom IMU

Data from IMU is acquired by the control angorithm and used for robots position and attitude estimation.

GPS Receiver

A low cost GPS receiver (see section A.2) is installed onboard, with the aim of providing the absolute robot position. Integrated in this device there is also a magnetometer, which data is collected by the Arduino Mega 2560. Even if there is

Figure 2.17: Ublox GPS receiver

a magnetometer also inside the IMU, that device is subject to high disturbances, due to its proximity to power devices, motors, and the robot frame. For this reason, also data from a second magnetometer is collected and used for calculating the robot heading. The device is a ksjdfn, developed by U-Blox (see figure 2.17). Readings from the Ublox GPS receiver are used to roughly localize the vehicle while acting outside orchards and the robot can not rely on environment features for localization anymore.

Velodyne LiDAR PUCK VLP-16

On the vehicle is also mounted a 3D laser scanner sensor (see section A.5), which provides environment perception and makes it possible to detect rows of trees, obstacles, branches or other orchard features. The model is a Velodyne LiDAR PUCK VLP-16. Figure 2.18 shows the LIDAR device, on left side, and an image of data coming from the sensor, on right side, giving an idea of its performances.

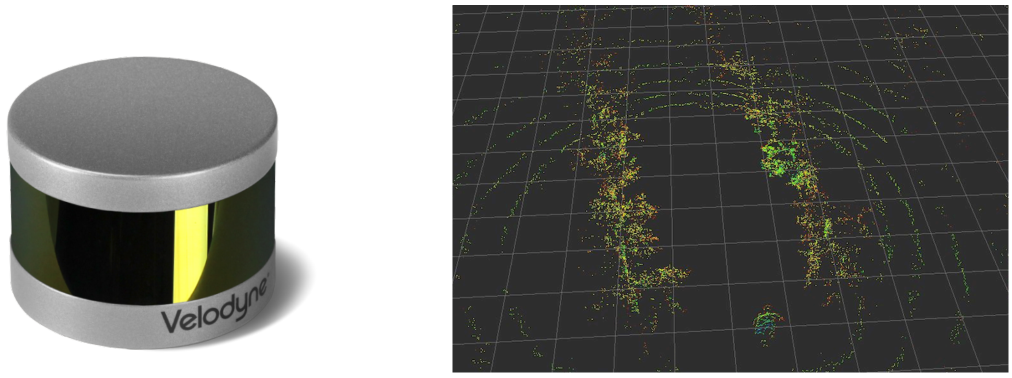


Figure 2.18: Velodyne LIDAR VLP-16 (left) and sensor data collected within rows of a plum field (right)

Trimble R8s GNSS Receiver

The robot is designed, in terms of sensors supports, in order to install onboard a Trimble R8s GNSS (see figure 2.19), which is an high precision RTK-GNSS satellite receiver (see section A.2). Currently, due to lack of time, this device has not been used yet. One of the first future developments is the substitution of the low cost U-Blox GPS with the Trimble one, in order to increase the robot global position precision and augment its navigation capabilities outside orchard structures.

2.4.4 External Devices

This final subsection completes the electronic box presentation, describes the external devices used by the user for actively interacting with the onboard computer. These external peripherals are:

Figure 2.19: Trimble R8s GNSS System.

- Touch screen monitor, by which the user can see computer outputs and interact with running applications
- Wireless mouse and keyboard, used for launching the system and interact with it
- Joypad, by which the user can manually drive the robot and give it some commands, by means of the available buttons

Since the onboard controller is actually a normal computer, the first two components functions are quite intuitive and do not need further explanations. Differently, the *joypad* needs an additional description, which is given here below.

Joypad

The joypad (see figure 2.20) is used for manually driving the robot and also to give it different inputs. This peripheral is directly read by the Windows operating system, i.e. by the HMI. The actions associated to its buttons depends on the operative mode (see section 4.4) selected by the user, except:

- Brake activation/deactivation, triggered by means of the *A* button
- Operative mode change, given through the *start* button

In *manual* mode, the most important commands are:

- Manual speed references, which can be given by means to the sticks.
- Winch lift or lower commands, given by means of left and right *shoulder*

Differently, in *auto* mode, the user can give the following inputs (see section 4.4 for commands explanation or other details):

- increase/decrease robot speed, by means of left and right *shoulder*

Figure 2.20: Logitech joypad

- Start mission, using right *thumb*
- Define home position, through the left *thumb*
- Trigger mission pause, by means of Y button
- Stop the mission, using X button

2.5 Conclusions

In this chapter the developed agricultural robotic platform has been presented. Implementation and design choises have been discussed, and the different subsystems composing the robot has been described in details. Mechanical structure and electric components features are presented in the first parts, while the electronics, sensor suite and controllers are described in the last part. In the next chapter the second step of the research work is proposed, dealing with the autonomous navigation system development, focusing both on localization and control problems.

Chapter 3

Developed Navigation Tecniques

This chapter describes the theoretical results about navigation in agriculture developed during the research work, focusing on environment structure exploitation both for robot localization and control.

3.1 Introduction

In this chapter, the navigation techniques developed and then tested during outdoor experiments are be presented. First of all the idea is to deepen

- the concept of navigation, clarifying the goals of this research project and the steps for reaching them
- the difference between *open field* conditions and *row structure*, and on how the latter have been exploited in the proposed localization and control algorithms (see 3.4)

Then, in the central part of the chapter (3.5 and 3.6), obtained conceptual results and developed navigation algorithms will be presented. Finally, the proposed robot control law for trajectory tracking is presented in section 3.7.

3.2 Reference Frames

The aim of this section is just to present the reference frames used in this work and useful for robot position, motion, and navigation description. In order to represent robot position on the earth's surface, an *inertial* frame is introduced

[23]. Since this application deals with low speeds and distances, compared with aeronautical ones, a *NED* reference frame is assumed as inertial (see figure 3.1).

This frame is defined by its origin O_i and three axes x_i, pointing *North*, *East*

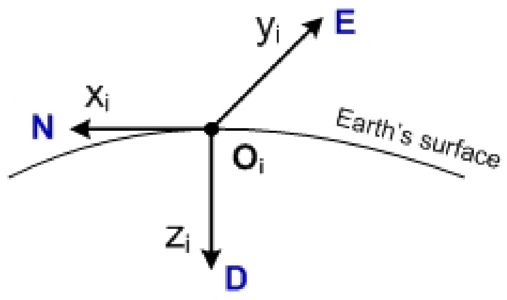


Figure 3.1: Representation of the adopted *pseudo-inertial* NED frame

and *Down* (i.e. towards the center of the earth), respectively. NED position can be directly computed from the GPS coordinates, while the origin is defined at the mission launch. This *home* position does not necesssarily correspond to the initial position of the robot, which changes time to time. Rather, it can be selected by the user in the most favorable location, since, if this point does not change with time, it is possible to compare trajectories between missions carried out in different times. Of course also a *body* frame is useful, for representing robot

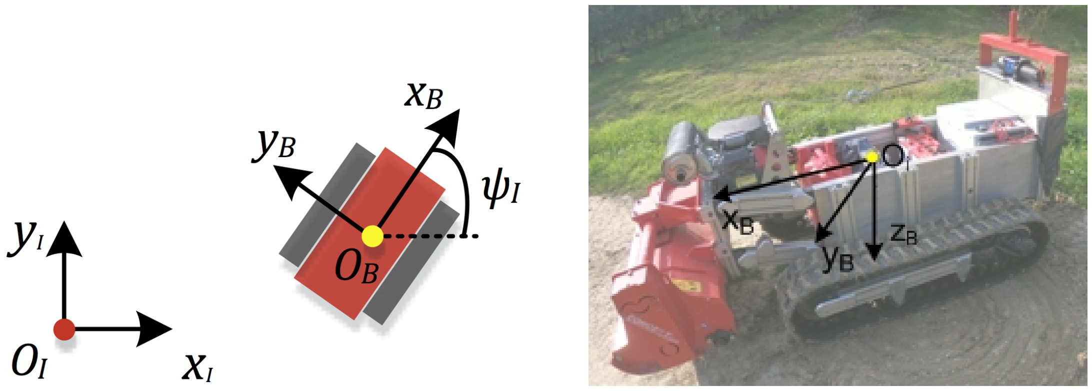


Figure 3.2: Body frame representations, respect to the inertial frame (left) and to the real robot structure (right)

position and orientation. This frame is defined by the origin O_B and the three axes x_B, corresponding to the longitudinal axes and pointing forward, y_B, transverse axes pointing to the left, and z_B, perpendicular to the first ones and pointing

downward. The robot orientation, i.e. the relative orientation between body and inertial axes, is represented by the Euler angles φ_I (roll), θ_I (pitch), ψ_I (yaw) [4]. A 2D representation of the body reference frame is proposed in figure 3.2, where angle ψ_I represents the orientation of the x_B axis respect to the x_I one, i.e.the North.
Finally, in order to take into account LIDAR data, it is important to refer the *pointcloud* position respect to the robot one. For this reason, also a *laser* reference frame is defined, with origin O_l, and axes x_l, y_l, z_l. O_l represents, of course, the LIDAR position, while the axes have the same orientation of the *body* ones (see figure 3.3). The difference is given by a spatial translation, between the robot

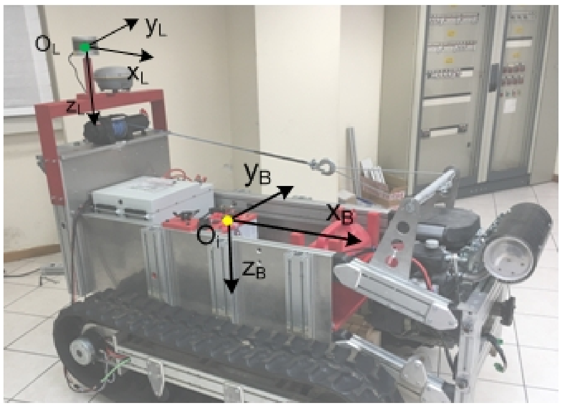


Figure 3.3: Representation of the *laser* frame, respect to the *body* frame

center of gravity and the sensor position, which is placed at the back of the vehicle for safety reasons.

3.3 Navigation

Navigation concept means the capacity of moving within a certain environment. For reaching this goal, the platform needs mainly two features:

- *Localization*, respect to the surrounding environment or some reference point, i.e. know where it is.
- *Control*, which means calculating the correct control actions and drive motors, in such a way to reach the desired position or configuration.

The environment plays an important rule in both this steps, since localization depends on its configuration and features, while control depends on ground conditions, frictions, slopes, and so on. The navigation environment, which encompasses the whole context, problems and difficulties related to this research work, has already been partially clarified in chapter 1. In particular, has been described the potential of autonomous navigation in farming environment, as a common feature of any autonomous agricultural robot (see subsection 1.2.1). The context has therefore focused from the application of robotics and automation in the agricultural field up to the specific task of autonomous outdoor navigation. This work environment was then described (see subsection 1.2.2), highlighting its intrinsic characteristics and difficulties it involves. Furthermore, the structure of the navigation environment was discussed, specifying the differences between *open field* and *structured* or semi-structured field (see subsection 1.2.3) and highlight the different sensor suites used in the two cases. In open field, where there are no references to exploit, on one hand, nor possible elements limiting satellite coverage, on the other, knowledge of the *absolute* position, such as that provided by GPS, is fundamental. Differently, in a structured environment, the perception of the surrounding environment is fundamental, both for precise navigation and collision aviodance, and for localization within the structure. For this purpose, nowadays, high-performance proximity sensors such as cameras or LIDARs are employed. So, starting from the concept of outdoor navigation, the problem has focused on navigation in structured agricutural environment.
Before starting the description of the research results, it's important to define goals and context. The context has been presented during chapter 1, and consists in agricultural environment organized in row structures. The goal is autonomous navigation, i.e. to make a robot localize and control itself within the structure, performing a certain task. Thanks to the organization in rows, the robot just needs *relative* localization respect to the structure, and not an absolute one. For this reason, environment perception is foundamental and a 3D LIDAR sensor has been chosen as a basis for localization.

3.4 Structured Environment and Internal Model

In the previously delineated context, it is extremely important to take advantage from the preliminary knowledge about the environment, namely on the kind of trajectories expected to the robot, in order to deal with the many challenging aspects behind the fact of operating in the outdoor agriculture domain. The idea is to perform a perception and control algorithm based on an **internal model** of the priori knowledge about the environment. Namely, the perception system and the control algorithm are co-designed in order to include the environment geometric parameters.
The intuition is better framed thinking of a human-interest story, which is navigating in a known dark room (for instance to get the bathroom from the bedroom in the middle of the night without switching on the light). The bedroom is known in the main nominal objects/obstacles, such as bed, chairs, wardrobe, dressers, carpet, and their nominal location in the room. What is also known is the rough *initial position* in relation to one object (the bed). The perception system is mainly given by tactile information that are wisely managed and adapted by the moving person according to the priori known geometry of the touched object and inertial information. Finer tactile actions are typically actuated if the pre-knowledge of the environment suggests that the human is close to the corner, while coarser perception is enough otherwise. Similarly, the applied control law is influenced by the priori knowledge of the environment, with geometric trajectories and time laws followed by the human that are strongly affected by the pre-knowledge of the room. For instance, when following the straight edge of the bed, the control law governing the human implicitly forces her/him to follow a straight geometric trajectory. Overall, the applied control strategy is influenced by the perception goal (time-laws are tuned according to fine/coarse perception objectives) and, on the other hand, the control law is necessarily driven by the outcomes of the perception system. In other words, *co-design* of perception system and control strategy is characterizing the described scene, with an internal model of the environment implicitly embedded in the whole system.
Returning now to the agricultural context, the idea could be to expoit the knowledge about orchard geometry structure, both for *localization* and *control.* About localization, during a mission the robot could be asked to travel different rows or

travel from the current to a new one. Therefore, the robot needs both to localize within rows and also when the row extremity has been crossed, i.e. outside the row structure. For this reason, two different algorithms have been developed, for position estimation both within and outside orchard rows (see sections 3.5 and 3.6). A proper control law is associated to each of the localization algorithm, in order to control the robot exploiting the environment based internal model (section 3.7).

3.5 Hough Transform Upgraded

When the robot is working within the orchard structure, the most important features of the surrounding environment are the rows between which it is moving. Indeed, during agricultural works, the tractor is usually driven through the rows imposing a constant translation speed, based on the current work to carry out, and controlling the steering in order to keep the vehicle at a desired later distance from the trees, or simply at the middle of the row. Basically, the proposed navigation technique replicates this kind of approach. Estimation of the trees position, i.e. localization respect to the lateral rows (see figure 3.4), becomes the most important issue. The idea is to model the rows as lines, trying to extract these features

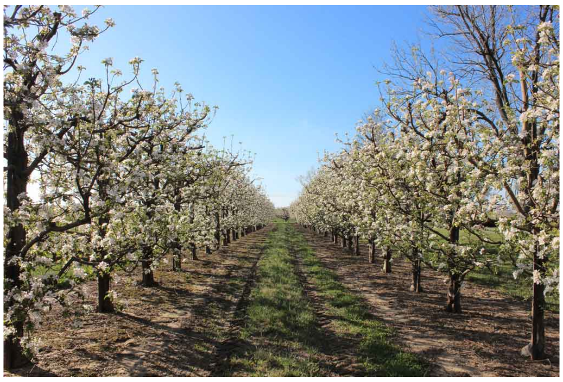


Figure 3.4: Example of view within rows

from LIDAR data. In figure 3.5 an example of LIDAR data collected in a plum field is proposed. In the picture two kinds of *linear features* are visible, which are actually the rows of trees, plus circles, representing the points in which the laser beams encounters the terrain, and finally a human operator, which was driving the robot during data collection.

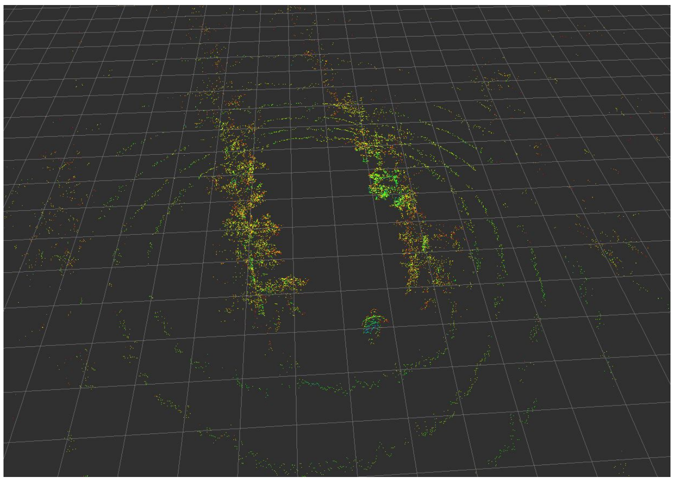


Figure 3.5: LIDAR data collected within two rows in a plum field

Of course, in order to reduce the computational weight, 3D laser data is filtered before the following feature extraction. In particular, since the row lines are identified by tree trunks and branches, an height range has been defined between $0.8\,m$ and $1.8\,m$. In this way, ground and higher branches are not taken into account, providing a faster computation and improving reliability.

Among different model extraction algorithms available in literature and employed for navigation and computer vision (see chapter B), the **Hough** transform (see section B.3) has been chosen. Since the robot acts between two lines of trees, the algorithm can estimate the position of both the rows, obtaining a redundant measurement. The *standard* algorithm estimates those linear features in terms of distance and relative orientation respect to the robot. In figure 3.6 a scheme is presented, with green shaded lines representing the orchard rows, red dot indicating the robot center of gravity and (x_B, y_B) the vehicle body frame axes (see section 3.2). Moreover, (d_l, θ_l) and (d_r, θ_r) indicates left and right lines distances and relative angles respect to the body frame, result of the estimation process. Finally, θ represents the result of θ_l and θ_r fusion, merged with different weights based on the reliability of left and right estimations, and used in the robot steering control algorithm. Figure 3.7 shows the research grid of the Hough transform algorithm, where abscissa and ordinate represent the angle θ and distance d, respectively. The angle goes from 0 to π, while the distance ranges from d_{min} to d_max, which are algorithm parameters assigned from the outside. The blue rectangular rep-

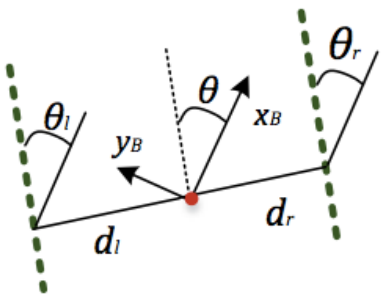


Figure 3.6: Distances and angles returned by the Hough transform algorithm

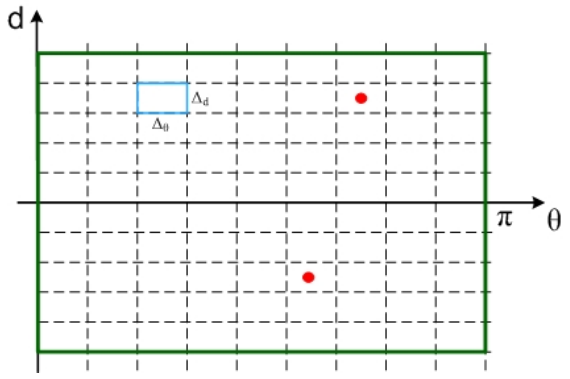


Figure 3.7: Research grid of the Hough transform algorithm

resents the grid *discretization*, defined by the angle and distance quantum, Δ_θ and Δ_d, on which depend result precision and of course computation time. The green rectangular defines the research *area* which, in the stardard version of the algorithm, covers the whole range of the research variable, while the red dots represents two possible solutions of the algorithm. In the represented situation, the robot is more or less in the middle of the rows (the two distance value are quite close) and the algorithm extracts two row lines which are pretty much parallel. As already introduced, the novelty is the inclusion in the algorithm of an **internal model** based on geometry knowledge about the environment. Since the orchard geometric parameters are available from the *Robot Control* (see chapter C), the platform knows

- Orchard rows are parallel
- Row width
- Row structure is fixed

For this reason the algorithm researches for *parallel* lines, imposing that the mutual *distance* corresponds to the row width. This concept is implemented, at each iteration, once the vote matrix is full and the Hough transform solutions have to be extracted (see B.3 and figure B.7). Defined a vector of possible solutions $s[i]$ (where $i = 2$, since the algorithm looks for two lines), this process can be summarized as follows:

1. The cells of the vote matrix, one by one, are analyzed inside a nested loop, scanning both *distances* and *angles* dimensions.
2. At the first iteration, the solution represented by the current cell is stored in $s[0]$, as possible solutions, in order to initialize the procedure.
3. Then, another cell of the matrix is scanned. If the vote of the new cell is higher than the solution stored $s[0]$, the latter is substituted. Now, if the solution in the second position $s[1]$ satisfies parallelism and distance respect to the new first solution $s[0]$, it is kept, otherwise rejected.
4. Differently, if the scanned cell has an higher vote respect to the one stored in $s[1]$, and the parallelism and distance conditions are satisfied, the new solution is better than the latter, and substitutes it.
5. The procedure above terminates once the vote matrix has been completely scanned.

Moreover, since the structure is fixed, the research grid is also reduced, asking the algorithm to research for lines *close* to the previous iteration solution. Figure 3.8 shows the research grid of the proposed localization algorithm, consisting in the Hough transform enriched with the orchard internal model. The difference respect to the original version of the algorithm (see figure 3.7), is the reduction of the research grid. The orange dots represent the previous iteration solutions, around which the research areas, represented by the green rectangulars, are calculated. In this way, the algorithm frequency can be reduced and it can run faster. The principle is that at the beginning the robot have to localize, researching for solutions into the complete grid, so including all the possibilities. Then, thanks to an asinmptotically stable control law, the robot will reach the desired position, at a certain distance from the trees and parallel to the lines. Therefore, once the robot has localized itself, the research grid can be progressively reduced around the previous estimated lines. Moreover, the loop frequency can be reduced, since once the

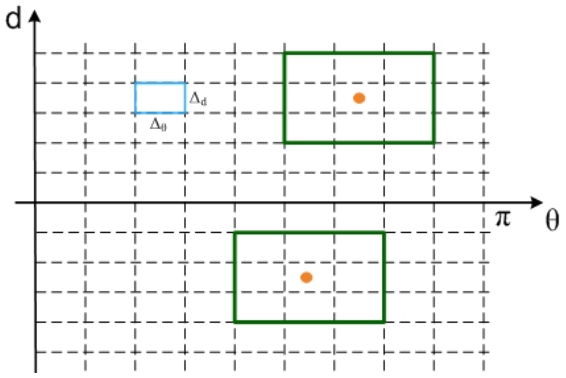


Figure 3.8: Research grid of the proposed localization algorithm within rows

robot is aligned can navigate using encoders and IMU, and just sometimes take a look at the surrounding environment, in order to calibrate the position estimation and correct drifts. So, if the trajectory tracking error is small, the robot can move faster and sample the surroundings just occasionally, while if the robot is reaching the row extremity, the algorithm encounters difficulties or the localization is not reliable, the robot slows down its speed and increase the environment perception, in order to correct the wrong situation and lead the robot on the desired trajectory.

3.6 N-Hough Transform Upgraded

Once the robot travelled a row, it needs to move to the next one, in order to continue its task. In this situation the vehicle is outside the rows structure (see 3.9) and needs to deal with a different localization problem. It does not see lateral

Figure 3.9: View of orchard rows from the outside of the structure

trees anymore, but many rows extremities and part the tree lines. In agriculture, this kind of maneuver is carried out, similarly to the previous case, keeping a low translation speed, and controlling the steering in order to describe a curvilinear trajectory which leads the tractor to the beginning of the next row. Trying to replicate that approach, the idea is to ask the robot some kind of semicircular trajectory, which can be tracked by means of both feedforward and feedback actions. The first is useful in order to provide to the robot a precalculated trajectory and quite easy to calculate, since the row structure is known. The problem is that IMU and encoders are affected by drift or slippage on the terrain and does not take into account obstacles or overhanging branches. For this reason a feedback action is needed, in order to define a reliable navigation system. Therefore, localization has to be carried out also in this situation, estimating the robot postion respect to the rows of trees from LIDAR data. In figure 3.10 is showed a screen of LIDAR

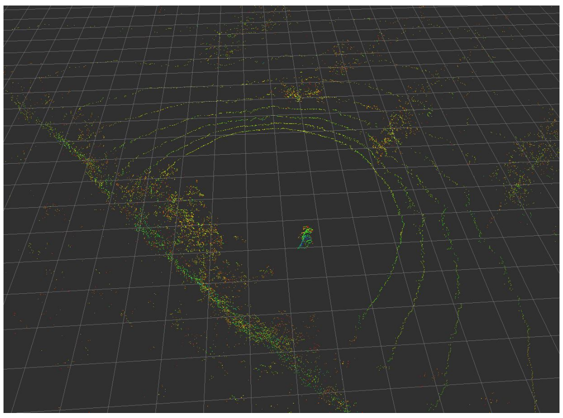


Figure 3.10: LIDAR data collected outside the rows of a plum field

data collected during experimental tests carried out in a plum field. On the right can be seen some kind of linear features, corresponding to the rows of trees, and another line, on the left, which corresponds to a feature not belonging to the orchard structure. In this case the lateral distance information is not enough for navigation and the robot needs no know:

- distance and orientation respect to row *lines*
- distance from the *beginning* of the rows

Since outside the structure many rows are visible, the developed algorith estimates distance and orientation respect to:

- Three row lines
- The line defined by the rows extremities

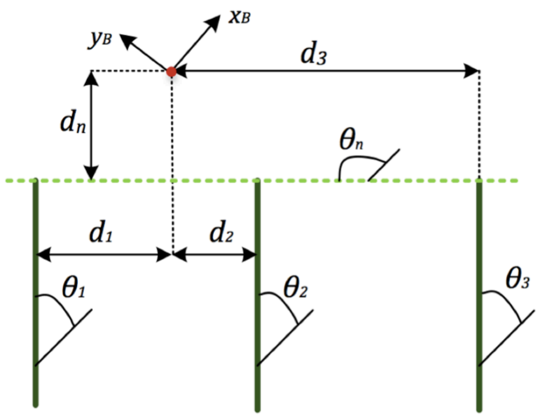


Figure 3.11: Distances and angles estimated by the n-Hough algorithm

Figure 3.11 clarifies the outputs of the n-Hough transform. (x_B, y_B) are the body frame axes, which origin is defined by the red dot, while the three parallel green lines represents the rows and the perpendicular shaded one is the line defined by them extremities. (d_i, θ_i), with $i = 1, 2, 3$, are distances and relative orientation of the row lines respect to the robot, while (d_n, θ_n) defines the position of the rows estremities line. Thanks to the structure knowledge, also in this case an *internal model* can be defined, which contains the following information:

- Rows are parallel
- Row width
- Relative orientation between rows and row extremities line, which in most cases are perpendicular.
- Row structure is fixed

Imposing the internal model, the algorithm is asked not for casual linear models, but for three parallel lines, at a certain mutual distance, and a fourth one, with a well defined orientation respect to the first three. The last condition allows to reduce the research grid, in order to make the algorithm faster and more reliable.

Figure 3.12 represents internal model application and research grid reduction of the localization algorithm outside the rows. Orange dots represents the previous iteration solutions, namely three parallel lines (right side) and a perpendicular one (left side). Similarly to the localization algorithm within rows, research areas are

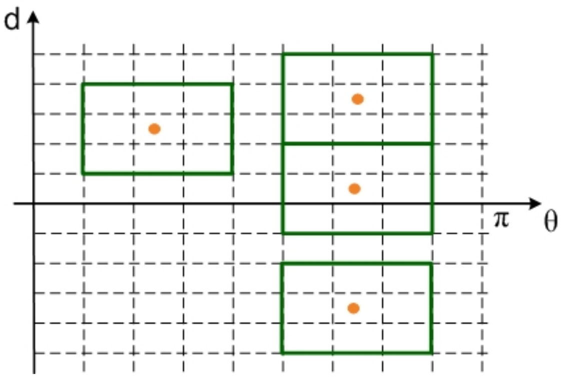


Figure 3.12: Research grid of the proposed localization algorithm outside the rows

calculated around the previous solutions, reducing the algorithm complexity and computation time.

3.7 Robot Control Law

Once the robot has localized itself in both situations described in sections 3.5 and 3.6, it knows its relative position respect to the structure in any moment during a mission. Actually, since the position of the structure is known, also the robot absolute position can be calculated.
What the robot needs now is a control law, which calculates the robot reference speed on the basis of localization and desired trajectory [7]. Localization algorithms presented in chapter 3 are not model based, in the sense that can be applied to any kind of robot or platform. Now, in order to introduce the proposed control law, the work must focus on the robot employed for experimental tests. As described in chapter 2, the developed agricultural robot is a tracked rover, moved by two tracks driven by independent motors (see figure 3.13). In order to frame the control problem, a mathematical model of the robot is presented. Since the focus is on navigation and localization techniques, with no emphasis on ground friction, forces and so on, a simple differential-drive model [4] of the rover kinematics is considered:

$$\begin{bmatrix} \dot{x}_I \\ \dot{y}_I \\ \dot{\psi}_I \end{bmatrix} = \begin{bmatrix} \cos\psi_I & 0 \\ \sin\psi_I & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{3.1}$$

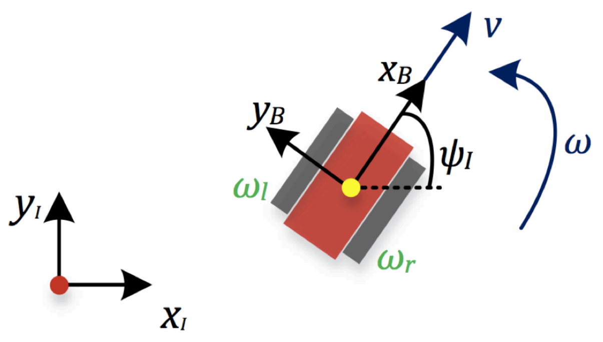


Figure 3.13: Representation of inertial and boby frames, and kinematics control inputs

where (x_I, y_I) is the position of the rover in the inertial frame (see section 3.2), ψ_I the attitude of the rover (i.e. the yaw angle) and (v, ω) are the inputs of the model, representing the translation and rotation velocity, respectively (see figure 3.13. v and ω are related to tracks velocities by the map

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} r/2 & r/2 \\ r/i & r/i \end{bmatrix} \begin{bmatrix} \omega_l \\ \omega_r \end{bmatrix} = T \begin{bmatrix} \omega_l \\ \omega_r \end{bmatrix} \tag{3.2}$$

where r is the radius of the track driving wheel, i the wheelbase and (ω_l, ω_r) the angular velocities of left and right motors, respectively. Equation 3.1 represents a model expressed in the *inertial* frame. Since the localization algorithm returns a relative position, i.e. the distance respect to the trees, and during navigation within rows the goal is to keep the rover at a certain lateral distance from the estimated lines, it is better to express the kinematic model in a *local* frame. The first assumption is that orchard rows are represented, in the inertial coordinate system, by straight lines described by

$$y = n + \tan\psi \tag{3.3}$$

where n and ψ are constants and represent, for each line, distance from the inertial frame origin and orientation respect to the x_I axes, respectively. Now the distance from the orchard rows can be defined as

$$d = \frac{y_I - \tan\psi - n}{\sqrt{1 + (\tan\psi)^2}} \tag{3.4}$$

while the relative angle between robot heading and row lines is $\theta = \psi_I - \psi$. After some mathematics, model (3.1) can be rewritten in local coordinates as

$$\begin{bmatrix} \dot{d}_i \\ \dot{\theta}_i \end{bmatrix} = \begin{bmatrix} \sin\theta_i & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ w \end{bmatrix} \tag{3.5}$$

with $i \in \{r, l\}$ indicating right and left side respectively.

3.8 Robot Control within rows

Usually, in the agricultural work, the tractor is driven imposing a constant low translation velocity and the steering is periodically corrected, in order to avoid obstacles and maintain the desired lateral distance from the row. Following this approach, a constant velocity v is asked to the robot, while the rotation velocity w is controlled by the algorithm and treated as control variable for the navigation within rows.
A conceptual scheme of the adopted control loop for navigation within rows is proposed in figure 3.14 and described here below, from left to right. The references are the desired lateral distance d_R and longitudinal velocity $v*$, both defined by the user. d_R is a number ranging from -1 to 1, where 0 means to keep the center of the row, negative values lead the rover towards left and positive towards right.

The regulator R, on the basis of references and distance d and angle θ between the robot and the row, calculates a desired steering action, represented by ω_R. Matrix T (see equation 3.2) gives back the desired motor velocities ωl_R and ωr_R. Then, the *Drivers* control the motors (*Track* block) in order to get the desired speeds ωl and ωr. This values, converted back in the form (v, ω), defines the inertial position and attitude of the robot (see equation 3.1). Moreover, environment perception is provided by the *LIDAR* sensor, which data goes to the *Localization* algorithm (see section 3.5). The latter computes distances and inclination of the lateral rows respect to the robot. Finally, from these values a kind of mean value is calculated, obtainig the feedback (d, θ) for the *Regulator*. Going deep in the control loop description, the regulator looks like

$$w = \frac{1}{v\cos\theta}(\ddot{d}_R - k_p(d - d_R) - k_d(\dot{d} - \dot{d}_R)) \tag{3.6}$$

where θ and d are the angle between the rover longitudinal axis (x_B) and the row

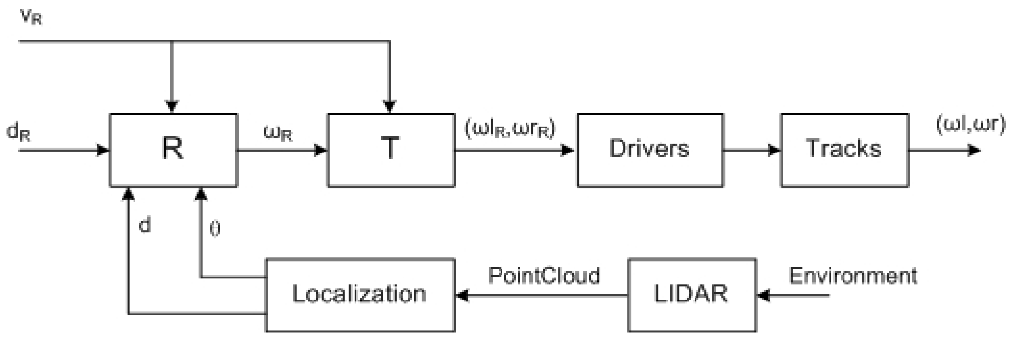


Figure 3.14: Conceptual scheme of the adopted lateral distance control loop

line and the distance from the rows, calculated as a combination of (θ_r, θ_l) and (d_r, d_l) given by the row estimation algorithm, d_R is the desired lateral distance and k_p and k_d are the proportional and derivative gains. Furthermore, the distance derivative $\dot{d}$ is estimated from the distance d. As previously mentioned, in this application the rover has to navigate at a given lateral distance from the rows, in order to perform certain tasks, meaning that the lateral distance reference is constant and leading to $\ddot{d}_R \equiv \dot{d}_R \equiv 0$. Basically the steering control law consists of a PD regulator, given by $k_d(\dot{d} - \dot{d}_R)$, plus two additional components:

- *Feedforward action*, given by the second derivative of the lateral distance reference $\ddot{d}_R$.
- *Non-linear gain*, inversely proportional to the translational speed v and to the cosine of the orientation θ. This means that the steering is lower at high speed and viceversa, in such a way to carry out stricht maneuvers at low speed, and give just small steering corrections at high speed. Regarding the angle θ, the steering results higher when the orientation respect to the rows is higher, and lower when the angle is small and the robot is more or less aligned to the desired trajectory.

3.9 Conclusions

In this chapter the proposed solution to autonomous navigation problem in orchard environment are described. Both the localization and control aspects of the autonomous navigation challenge are teken into account. For first, LIDAR-based localization in orchard environment is described, for estimating local robot position both within rows and outside, during the row change. Then, the control problem is framed, by means of a simple mathematical model, and a control law is proposed and described. Now the research work description can go on, towards validation of the proposed localization and control algorithm. In the next chapter, the developed software structure is presented, and implementation of the proposed navigation algorithms and additional functionalities described in details.

Chapter 4

Software structure

In this chapter an exhaustive description of the software structure implemented on the rover onboard computer is given, focusing on control system and HMI details.

4.1 Introduction

In the last chapter, the developed autonomous navigation algorithm has been presented in details. Approaches and most important features of both localization and control laws have been described. Of course its is foundamental to test these algorithm, in order to validate them on the field and see their behaviour in real work conditions. This chapter covers the last step of this research work towards experimental tests and deals with software implementation.
Defined the navigation algorithm and the phisical structure of the developed prototype, now the software implementation can be described, meaning all the code loaded on the onboard computer, which controls the robot, on one hand, and interfaces it with the outside world, on the other. Indeed, the proposed architecture consists of:

- **HMI** (see section 4.4), which implement communication between onboard controller and motors and allows the operator to interact with the robot, i.e. start the system and assign missions, rather than monitoring task progresses and rover position within its working environment.
- **Robot Control** (see section 4.5), which implements sensors reading, envi-

ronment features extraction, location and calculation of the motors control inputs.

First, the on-board computer will be described, focusing on the structures used for the two software components implementation and on communication between these two worlds. Once this structure has been introduced, the interfacing and control components will be described in details.

4.2 Onboard Controller

A functional diagram of the overall software structure is showed in figure 4.1, which highlights the connections between HMI, Robot Controller and external components. Green blocks represents external computation peripherals, blue are actuators, yellow are system sensors, while red boxes mean components which interacts with the human operator. Moreover, motors and encoders are phisically embedded inside the tracks, while HMI and Robot Control are implemented in the onboard controller. *Robot Controller* acquires sensors data, directly, such as laser

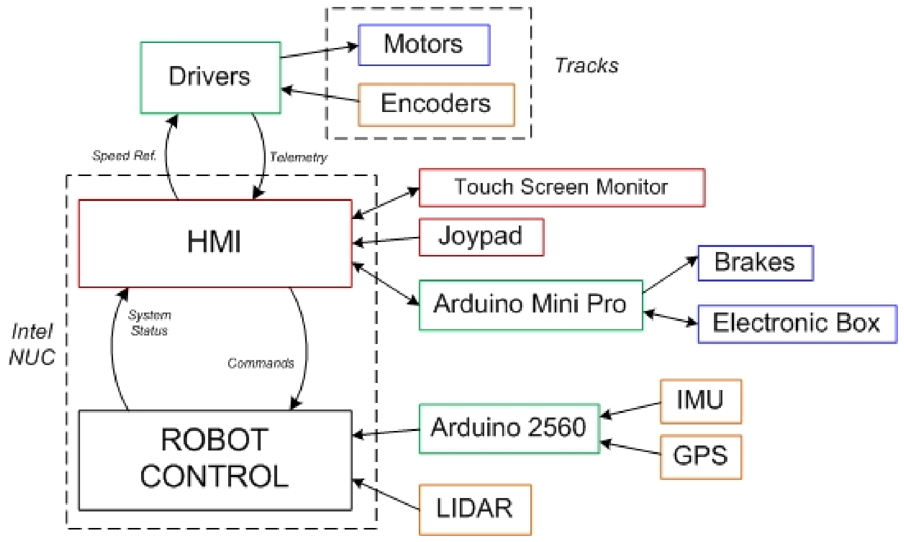


Figure 4.1: Block scheme of sotfware architecture and most important inner and outer connections

scanner, or through the *Arduino 2560* microcontroller, for GPS and IMU. It also receives, from the HMI, motors telemetry and user commands, such as missions or parameter changes. The *HMI*, on the other hand, receives joypad commands, both for manual guidance and for missions starting or stopping, and communicates with motor controllers, sending reference speeds and receiving telemetry feedback. The user interface also receives from the Robot Control different feedbacks from sensors, like GPS position, or about mission status, and the tracks reference speeds. Finally, the HMI communicates with an *Arduino Mini Pro* microcontroller, which allows brakes management and interaction with the electronic circuit. The funtionalities mentioned in the introduction (see section 4.1) are implemented within the on-board controller, *Intel NUC D54250WYKH*, a mini-PC equipped with USB, ethernet port and HDMI output. Inside are installed an Intel i5 processor, 16 GB RAM module and 120 GB SSD hard drive, ensuring sufficient performances for managing both user interface and Robot Control. Two observations come out from the described software scheme:

- The HMI is not *remote*, implemented on tablet or laptop, as in most applications, but runs on the robot's on-board computer. Even though the HMI offers all the needed functionalities for testing and vehicle management, has the disadvantage of always requiring the operator presence close to the robot. On the other hand, this feature could also represent an advantage, since prevents problems due to loss of wireless communication, expecially in critical situations, in which the robots needs to be stopped before crashing.

- Moreover, it would be natural, from the functional point of view, to have Robot Control communicating with the motors, instead of HMI. In the proposed architecture, the HMI block is not just a user interface, but a fundamental software component, implementing both *human-robot* interface and *low-level* communication.

From a different perspective, the two software components can be seen as:

- **Low Level**, composed by HMI and motor communication, on which depends system safety and reliability, since it drive the robot, dealing with user manual commands, control actions from the other software block and emergency situations.

- **High Level**, consisting in the Robot Control, which reads sensors and implements the whole navigation algorithm, with no direct interaction with motors, user and low level communication problems

-

Due to some implementation reasons and lack of time, the features discussed here above have been kept and are currently present on the developed prototype. Possible correction, or modification, of the software architecture is part of the project future developments and will be resumed in chapter 6.

4.3 HMI-Robot Control Interaction

This section describes how the foundamental components of the software structure, i.e. HMI and Robot Control, are executed on the same computer, and the communication between the two worlds is implemented. Of course, a more detailed description about this interaction will be given later (see sections 4.4 and 4.5), where the two components are described.
The robot HMI is written as a Visual Studio *form*, running under a Windows 7 operating system, while the Robot Control is implemented in ROS [14][19][18] and executed under Linux Ubuntu. In order to allow the execution of both HMI and Robot Control on the same computer, a virtual machine was created, defining the following structure:

- Physical machine with Windows 7 operating system, executing the HMI.
- Virtual machine, with Linux Ubuntu operating system, under which runs ROS system and Robot Control.

The choice to keep the HMI on physical machine is due to the fact that in the current configuration (see 4.2), the interface manages the communication with joypad, engines and brakes. Therefore, this functions, critical from a safety point of view, has been kept on the physical machine, in order to get faster responses in case of emergency and to bypass any virtual machine problem. Communication between HMI and Robot Control takes place through exchange of UDP packages, encoded according to the Mavlink protocol (see reference). The latter provides a package encoding system and a set of predefined messages, which can be used to enclose informations and thus implement data exchange between the two systems. On the Robot Control side, such a communication is implemented through the

Mavros package (see subsection 4.5.4), which provides communication with external devices. Basically, it receives Mavlink messages from the outside, extracts data and publishes it in the form of ROS messages, and viceversa from ROS to the ouside data flow. The data exchange is based on executables named *plugins* (see subsection 4.5.3), which defines Mavlink messages to read, rather than topics to publish or subscribe and all the data extraction.

4.4 Human Machine Interface

HMI is a user interface, which allows the operator to check the system status, from the electronics to fault presence on CAN bus which controls the motors, up to mission assignment and monitoring. The HMI features can be summarized as:

- Starting joypad, motors and microcontroller communication.
- Monitoring robot position via satellite map
- Feedback on electronic box and the brakes status
- Monitoring motors telemetry, from torque/currents applied to the motors up to the presence of communication faults
- Starting communication with the Robot Control
- Tests or missions assignment
- Monitoring mission status
- Winch management, for vertical position adjustment of tools installed onboard

The robot HMI is implemented as a Visual Studio *form*, developed in C# code and run under a Windows 7 operating system. The aim of this section is to give a more detailed description of the different HMI functionalities. Since any feature is linked to visual elements of the form, the interface will be presented by proposing its different screens and describing the present items. The latter mainly consists of:

- *Buttons*, used to give commands or trigger actions, like start communication with a device or send a mission to the robot.
- *Labels*, which gives a feedback to the user, showing variable content or changing background color based on some conditions or operation status.
- *TextBoxes*, which can be filled by the user, in order to set parameters, or by

the system, to show variables or feedback.

- *Boxes*, just a container for the items listed above.

4.4.1 Screens Description

The aim of this subsection is to present the most important HMI functionlities. Their description will follow a logical flow, starting from the system initialization up to mission monitoring and then having a look about less important features.

System Initialization

As the HMI is started, the screen showed in figure 4.2 appears to the user. The

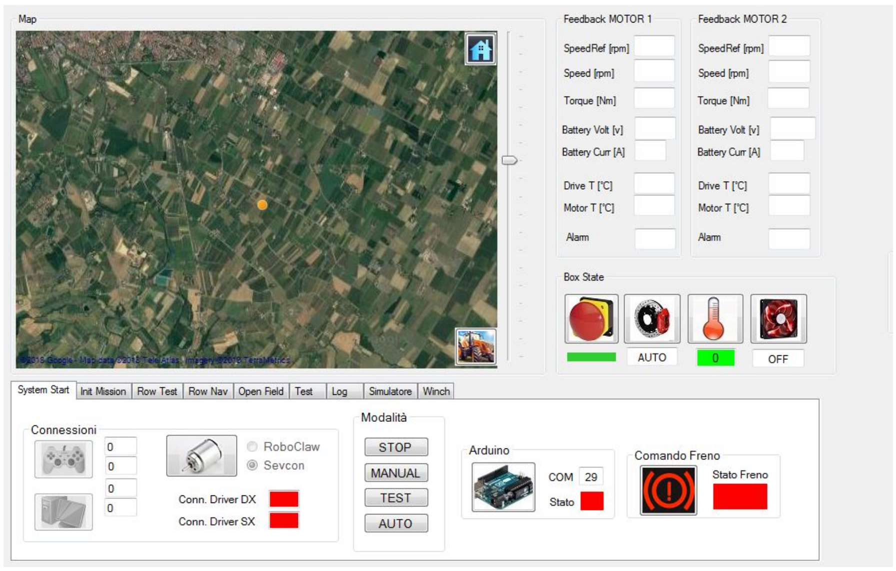


Figure 4.2: HMI initial screen

elements visible on the screen and the related functionalities are:

Satellite Map

Allows to monitor the robot position (orange dot) at any point on earth's surface. It is updated thanks to the position feedback coming from the GPS receiver, through the Robot Control block. On the right of the map there are *Home* and *Tractor* buttons, for centering the map respectively on the GPS home coordinates, where the robot control is initialized, and on the robot position.

Feedback Motor 1 and 2

It shows, for both drivers and engines, the most important variables of interest, such as (from top to bottom):

- Motor speed (provided by encoders)
- Motor reference speed (asked to the motors)
- Torque
- Battery current
- Battery voltage
- Driver temperature
- Motor temperature
- Alarms (indicating the presence of faults drivers or motors)

Box State

It reports some feedback about system and electronic box status. In particular, from left to right:

- Emergency button. Indicates whether the emergency button is pressed or not, and therefore whether engines and brake circuit are power supplied.
- Parking brake. Indicates whether tracks brakes are automatically controlled, through the microcontroller, or the circuit is bypassed. This last function is regulated by a switch placed on the electronic box and allows to manage the brakes manually and without starting the system, useful in case of failure or emergency,
- Electronic box temperature. Through the microcontroller, feedback on the electronic box internal temperature is received, in order to monitor it and prevent critical situations.
- Fans status. On the basis of read temperature, the microcontroller manages in different ways the fans placed inside the electronic box, in order to control the temperature. Feedback on fan status is reported to the HMI.

System Start

At the bottom of figure 4.2 there are some panels, the first of which allows to start the system. From left to right, the following items compare:

- Joypad connection. The first thing to do is connecting to the joypad, so that the user is always able to manually control the robot or apply the brakes, once the system is started.

- Simulator start. Action alternative to joypad connection, allows to start a simulator, in order to test so many features without the computer being physically connected to robot or joypad.
- Motors connection. Starts CAN communication with the motors. From here, the robot is active and can be manually moved.
- Mode selection. Allows to chose the operative mode, among
  - *Stop*, safety mode in which motors are stopped and brakes are always active. It is useful at the system start or to carry out debug safely.
  - *Manual*, in which the robot receives the user's manual commands. The motors reference speeds are directly computed by the HMI, on the basis of the joypad sticks and does not come from the Robot Control block.
  - *Test*, useful for testing the tracks motion, through the imposition of different speed levels or some predefined trajectories, for example rectilinear, circular or ”8” shape trajectories.
  - *Auto*, automatic mode in which the Robot Control calculates the tracks reference speeds, through the implemented localization and control algorithms.
- Microcontroller connection. Starts the serial connection with the Arduino Mini Pro microcontroller. From now the HMI can receive feedback from the electronic box and manage the parking brakes.
- Brake management. Allows to activate or deactivate the parking brake. This function is replicated also on the joypad, in order to remotely start or stop the robot.

The functions seen so far allows to start the system and manually move the robot. Instead, in order to start control system and autonomous navigation, it is necessary to take another step and go on to the HMI second panel.

Init Mission

It allows to start communication with the Robot Control, check for errors in sensors reading and start missions (see figure 4.3). From left to right, the panel is composed of the following boxes:

Switch and Laser

The two buttons allow to power ethernet switch and laser scanner, enabling com-

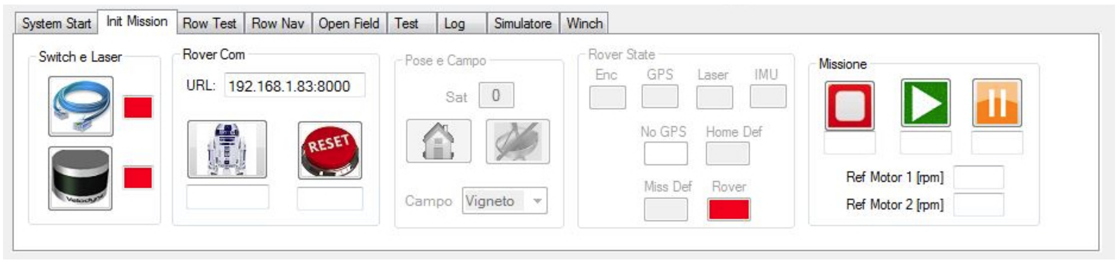


Figure 4.3: HMI panel for starting the communication with the robot and setting the foundamental parameters

munication between HMI, Robot Control and LIDAR. This feature is useful just for a matter of energy savings, since the two devices are employed only in the automatic mode. Alongside the buttons two labels give a feedback about the devices power supply condition, changing color changes from red to green when they are activated.

Rover Com

Manages the communication between HMI and Robot Control, through the following items:

- URL. Allows to set or modify Robot Control IP address and port.
- Start communication. Starts UDP communication between HMI and Robot Control. The label below reports the communication status.
- Reset. In case of necessity or problems, the whole Robot Control block, as far as the HMI, can be reset, in terms both of mission and communication status.

Pose e Campo

Allows to monitor the satellite coverage, manage the initial GPS settings and choose the orchard in which the mission has to be carried out. From top to bottom, the box is composed of:

- Sat. This label reports the number of GPS satellites currently visible from the receiver.
- Home. If the number of visible satellites is enough, allows to set the *home* position of GPS latitude and longitude coordinates
- No GPS. Here the user can decide whether to use the GPS or not during the current mission. This can be useful for indoor test experiments, or in case the GPS receiver is not present onboard or out of order.

- Campo. Allows the user to decide the field where the robot has to carry out its mission. Once the orchard is defined, the system automatically loads the related *data structure* (see section C.1), containing orchard foundamental parameters like GPS position, number and length of rows and so on.

Rover State

This box contains some labels which give feedback about sensors and task definition status and have to be checked before starting the mission. Red color means a fault or a missing condition, green a positive feedback, while grey means a missing component, like a sensor non installed on the robot in the current configuration. In particular, feedback tells if (from top to bottom)

- Onboard sensors (Encoder, GPS, laser scanner and IMU) are read correctly
- GPS is used or not in the current mission
- Home position has been defined (of course, only if GPS is used)
- Mission parameters have been set
- Robot parameters have been defined

Missione

The buttons on top of this box allow to *Stop*, *start* and *Suspend* the mission. At the bottom, two labels show the motor reference speeds calculated by the Robot Control block.
The next two panels let the user defining mission or test parameters, as far as monitoring robot operative status during the work.

Row Test

This panel allows the user to define tests that the robot has to carry out and to monitor the work progression. This function is useful for testing modifications or the robot in general, before starting a real mission.The screen (see figure 4.4) is composed of the following boxes.

Test Parameters

This box allows to define test parameters and send them to the Robot Control. From left to right, the following items compare:

- Test type. Using the button and the two radio buttons (top left) the test type can be selected, choosing between navigation in the row, row change or

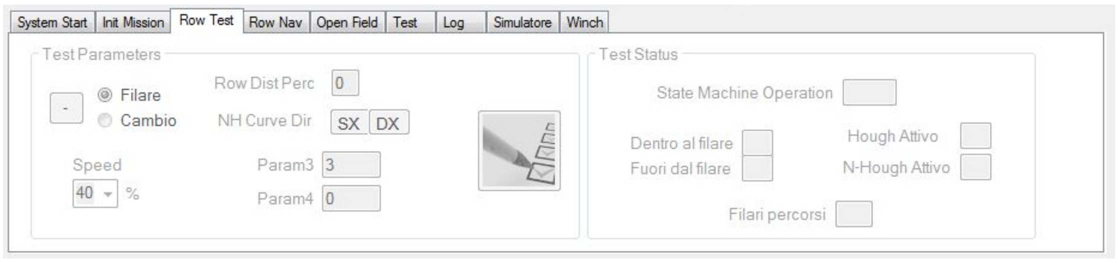


Figure 4.4: HMI panel for test definition, assignment and monitoring

some combinations of the two possibilities.

- Speed. It allows to regulate the speed that the robot has to keep during the test, choosing between different percentage values.
- RowDistPerc. This label defines the lateral distance that the robot has to mantain during navigation within rows
- NHCurveDir. In case the test includes row changes, this label defines the curve direction, towards left or right.
- Invio. The button (extreme right of the box), if the robot is not busy, sensors are read correctly and there are no missings in the initialization process, allows to send the test to the Robot Control.

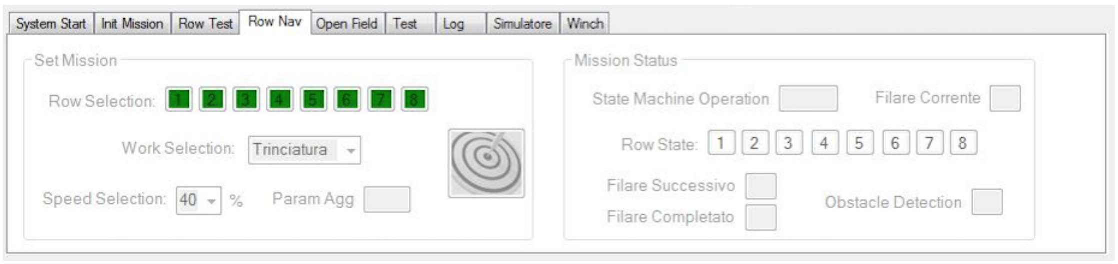
Test Status

This box is designed for monitoring the ongoing test. For top to bottom, is composed of:

- State machine operation. This label reports the operating condition of the Robot Controller, communicating if the system is busy carrying out the test, rather than initializing or waiting for new commands
- Dentro/Fuori dal filare. Indicates, on the basis of LIDAR localization, whether the robot is within or ouside the rows.
- Hough/N-Hough attivo. This labels give a feedback about localization algorithm status, telling if they are active or not.
- Filari percorsi. Indicates the number of rows travelled by the robot during the current test.

Row Nav

Using this panel the user can set and send missions to the robot, as far as monitor its progresses once work has started. The screen (see figure 4.5) is composed of the following boxes:



Work Selection: Trinciatura

Speed Selection: 40 %

Figure 4.5: HMI panel for setting missions, send them to he robot and monitor their progesses

Set Mission

Allows to set the mission parameters, through the following items (from top to bottom):

- Row Selection. The buttons allow to choose in which rows the robot has to work, rather then skip.
- Work Selection. Through this item the user can define the task that the robot has to carry out, like fertilizer distribution, inspection, and so on. Based on this choise, a series of parameters is automatically load on the Robot Control, like dimensions and weight of the needed tool, or speed which has to be mantained in order to succesfully carry out the task.
- Speed Selection. Allows to select, between different percentage values, a speed level at which the mission will be carried out.
- Send. Pressing this button (right side of the box), the mission is sent to the Robot Control.

Mission Status

This box, placed in the right side of the panel, contains some labels which shows the mission status. From top to bottom, compare:

- State Machine Operation. Reports the operating condition of the Robot Control, indicating if the platform is waiting for the mission, rather that busy or stuck due to obstacle presence.

- Filare corrente. Shows in which row the robot is currently working
- Row State. This labels describes the mission progresses, indicating in which rows the robot has already worked or not, and in which one the work is currently in progress.
- Filare Successivo. Reports which row will be the next to be travelled, based on the trajectory pre-calculated by the Robot Control.
- Filare completato. Indicates if the robot has finished to work in the current row.
- Obstacle Detection. Communicates whether the robot has encountered or detected obstacles on its path, and if it is stuck or trying to bypass the object.

Open Field

This mode has been designed in order to guide the robot in open field environments, i.e. to work on wheat fields or automatically move the rover from its base to the orchard where is has to carry on its mission. For lack of time this feature has not been implemented yet, and therefore, the corresponding panel is empty. See future developments (chapter 6) for a more detailed description of this feature.

Test

This panel activates when the user selects the *Test* mode and allows to define trajectories and parameters and finally start the experiment. From left to righ,

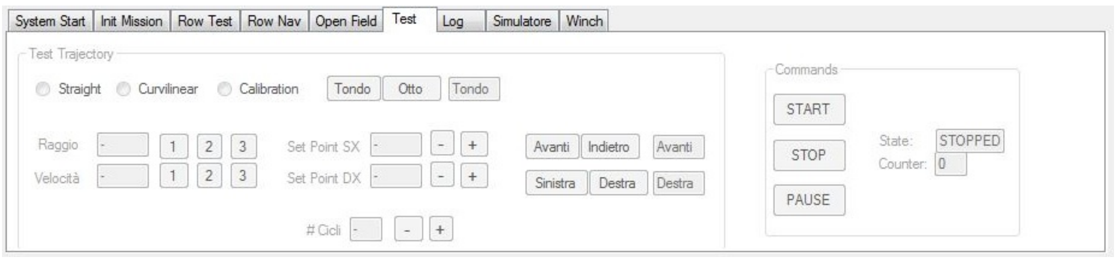


Figure 4.6: HMI panel for setting and starting experiments for testing the robot tracks movements

this panel (see figure 4.6) is composed of two boxes:

Test Trajectory

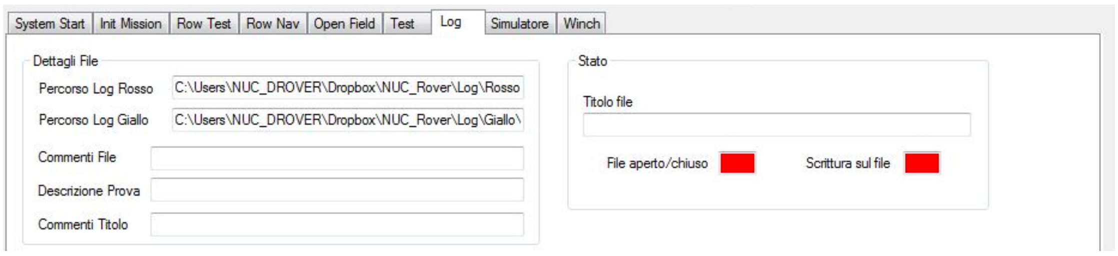
- textStraight, Curvilinear and Calibration
  Here the user can choose between rectilinear and curvilinear trajectories, or a test in which the single robot tracks are driver at arbitrary velocities.
- Tondo/Otto.Allows to choose the type of curvilinear trajectory, between circular and 8-shape path.
- Raggio/Velocità. Defines the curve radius, in case of curvilinear trajectory, and the robot translation velocity to be kept during the test
- Set Point DX/SX. This button set, both offline and online, the tracks velocities, in case of *Calibration* test.
- Avanti/Indietro. Defines if the test has to be carried out moving forward or backward.
- Sinistra/Destra. Allows to define, in case of curvilinear trajectory, if the curve has to be performed towards left or right directions.
- Cicli. Defines how many time the selected trajectory has to be repeated during the test.

Commands
This box contains the buttons which allows to start, stop or suspend the current test, and two labels, which give a feedback about test status and number of cycles currently performed.

Log

Log files collect, during robot operations, all available data on what is happening on board. They are specialized according to the current operative mode. For example, in the *Manual* case, only data related to motors telemetry is collected, while in the *automatic* one also sensors data and Robot Control internal status are saved. Through this panel (see 4.7) the user can set log files details, like saving path, keywords in the title or comments, in order to make it easy to check or analyze collected data.

Log files are automatically managed by the interface, starting data collection when the brake is disabled, in manual mode, or when the mission is started, in Automatic mode. The Status box (right of the panel) allows to check the current log path, whether the file is open or closed and actually written.

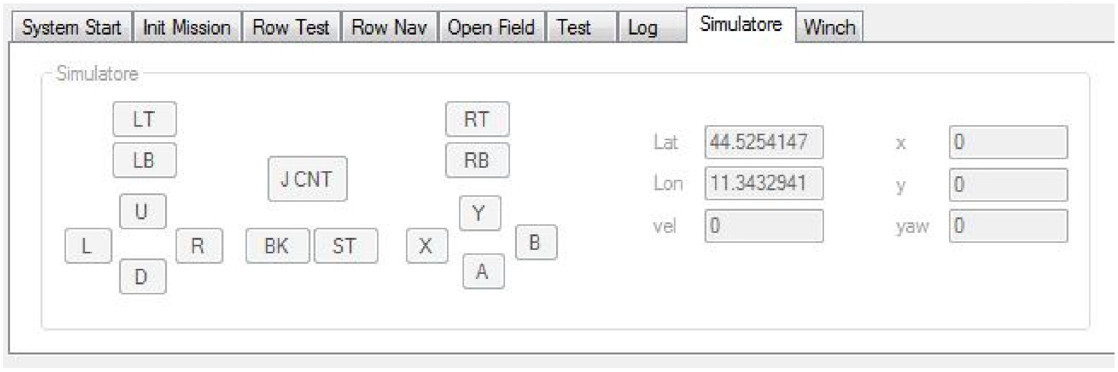


Percorso Log Rosso	C:\Users\NUC_DROVER\Dropbox\NUC_Rover\Log\Rosso
Percorso Log Giallo	C:\Users\NUC_DROVER\Dropbox\NUC_Rover\Log\Giallo\
Commenti File	
Descrizione Prova	
Commenti Titolo	
Titolo file	

Figure 4.7: HMI panel for setting log files detail

Simulatore

The simulator is an under development functionality which allows to test different system components without the on-board computer being connected to joypad and motors. The only components currently present in the simulator are joypad, GPS and magnetometer (see figure 4.8), virtualized by means of buttons and labels. Possible developments of this functionality are discussed in chapter 6.



Lat	44.5254147	x	0
Lon	11.3432941	y	0
vel	0	yaw	0

Figure 4.8: HMI panel for the robot simulator

Winch

This panel is used for controlling winch through the joypad. The tool mounted onboard can therefore be raised or lowered easily, without using the manual switch placed at the front of the robot.
If the HMI is in Manual mode, the panel is accessible and the user can enable winch remote control by pressing the button (see figure 4.9). At this point, using the joypad, tool height can be regulated, also receiving visual feedback on the panel at right side.

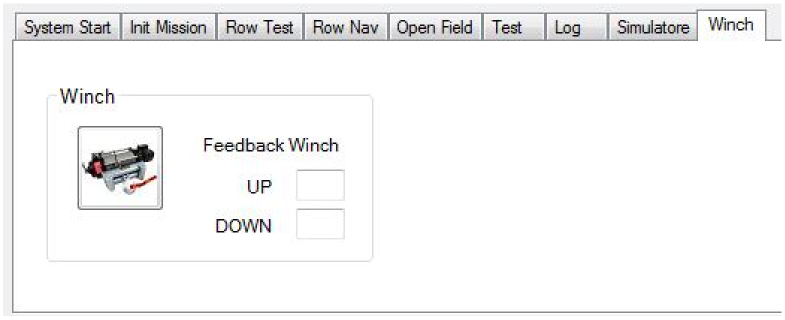


Figure 4.9: HMI panel for winch remote control

4.5 Robot Control

The Robot Control includes everything that implements navigation, communication and sensors reading, HMI interaction and mission management. More precisely, starting from the low level, the Robot Control functionality can be summarized as:

- LIDAR communication
- Serial communication with microcontroller, for GPS and IMU reading
- Sensors data processing
- Localization respect to the environment
- Robot *low-level* control
- Mission and robot management
- HMI communication

The Robot Control is implemented in ROS (acronym for Robot Operating System) [19], a development and execution environment running under Linux Ubuntu operating systems. The algorithms representing the different features of Robot Control are implemented as C ++ classes and executed as *ROS nodes*. These applications have access only to their own variable space and communication through memory sharing is not possible. Communication between the nodes is implemented through an exchange of messages, *ROS messages*, which travel on channels, visible from all nodes, called *ROS topics*. Nodes, as far as settings and messages definition files, are included inside appropriate containers, called *ROS packages*.
Since the direct relation between nodes and algorithms, the latter will be expained describing the correspondent nodes and packages. Before this, messages and topics used in the Robot Control will be listed and explained, thus introducing the

different informations exchanged within the ROS system and their meaning. This will form a basis for nodes description. After this, the created Mavros plugins will be presented, which implements communication with HMI, concluding the introduction started in section 4.3, and Arduino 2560 microcontroller, for sensor reading. Finally, ROS packages will be presented, showing how they are composed and how the nodes can be launched.

4.5.1 ROS Messages

In this subsection the ROS messages used in the Robot Control structure will be presented, describing their meaning. First, the user-defined messages will be presented, for which also a list of the contained variables will be given. Than, also the employed ROS *standard* messages will be taken into account, in order to give an exhaustive description of all the informations exchanged within the Robot Control.

User-defined Messages

- **Commands.msg**. Contains the most important commands coming from the HMI, i.e. from the user, for mission and brake management, GPS settings, and so on. See table 4.1.

Variable name	Type	Description
header	std_msgs/Header	Standard message component, containing data like timestamp, parent reference frame, ...
start	bool	Starts the mission
stop	bool	Stops the task mantaining mission details, giving the possibility to restart it
reset	bool	Stops the mission and resets mission details and Robot Control status
pause	bool	Suspends the mission. In the meanwhile, the robot can be manually driven by the user
pose_init	bool	Defines the *home* GPS position, i.e. the origin of the inertial reference frame
brake	bool	Activates/deactivates brakes
vel_traslaz_ref	uint8	Defines the percentage value of speed to be mantained
use_gps	bool	Enables/disables GPS use for the current task

Table 4.1: *Commands* ROS message

- **ExtremeRow.msg**. Contains informations about the robot approach to the row extremities, i.e. whether it is entering or leaving a row, extracted from LIDAR data. See table 4.2.

Variable name	Type	Description
extreme_row	uint8	1 means that the robot is exiting the current row, 2 that the robot is entering a new row

Table 4.2: *ExtremeRow* ROS message

- **HoughPose.msg**. Contains the results of the localization algorithm, reporting the robot relative position respect to row lines. See table 4.3.

Variable name	Unit	Type	Description
active	–	bool	This flag is set when the Hough algorithm is active, reset otherwise
valore_init_letto	–	bool	Reports if the algorithm has initialized, at the first iteration of row navigation or row chenge
d_destra	m	float32	Lateral right distance from the estimated row line
teta_destra	rad	float32	Robot heading respect to right estimated row line
d_sinistra	m	float32	Lateral left distance from the estimated row line
teta_sinistra	rad	float32	Robot heading respect to left estimated row line
d_filare_1	m	float32	Lateral distance from first estimated row line
teta_filare_1	rad	float32	Robot heading respect to first estimated row line
d_filare_2	m	float32	Lateral distance from second estimated row line
teta_filare_2	rad	float32	Robot heading respect to second estimated row line
d_filare_3	m	float32	Lateral distance from third estimated row line
teta_filare_3	rad	float32	Robot heading respect to third estimated row line
d_normale	m	float32	Lateral right distance from perpendicular estimated row line
teta_normale	rad	float32	Robot heading respect to perpendicular estimated row line

Table 4.3: *HoughPose* ROS message

- **Kinematics.msg**. Contains the estimated kinematics of the robot, in terms of planar position and speed . See table 4.4.

Variable name	Unit	Type	Description
x	m	float32	Translation on the x_i axes
y	m	float32	Translation on the y_i axes
psi	rad	float32	Yaw attitude respect to the x_i axes
v_x	m/s	float32	Velocity on the x_b axes
v_y	m/s	float32	Velocity on the y_b axes
w	rad/s	float32	Rotation speed around the z_b axes

Table 4.4: *Kinematics* ROS message

- **RefSpeed.msg**. Contains the reference speeds for the track motors calculated by the Robot Control. See table 4.5.

Variable name	Unit	Type	Description
speed_dx	*rpm*	float32	Reference speed for the right track motor
speed_sx	*rpm*	float32	Reference speed for the left track motor
standby	–	bool	Standby command for the motors

Table 4.5: *RefSpeed* ROS message

- **RosState.msg**. Describes the status of the Robot Control, in terms of sensors reading and robot initializations. Only if these conditions are satisfied, the robot accept to start the mission. See table 4.6.

Variable name	Type	Description
use_gps	bool	Reports whether GPS is in use or not
pose_initialized	bool	Tells if the home position has been defined
rover_defined	bool	Reports whether the motor communication is active and robot parameters have been loaded on the Robot Control
gps_state	uint8	Reports GPS state: 0 means the sensor is ok, 1 stays for sensor fault and 2 for sensor disconnected
laser_state	uint8	Reports LIDAR state: 0 means the sensor is ok, 1 stays for sensor fault and 2 for sensor disconnected
imu_state	uint8	Reports IMU state: 0 means the sensor is ok, 1 stays for sensor fault and 2 for sensor disconnected

Table 4.6: *RosState* ROS message

- **RowReference.msg**. Contains the mission details, defined by the user and sent to Robot Control from HMI. See table 4.7.

Variable name	Type	Description
field	uint8	Indicates the orchard in which the mission has to be carried out
work	uint8	Defines the work to perform
param1	uint16	Additional mission parameter
row_detail	bool[64]	Tells which rows have to be worked or not
system_type	uint8	Indicates which robot has to carry on the mission, in case of a team instead of single unity

Table 4.7: *RowReference* ROS message

- **State.msg**. Contains the mission status, i.e. actual position respect to the rows, obstacle presence and so on. See table 4.8.
- **SystemParam.msg**. Contains details about GPS management and experimental platform. See table 4.9.

Variable name	Type	Description
state	uint8	Indicates the operative state of the Robot Control
filare_corrente	uint8	Identifies thw row where the robot is currently acting
filare_prossimo	uint8	Indicate the next rox inside which the robot will enter
stato_filare	uint8	Tells the work progress in the current row
fine_filare	uint8	Reports whether the robot is approaching the row extremity
obs_det	bool	Indicates whether the robot has encountered any obstacle
pos_obs	int16	In case of obstacle detection, indicates the obstacle position within the row
bypass	bool	Indicates if the robot has managed to bypass the obstacle

Table 4.8: *State* ROS message

Variable name	Type	Description
system_type	uint8	Defines the selected experimental platform, in case multiple robots are available
use_gps	bool	Indicates whether GPS data is used for position estimation

Table 4.9: *SystemParam* ROS message

- **TestParam.msg**. Contains the test details, defined by the user and sent to the Robot Control. See table 4.10.

Variable name	Type	Description
test_type	uint8	Indicates the test type chosen by the user (within rows, row change, or combination of these ones)
vel	int8	Speed at which the test has to be carried out
lateral_dist	float32	Distance to keep respect to the rows of trees
param2	float32	Additional test parameter
NH_curve_dir	float32	Curve direction, in case of row change navigation test
param4	float32	Additional test parameter
system_type	uint8	Indicates which robot has to carry on the mission
campo	uint8	Orchard in which the mission has to be carried out

Table 4.10: *TestParam* ROS message

- **TestState.msg**. Reports the current test status, in terms of position within or outside the rows, number of rows travelled, and so on. See table 4.11.

Variable name	Type	Description
test_type	uint8	Type of test that the robot is carrying out
state	uint8	Reports the Robot Control operative state
dentro_fuori	uint8	Indicates whether the robot is within or outside a row
cont_filari	uint8	Number of rows travelled during the current test

Table 4.11: *TestState* ROS message

Standard Messages

- **sensor_msgs/PointCloud2.msg**. Contains the LIDAR 3D pointcloud, plus some additional parameters, like pointcloud dimensions and reference frame. For message details the reader is referred to bibliography.
  - **sensor_msgs/LaserScan.msg**. Contains 2D laser scanner data and some additional details. For message details the reader is referred to bibliography.
  - **geometry_msgs/Imu.msg**. Contains data from an IMU, composed of attitude, angular velocity and linear acceleration, complete with covariance matrices. For message details the reader is referred to bibliography.
  - **geometry_msgs/PoseStamped.msg**. Contains pose informations, in terms of 3D position expressed on x, y, z axis and attitude, parametrized as quaternion. For message details the reader is referred to bibliography.
  - **mavros_msgs/HomePosition.msg**. Contains home position informations, in terms of GPS coordinates, local position and orientation, expressed as quaternion. For message details the reader is referred to bibliography.
  - **mavros_msgs/RawGps.msg**. Contains data coming from GPS receiver, like coordinates, visible satellites, velocity, and so on. For message details the reader is referred to bibliography.

4.5.2 ROS Nodes

Here the user-defined ROS nodes implementing the Robot Control are presented. For each node, a table reports the read/written topics, specifying the associated message, and the functionalities and working principle are described. The nodes downloaded from *github* or other sources are briefly described in 4.5.4, since are always part of ROS packages. The only exception is the mavros package, since it has been modified in order to provide communication with microcontroller and HMI. The added functionalities are described in section 4.5.3.

- **row_state_machine.cpp**
  This node is the heart of the Robot Control. It manages missions, robot behaviour and user commands coming from the HMI. Other nodes of the implemented ROS structure act on the basis of its operative state. Published and subscribed topics are summarized in table 4.12. In figure 4.10

Published Topics	Message
/driver_commands	RefSpeed
/state_row_nav	State
/state_row_test	TestState

Subscribed Topics	Message
/ros_state	RosState
/hough_pose	HoughPose
/vs/commands	Commands
/vs/row_references	RowReference
/vs/test_param	TestParam
/extreme_row	ExtremeRow

Table 4.12: *row_state_machine* node published and subscribed topics

a representation of the implemented state machine is proposed. It reports all the possible states, but, for semplicity, just the most important transitions. Orange and red colors represent the starting and ending points, i.e. *INIT* and *END* states. Blue indicates the states belonging to the nominal navigation behaviour (*ROW*, *CHG* and *PAUSE*), while the other regards recovery actions related to obstales presence. In particular, red represents states (*OBS_AV_ROW* and *OBS_AV_CHG*) in which an obstacle has been detected, but the robot can avoid or bypass it, meaning that the mission is not compromised. Differently, violet color indicates a more serious situation (states *TO_INIT_ROW* and *TO_END_ROW*),in which the mission on a row, or a row change, can not be carried out because of obstacle presence. An

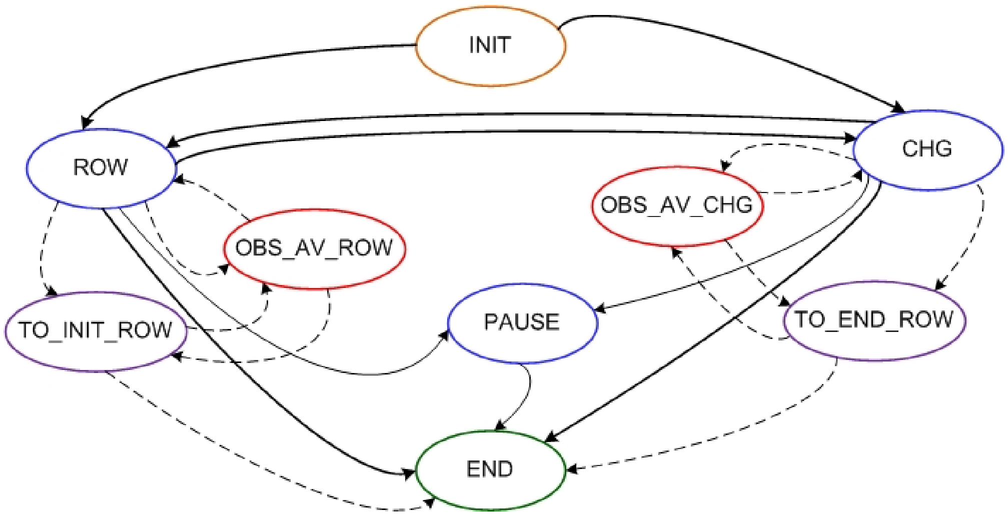


Figure 4.10: Representation of the *row_state_machine*, with possible states and most important admitted transitions

in-depth description of the possible states is proposed here below, showing ID, name and function. For each state, also admitted transitions and related conditions are described.

0 **INIT**. The Robot Control is started or has just been reset. Once all the parameters and missions have been set, the mission can start. In this state the robot can't move and the reference speeds are always *zero*. Possible transitions are:

→ *ROW*. The robot receives the *start* command from the HMI and the mission starts with a row navigation section.

→ *CHG*. *start* command is received and the test starts with a row change action (cases of *TEST_CAMBIO* or *TEST_CAM_FIL* tests)

→ *PAUSE*. The user can decide to *pause* the Robot Control before mission beginning, in order to manually move the robot and set a different initial position.

1 **ROW**. The robot is automatically navigating within rows. Possible transitions are:

→ *CHG*. The rover has arrived at the extremity of the current row and the mission goes on with a row change or navigation in the next row (*TEST_FIL_CAM* or *TEST_FIL_CAM_LOOP* tests or *ROW_NAV* mission, if the current row is not the last one).

→ *OBS_AV_ROW*. An obstacle is detected in *yellow zone* and the robot estimates to have enough space to bypass it.

→ *TO_INIT_ROW*. An obstacle is detected in *red zone* or the obstacle in *yellow zone* can't be bypassed.

→ *END* The current row is finished and the mission is complete (case of *TEST_FILARE* or *TEST_CAM_FIL* tests or *ROW_NAV* mission, if the current row is the last one).

→ *PAUSE*. The current action is paused by the user (*/vs/commands* topic).

2 **OBS_AV_ROW**. An obstacle has been detected in front of the robot and it is trying to bypass it and continue the mission. Possible transitions are:

→ *ROW*. The robot has bypassed the detected obstacle, and resumes its mission.

→ *CHG*. The robot has bypassed the obstacle, but during the maneuver it encounters the row extremity. So, if the latter is not the last row, it can directly carry on the row change and continue the mission.

→ *TO_INIT_ROW*. The obstacle can't be bypassed, since it enters the *red zone* ruring the maneuver, or it *moves* towards the robot.

→ *END*. The robot has bypassed the obstacle, but during the maneuver it encounters the row extremity. So, if the latter is the last row, the mission is finished.

→ *PAUSE*. The current action is paused by the user (*/vs/commands* topic).

3 **TO_INIT_ROW**. The obstacle can't be bypassed. The robot returns to the beginning of the row and takes another one, recalculating its path. Possible transitions are:

→ *ROW*. The obstacle moved away (case of *human* obstacle) or the rover went back outside the curret row and then to the entrance of a new one, if the row in which obstacle has been detected is not the last, according to mission settings.

→ *OBS_AV_ROW*. The robot sees the obstacle in *yellow zone*, and not in the *red* one anymore, since it moved, and now there is enough space for bypassing it.

→ *END*. Similar to the *ROW* transition condition, but in which the blocked row was the last to travel.

→ *PAUSE*. The current action is paused by the user (*/vs/commands* topic).

4 **CHG**. The robot is automatically performing a row change. Possible transitions are:

→ *ROW*. The robot encountered the entrance of the new row and the mission is not finished (case of *ROW_NAV* mission or *TEST_FIL_CAM_LOOP* or *TEST_CAM_FIL* tests).

→ *OBS_AV_CHG*. An obstacle is detected inside the *yellow zone* and there is enough space to bypass it.

→ *TO_END_ROW*. The robot sees an obstacle in the *red zone* or has not enough space to bypass the object in *yellow line*.

→ *END*. Similar to the *ROW* transition condition, but in which the row change action completed the mission (case of *TEST_CAMBIO* or *TEST_FIL_CAM* tests).

→ *PAUSE*. The current action is paused by the user (*/vs/commands* topic).

5 **OBS_AV_CHG**. An obstacle has been detected in front of the robot and it is trying to bypass it and continue the mission. Possible transitions are:

→ *ROW*. Avoiding the obstacle, the robot reached the beginning of the row in which the mission has to go on.

→ *CHG*. The obstacle has been bypassed and the robot can continue the row change maneuver.

→ *TO_END_ROW*. The robot can't bypass the obstacle, which enters the *red zone*, or the obstacle is moving toward the robot.

→ *END*. Avoiding the obstacle, the robot reached the beginning of the desired row and the mission is completed (case of *TEST_FIL_CAM_LOOP* or *TEST_CAM_FIL* tests).

→ *PAUSE*. The current action is paused by the user (*/vs/commands* topic).

6 **TO_END_ROW**. The obstacle can't be bypassed. The robot returns to the row from which it was exiting.Possible transitions are:

→ *OBS_AV_CHG*. The obstacle is in the *yellow zone*, and not in the *red* one anymore, so the robot can try to bypass it.

→ *TO_INIT_ROW*. The obstacle couldn't be bypassed, so the robot returns at the beninning of the row from which it has just exited.

→ *PAUSE*. The current action is paused by the user (*/vs/commands* topic).

7 **END**. The task is completed. The user can both restart the mission or assign a new one to the robot. From this state there are no transition. Once the mission is finished, the state machine can be forced to *INIT* by the *stop* or *reset* commands, depending whether the user wants to restart the mission or assign a new one to the robot.

8 **PAUSE**. The mission or test has been paused by the user, which can

now manually drive the robot. Possible transitions are:

→ *Previous state*. If during the pause the robot has not exited the current row or entered a new one during manual maneuvers, as soon as the *pause* is deactivated the state from which it has been called is restored.

→ *ROW*. Called if during a manual maneuver the robot has encountered the beginning of the new row, the previous state is *CHG* and the mission is not finished (case of *ROW_NAV* mission or *TEST_CAM_FIL* or *TEST_FIL_CAM_LOOP* tests).

→ *CHG*. Called if if during a manual maneuver the robot exited the current row, the previous state is *ROW* and the mission is not finished (case of *ROW_NAV* mission or *TEST_FIL_CAM* or *TEST_FIL_CAM_LOOP* tests).

→ *END*. Called if during a manual maneuver the robot has encountered the beginning of the new row, the previous state is *CHG* and the mission is finished (*TEST_CAMBIO* or *TEST_FIL_CAM* tests), or if it exited the current row, the previous state is *ROW* and the mission is finished (*TEST_FILARE* or *TEST_CAM_FIL* tests).

In order to clarify the state machine behaviour, at least in *nominal* conditions, i.e. in absence of obstacles on the path, an example of state transition during a mission is proposed.

– After the launch, the system is in *INIT* state, in which it initialize sensors, and receives mission parameters and field details from the HMI.
– When the state machine receives the *start* command, if the initialization procedure has been correctly completed, it start the mission and change state from *INIT* to *ROW*, in which the robot navigates from the beginning to the end of the current row.
– Once the extremity of the row is recognized, the transition from *ROW* to *CHG* is activated. In this state the robot carry out the row change.
– When the beginning of the new row is detected, the system change state from *CHG* to *ROW*. This procedure is repeated for all the next rows, alternately row navigation and change.

- Once the end of the last row is detected, the system evolves from *ROW* to *END* state. At this point the mission is finished and the robot waits for new commands.

Possible missions assigned by the user, and accepted by the Robot Control, are explained below, reporting their ID, name and description:

0 **ROW_NAV**. A complete mission, in which the robot has to travel the selected rows carrying out the assigned task.

1 **TEST_FILARE**. Test mode in which the robot has to navigate within a single row.

2 **TEST_CAMBIO**. In this test mode the robot has to carry out a single row change, from the end of a row to the beginning of the next one

3 **TEST_FIL_CAM**. Test mode, combination of navigation in the row and row change.

4 **TEST_CAM_FIL**. Test mode, similar to the previous case, in which the row change is performed before the navigation within rows.

5 **TEST_FIL_CAM_LOOP** Test mode in which the robot carries out row nagigation and row change, in loop.

Finally, the most important features of this node are described in the list below.

- The node subscribes the */vs/commands* topic. Its *stop* command cause state machine returning to the *INIT state*and reset of mission-related variables, while the *reset* command cause a complete reset of the node memory.
- At the reception of */vs/row_references* and */vs/test_param*, orchard and robot parameters are loaded on node variables from the structure where they are stored (see chapter C). Moreover, also the mission or test type is defined.
- The node state (*/state_row_nav*and */state_row_test* topics) is published only if a change happened. Differently, once the mission started, motor reference speeds (*/driver_commands* topic) are sent at every node iteration.
- The non-linear control law which computes motors reference velocities

is implemented inside this node as a function. This latter is recalled by the state machine in the *ROW* state.

- **hough_transform.cpp**
This node estimates the relative position respect to the rows of trees, when the robot is navigating within them. Published and subscribed topics are summarized in table 4.13. Working principle and most important features

Published Topics	Message
/line_l	geometry_msgs::PoseStamped
/line_r	geometry_msgs::PoseStamped
/line_l_centre	geometry_msgs::PoseStamped
/line_r_centre	geometry_msgs::PoseStamped
/hough_pose	HoughPose

Subscribed Topics	Message
/scan_filtered	sensor_msgs::LaserScan
/state_row_nav	State
/state_row_test	TestState
/vs/commands	Commands
/n_hough_pose	HoughPose
/vs/row_references	RowReference
/vs/test_param	TestParam

Table 4.13: *hough_transform* node published and subscribed topics

of the *hough_transform* node are described here below.

  – If the *stop* or *reset* commands are received (*/vs/commands* topic), the node memory is cleaned.
  – When topics */vs/row_references* and */vs/test_param* are received, the node loads robot dimensions, really important for space perception and row estimation.
  – Through topics */state_row_nav* and */state_row_test*, the node knows whether the mission is paused or not and memorize this information.
  – The iteration frequency is regulated by means of a *timer* and can be defined at mission beginning
  – Code iteration is enabled only if the *row_state_machine* state is *ROW* or if the mission is paused, since the algorithm is designed for lines estimation when the robot is acting within rows, both manually or automatically. As soon as the code is running, the node communicates it is *active* setting a flag in the published message (see table 4.3).
  – The node can run in three different ways, depending on initialization conditions. Indeed, since the included internal model, the algorithm looks for a previous solution, around which carry on its research.
    1. Mission is starting, meaning that the robot is at the beginning of a row. In this case, just for *one* cycle, the algorithm research is

extended to the whole grid, both in terms of angle and distance.

2. The operative state is changing from *CHG* to *ROW*, meaning that the robot is finishing a row change and approaching a new row. In this case, just for *one* cycle, the algorithm extracts angles and distances from the message (*/n_hough_pose* topic) coming from the *n_hough* node, which provides localization outside the rows. In this way, when the state changes, *n_hough_transform* stops running and *hough_transform* starts, using its last message for reducing the research grid. Viceversa, when the state goes from *ROW* to *CHG*, *hough_transform* stops and its last message is exploited by *n_hough_transform*.
3. The mission within rows has already started. In this case the algorithm can look for a solution close to its previous one and so on for all the following iterations.

– Once the algorithm returns a solution, a couple of functions are recalled, in order to check for errors and discriminate left and right lines.

– Finally, the relative rows positions are published both on */hough_pose* topic, using the user-defined message *hough_pose*, and on other topics (see table 4.3), using standard message *geometry_msgs::PoseStamped*, in order to visualize the results on *rviz*.

- **n_hough_transform.cpp**

This node provides environment features extraction outside the rows, for localization during row change. Published and subscribed topics are summarized in table 4.14. Working principle and most important features of the

Published Topics	Message
/n_hough_row1	geometry_msgs::PoseStamped
/n_hough_row2	geometry_msgs::PoseStamped
/n_hough_row3	geometry_msgs::PoseStamped
/n_hough_row_norm	geometry_msgs::PoseStamped
/n_hough_pose	HoughPose

hfill

Subscribed Topics	Message
/scan_filtered	sensor_msgs::LaserScan
/state_row_nav	State
/state_row_test	TestState
/vs/commands	Commands
/hough_pose	HoughPose
/vs/row_references	RowReference
/vs/test_param	TestParam

Table 4.14: *n_hough_transform* node published and subscribed topics

n_hough_transform node are described here below.

– If the *stop* or *reset* commands are received (*/vs/commands* topic), the node memory is cleaned.

- When topics */vs/row_references* and */vs/test_param* are received, the node loads robot dimensions, really important for space perception and row estimation.
- Through topics */state_row_nav* and */state_row_test*, the node knows whether the mission is paused or not and memorize this information.
- The iteration frequency is regulated by means of a *timer* and can be defined at mission beginning
- Code iteration is enabled only if the *row_state_machine* state is *CHG* or if the mission is paused, since the algorithm is designed for lines estimation when the robot is acting outside rows, both manually or automatically. As soon as the code is running, the node communicates it is *active* setting a flag in the published message (see table 4.3).
- The node can run in three different ways, depending on initialization conditions. Indeed, due to the included internal model, the algorithm looks for a previous solution, around which carry out its research.
  1. A test is starting, with the robot placed just outside a row. In this case, just for *one* cycle, the algorithm research is extended to the whole grid, both in terms of angles and distances (*INIT*).
  2. The operative state is changing from *ROW* to *CHG*, meaning that the robot is finishing to travel a row and approaching a row change. In this case, just for *one* cycle, the algorithm extracts angles and distances from the message (*/hough_pose* topic) coming from the *hough* node, which provides localization within the rows. In this way, when the state changes, *hough_transform* stops running and *n_hough_transform* starts, using its last message for reducing the research grid. Viceversa, when the state goes from *CHG* to *ROW*, *n_hough_transform* stops and its last message is exploited by *hough_transform*.
  3. The row change has already started. In this case the algorithm can look for a solution close to its previous one and so on for all the following iterations.
- Finally, the relative rows positions are published both on */n_hough_pose* topic, using the user-defined message *hough_pose*, and on other topics

(see table 4.3), using standard message *geometry_msgs::PoseStamped*, in order to visualize the results on *rviz.*

- **extreme_row.cpp**
  This node detects whether the robot is entering or exiting a row, when it is outside or inside it, respectively. Published and subscribed topics are summarized in table 4.15. The most important features implemented in this

Published Topics	Message
/extreme_row	ExtremeRow

Subscribed Topics	Message
/scan_filtered	sensor_msgs::LaserScan
/hough_pose	HoughPose
/n_hough_pose	HoughPose
/state_row_nav	State
/state_row_test	TestState
/vs/commands	Commands
/vs/row_references	RowReference
/vs/test_param	TestParam

Table 4.15: *extreme_row* node published and subscribed topics

  node are:
  - Reception of *Reset* or *Stop* commands, from topic */vs/commands*, causes the node to clean its memory, getting it ready to restart the mission.
  - Through the */hough_pose* and */n_hough_pose* topic, the node receives its relative orientation respect to the rows, and can better detect the proximity of rows extremities.
  - Reading the topics */state* and *test_state* the node knows the state of the *row_state_machine* which supervises the mission. Basically, four behaviours are discriminated, based on the read state:
    1. ROW. If the robot is navigation within rows, the algorithm will look for the row exit
    2. CHG. If the robot is changing row, the algorithm will look for the row entrance
    3. PAUSE. When the mission is paused, the robot can be manually driven by the user. During this operation, the vehicle could both exit or enter rows. In order to avoid to miss these events, and cause algorithm failures once the mission is restored, the node keeps running, looking for entrances or exits based on the system state

at the moment in which the pause was called.

4. END. The *extreme_row* node stops running, since the mission is completed.

– The heart of the node is contained within the LIDAR data *callback*, meaning that at every message received on */filtered_scan* topic, corresponds an algorithm iteration. The code scans the laser scanner data, looking for row extremity features, and communicates if exit, entrance or nothing has been found through the */extreme_row* topic.

- **pose_estimation.cpp**

  This node reads sensors data and computes robot position, respect to the defined home position, and velocity. Published and subscribed topics are summarized in table 4.16. The most important features implemented in this

Published Topics	Message
/home_pose	mavros_msgs::HomePosition
/ned_pose	Kinematics
/ros_state	RosState

Subscribed Topics	Message
/scan	sensor_msgs::LaserScan
/arduino/gps_raw	mavros_msgs::RawGps
/arduino/imu_raw	sensor_msgs::Imu
/vs/encoders	RefSpeed
/hough_pose	HoughPose
/vs/commands	Commands
/vs/row_references	RowReference
/vs/test_param	TestParam

Table 4.16: *pose_estimation* node published and subscribed topics

node are described in the list below:

– The node reads the topic */vs/commands*, through which can receive the following commands:

1. *Reset*. Completely reset the node internal memory
2. *Stop*. The state informations related to the current mission are cleaned and then the node can accept new missions or restart the interrupted one.
3. *Use_gps*. Tells if the GPS coordinates have to be used for pose estimation, i.e. if the node has to read the sensor, send GPS to the HMI or expect the *home position* command.
4. *home_pose*. If the GPS is used in the current configuration, fix the curret coordinates as *home position*, i.e. origin of the *inertial* reference frame.

- When the GPS coordinates are received (from */arduino/gps_raw* topic), the node save and send them to the HMI, for global position visualization on the satellite map, and computes the robot position within the *NED* frame.
- From the *hough_transform* node (*hough_pose* topic), the relative position and orientation is received and used for sensor fusion and position estimation.
- The node receives also encoders feedback (*/vs/encoders* topic) and reads LIDAR and IMU (*/scan* and */arduino/imu_raw* topics), employing these data for position estimation.
- Once mission parameters are received from the HMI (topics */vs/row_references* and */vs/test_param*), the robot parameters are automatically load from the structures of parameters defined in the ROS environment (see chapter C).
- A sensor fault detection function is implemented in the node structure. Using timers and opportune flags, the node can detect if a sensor is not responding correctly or is disconnected. Any wrong situation or sensor fault is communicate to the HMI and the ROS system through the */ros_state* topic. Only if the system state is correct the mission supervisor (*/row_state_machine*) accept to start new missions or continue the ongoing one.

• **row_log.cpp**
The node subscribes the Robot Control topics, containing sensors data and informations about system state and current mission. Subscribed topics are summarized in table 4.17. The behaviour of *row_log* node is described here below.

- The node can work in three operative modes, which can be chosen specifying the right arguments when the executable is launched:
  1. AUTO. The automatic mode is designed for log collection during robot autonomous missions. Log writing starts when the mission is started and stops as soon as the mission is completed or stopped by the user.

Subscribed Topics	Message
/scan	sensor_msgs::LaserScan
/arduino/gps_raw	mavros_msgs::RawGps
/arduino/imu_raw	sensor_msgs::Imu
/hough_pose	HoughPose
/vs/commands	Commands
/vs/row_references	RowReference
/vs/test_param	TestParam
/state_row_nav	State
/state_row_test	TestState
/ned_pose	Kinematics
/home_pose	mavros_msgs::HomePosition
/ros_state	RosState
/extreme_row	ExtremeRow

Table 4.17: *row_log* node subscribed topics

2. MANUAL. This mode is designed for when the robot is manually driven and the user wants to collect data in the meanwhile. Log starts when the brake is deactivated, i.e. as soon as the robot supposed to be moved, and stops once the brake is activated.
3. ALONE. Allow to collect logs independently from the current mode or mission status. Log collection starts as the node is launched and stops when the node is quitted.

In *AUTO* or *MANUAL* mode, the log node can be kept active also for different mission, since after the log has been stopped, it can restart its activity when the brake is deactivated or a new mission starts, writing new files.

– Three kinds of data can be collected:
  1. Sensors. Basically consists in GPS and IMU data.
  2. Laser. This is nothing but the data coming from LIDAR sensor.
  3. Row. This means data regarding the current mission, such as *row_state_machine* status, numer of rows travelled, obstacle presence, and so on

  The user can choose to collect any group of this data types, rather than just one kind, through the executable arguments.
– Basically, when the brakes are deactivated or the mission starts (*/vs/commands* topic), the node creates new log files and starts to collect and write data.

Of course, in the first lines of the created files, proper comments are inserted, in order to make it easy to identify the kind of mode or mission, and the data collected below.

- **laser_scan_time_remap.cpp**
  The node basically subcribes the 2D laser data (/laser_scan topic) and replicates it in the published /scan topic, updating the messages timestamp. This function is useful when laser data comes from previously recorder *bag* files and its timestamp is older respect to the other system messages, causing *errors*. This time conversion allows to carry out simulations, based on recorded LIDAR data and easily test functions and algorithms. Published and subscribed topics are summarized in table 4.18.

Published Topics	Message	Subscribed Topics	Message
/scan	sensor_msgs::LaserScan	/laser_scan	sensor_msgs::LaserScan

Table 4.18: *laser_scan_time_remap* node published and subscribed topics

- **tf_broadcaster.cpp**
  This node publishes constant frames transformations. This function is useful to link the different robot frames. For example, a proper transformation is always published, between laser scanner data, expressed in the *velodyne* frame, and robot position, described by the *base_frame*.

4.5.3 User-defined Mavros Plugins

In this subsection, the plugins created for implementing communication with HMI and Arduino Mega 2560 microcontroller are presented, using the same method employed for nodes description. These executables provides bi-directional data exchange, between the ROS environment and the connected devices. Incoming data is received as *Mavlink* message and published as *ROS topic*. Viceversa, outgoing data is read by the *plugin* as *ROS topic* and sent as *Mavlink* message.

- **rover.cpp**
  Defines the whole data exchange between Robot Control and HMI. Incoming and outgoing messages correspondence is showed in tables 4.19 and 4.20.

In this case the communication channel is the LAN network to which the onboard computer is connected. The exchange of UDP messages implements the communication between *phisical* and *virtual* machines.

Received Mavlink Msg	Published Topic	Message
HEARTBEAT	/	/
RC_CHANNELS_RAW	/vs/commands	Commands
RAW_PRESSURE	/vs/encoders	RefSpeed
SYS_STATUS	/vs/system_param	SystemParam
RC_CHANNELS_SCALED	/vs/row_references	RowReference
RC_CHANNELS_SCALED	/vs/test_param	TestParam

Table 4.19: *rover* plugin incoming data flow

Sent Mavlink Msg	Subscribed Topic	Message
SYS_STATUS	/state_row_nav	State
SYS_STATUS	/state_row_test	TestState
HOME_POSITION	/home_pose	mavros_msgs::HomePosition
LOCAL_POSITION_NED	/ned_pose	Kinematics
GPS_RAW_INT	/arduino/gps_raw	mavros_msgs::RawGps
COMMAND_INT	/driver_commands	RefSpeed
SERVO_OUTPUT_RAW	/ros_state	RosState
SERVO_OUTPUT_RAW	/hough_pose	HoughPose
SERVO_OUTPUT_RAW	/n_hough_pose	HoughPose

Table 4.20: *rover* plugin outgoing data flow

- **arduino.cpp**
  Defines the whole data exchange between Robot Control and Arduino Mega 2560 microcontroller. Incoming and outgoing messages correspondence are showed in table 4.21. In this case the communication channel is the *USB* wire connecting PC and microcontroller. Mavlink messages are exchange by means of a serial communication.

Received Mavlink Msg	Published Topic	Message
RAW_IMU	/arduino/imu_raw	sensor_msgs::Imu
SCALED_PRESSURE	/arduino/temperature	sensor_msgs::Temperature
GPS_RAW_INT	/arduino/gps_raw	mavros_msgs::RawGps

Table 4.21: *arduino* plugin incoming data flow

4.5.4 ROS Packages

In this subsection *ROS packages* composing the *Robot Control* structure are presented. For the user-defined packages, function, contained nodes, defined *launch* and *yaml* files [18] will be described. Differently, for downloaded ones, just function and launch files will be mentioned.

User-defined Packages

In this subsection the user-defined packages of the Robot Control are described.

- **hough_transform**
  This package contains the nodes providing for robot localization respect to the orchard structure, both within and outside rows, which are:
  - hought_transform.cpp
  - n_hought_transform.cpp

- **row_navigation_task**
  This package containes the node providing for row extremities detection, position estimation, Robot Control management and log collection. The executables of this package are:
  - extreme_row_node.cpp
  - pose_estimation.cpp
  - row_log.cpp
  - row_state_machine.cpp

  All the user-defined messages (see subsection 4.5.1) of the implemented *ROS* structure are defined within this package. Moreover, it contains also some *launch* files, by which the user can start robot localization, mission management and position estimation.
  - **extreme_row_hough.launch**. Starts robot localization and rows extremities detection. Recalled nodes are listed in table 4.22.
  - **row_nav.launch**. Launches Robot Control management and position estimation (see table 4.23).

Node	Package
tf_broadcaster.cpp	*laser_data_processor*
extreme_row_node.cpp	*row_navigation_task*
hought_transform.cpp	*row_navigation_task*
n_hought_transform.cpp	*row_navigation_task*

Table 4.22: Nodes recalled by *extreme_row_hough.launch* file

Node	Package
pose_estimation.cpp	*row_navigation_task*
row_state_machine.cpp	*row_navigation_task*

Table 4.23: Nodes recalled by *row_nav.launch* file

In the launch folder is also contained the file **row_nav_param.yaml**, in which useful parameters for robot localization, navigation and mission management are stored. In particular:

- Orchards data structures (number of rows, lenght, useful GPS positions, kind of culture, row width, ...)
- Robot data structures (weight, dimensions, onboard sensors, ...)
- Parameters for etremities detection
- Parameters for *hough* and *n_hough* algorithms
- Low-level control loop gains and parameters

These parameters are stored in the *yaml* file and loaded in the *rosParam server* [14] when the launch files are recalled. In this way the parameters are available for the started nodes and can be read from the server during the node initialization. Of course the nodes load parameters on the basis of the user commands, i.e. selected robot, orchard, and kind of mission.

• **laser_data_processor**
This package provides auxiliary services for the Robot Control functioning, like time remapping of recorded data or frames transformation broadcasting. The most important contained executables are:

- laser_scan_time_remap.cpp
- tf_broadcaster.cpp

• **laser_launch**
This package containes just launch files, by which different functionalities,

like laser data conversion from 3D to 2D, HMI and microntroller communication can be easily recalled. Within the package, launch files are collected in folders, on the basis of their functions.

1. laser_scan_from. This group of launch files takes care of LIDAR data and related conversions. In particular:
   - *ls_from_velodyne_online.launch.* This functionality provides 3D (*/velodyne_pointcloud* topic) and 2D (*/scan* topic) laser scanner data, coming from the sensor. LIDAR reading function and 3D to 2D laser data conversion (see table 4.24) are recalled.
   - *ls_from_velodyne_offline.launch.* This functionality provides 3D (*/velodyne_pointcloud* topic) and 2D (*/scan* topic) laser scanner data, coming from previously recorded bag file. 3D to 2D laser data conversion and time remapping (see table 4.25) are recalled,
   - *ls_from_ls_offline.launch.* This functionality provides 2D (*/scan* topic) laser scanner data, coming from previously recorded bag file. Just the time remapping function (see table 4.26) is recalled.

   Conversion parameters are stored in the file *3d_to_2d_param.yaml*, recalled in any of the three presented launch files. Contained parameters define which portion of the 3D pointcloud have to be converted in planar form, in terms of distance ranges and angles.

Node/launch file	Package
VLP16_points.launch	*velodyne_pointcloud*
pointcloud_to_laserscan_node.cpp	*pointcloud_to_laserscan*

   Table 4.24: Nodes recalled by *ls_from_velodyne_online.launch* file

Node	Package
pointcloud_to_laserscan_node.cpp	*pointcloud_to_laserscan*
laser_scan_time_remap.cpp	*laser_data_processor*

   Table 4.25: Nodes recalled by *ls_from_velodyne_offline.launch* file

2. mavlink_com. Thiese launch files provide communication between Robot Control and external devices. In particular:

Node	Package
pointcloud_to_laserscan_node.cpp	*pointcloud_to_laserscan*
laser_scan_time_remap.cpp	*laser_data_processor*

Table 4.26: Nodes recalled by *ls_from_ls_offline.launch* file

- *arduino_com.launch.* Launches the serial communication between microcontroller and *ROS* environment. An instance of the mavros node (see table 4.27) is recalled.
- *vs_udp_com.launch.* Starts UDP communication between HMI and *ROS* environment. An instance of the mavros node (see table 4.27) is launched.

Node/launch file	Package
node.launch	*mavros*

Table 4.27: Nodes recalled by *arduino_com.launch* and *vs_udp_com.launch* files

In both cases two configuration files contained in the folder are recalled, the first for parameters and configuration (COM port, IP address, baude rate and so on) and the second defining the plugins to launch (see 4.5.3). *arduino_com.launch* recalls files *arduino_com_config.yaml* and *arduino_com_pluginlists.yaml*, while *vs_udp_com.launch* reads *vs_udp_config.yaml* and *vs_udp_pluginlists.yaml.*

3. visualization. Recalls some tools for ROS system analysis and output monitoring.
   - *visualization.launch.* Launches some tools which allow to monitor system outputs. In particular, the user can see frames transformations, topics content, a node graph and graphically visualize laser pointcloud and frames position (see table 4.28).

Node/launch file	Package
rqt_graph	*rqt_graph*
rqt_tf_tree	*rqt_tf_tree*
rqt_topic	*rqt_topic*
rviz	*rviz*

Table 4.28: Nodes recalled by *visualization.launch* file

When this launch file is recalled, the configuration file *rviz_config.rviz* is loaded, containing the *rviz* configuration parameters, like fixed frame for visualization, topics to graphically represent and other details.

Downloaded Packages

In this subsection the downloaded ROS packages employed in the Robot Control are briefly described.

- **mavros**
  This package provides communication between *ROS* environment and external devices, connected via *serial* or *UDP* connection. As the name suggests, *Mavros* deals only with devices using *Mavlink* communication protocol. The heart of the system consists of the *Mavros* plugins, executables which implements the exchange of *Mavlink* messages, on the external side, and *ROS topics*, on the inner side (see subsection 4.5.3).

- **pointcloud_to_laserscan**
  This package provides conversion from 3D to 2D LIDAR data, i.e. from *PointCloud2* to *LaserScan* messages, obtaining a kind of environment top view. The contained executables are:
  - pointcloud_to_laserscan_node.cpp
  - pointcloud_to_laserscan_nodelet.cpp

  Basically, just an *horizontal slice* of the 3D pointcloud is considered, on the basis of range parameters, and filtered, obtaining planar data. In particular, the most important conversion parameters allows to:
  - Define maximum and minimum distance ranges
  - Select maximum and minimum height to consider
  - Define circular area of the planar data, i.e. the *viewing angle*

  Planar laser scanner data is foundamental for localization algorithms, which employes this information for row lines estraction.

- **velodyne**
  This package is used for reading LIDAR sensor. It acquires laser scanner data from the ethernet port and publishes in the *ROS* Robot Control the */velodyne_pointcloud* topic, containing the 3D *pointcloud*.

4.5.5 Robot Control launch

In this subsection the procedure for Robot Control launching is described. Actually, the *launch* files related to the single components have already been presented with the different ROS packages in subsection 4.5.4. Therefore, describing the whole launching procedure is just a matter of combining the single components. Nodes or launch files needed for a correct and complete Robot Control activation are:

- *roscore*, for starting the ROS system.
- *arduino_com.launch*, for communication with the Arduino microcontroller and get sensors data.
- *vs_udp_com.launch*, for starting communication with the HMI.
- *ls_from_velodyne_online.launch*, providing 3D and 2D LIDAR data to the ROS system.
- *extreme_row_hough.launch*, starting localization and rows extremities detection algorithms.
- *row_nav.launch*, running position estimation algorithm and the row_state_machine, for *Robot Control* and mission management.
- *visualization.launch*, for debugging or results visualization.
- *row_log.cpp*, recording log files about sensors data and Robot Control operations.

In order to clarify connections and interactions between the different nodes of the proposed software structure, a scheme is presented in figure 4.11. For first, in the top of the scheme there are the two Mavros *plugins* (red circles):

- Rover (see 4.5.3), which interfaces the ROS environment with the HMI. *Commands* represents incoming data (see tables 4.19, while *System status* is the state of the robot, sent to the HMI (see table 4.20).
- Arduino (see 4.5.3) interfaces the Robot Control with the external GPS and IMU sensors. Outgoing data is represented by *GPS, IMU* (see table 4.21).

At the bottom of the picture compare the nodes related to LIDAR data acquisition and filtering (violet circles):

- Velodyne (see 4.5.4), which receives 3D LIDAR data. *3D pointcloud* represents data stream coming from the sensor.
- PC to LS (see 4.5.4) receives 3D LIDAR data (*3D pointcloud*), which is

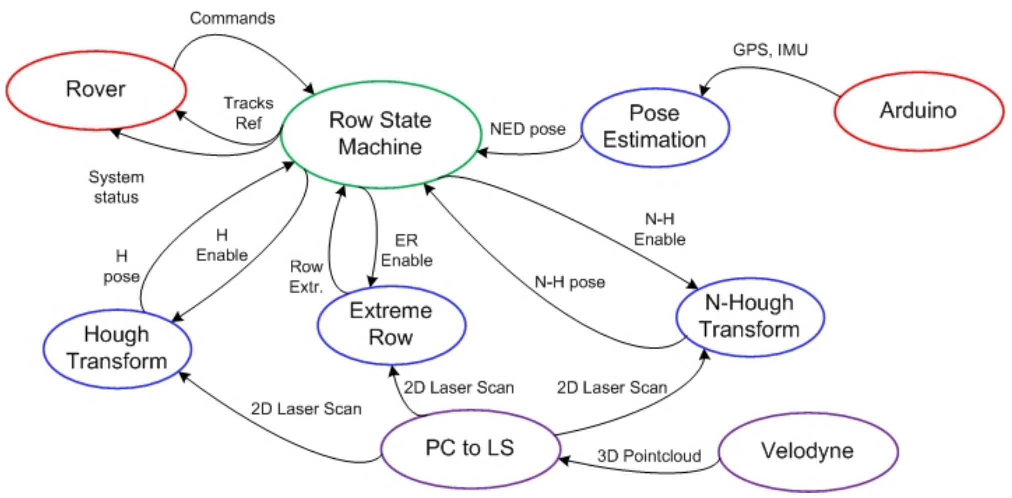


Figure 4.11: Functional scheme of the most important ROS nodes implementing the Robot Control

filtered and converted in 2D laser scanner data (*2D Laser Scan*).

Then, the nodes in the middle implement localization functions (blue circles):

- Pose Estimation (see 4.5.2), which receives *GPS, IMU* data (see table 4.16) and gives back the robot position in the inertial frame (*NED pose*).
- Hough Transform (see 4.5.2), which receives *2D Laser Scan* data and an enable signal (*H enable*) from the Row State Machine and computes lateral rows position respect to the robot (see *H pose* in the scheme). Table 4.13 reports all the ROS topics subscribed and published by the node.
- N-Hough Transform (see 4.5.2), which receives *2D Laser Scan* data and an enable signal (*N-H enable*) from the Row State Machine and computes the position of the surrounding rows during the row change (see *N-H pose* in the scheme). Table 4.14 reports all the ROS topics subscribed and published by this node. *GPS, IMU* data (see table 4.16) and gives back the robot position in the inertial frame (*NED pose*).
- Extreme Row (see 4.5.2) receives *2D Laser Scan* data and an enable signal, *ER enable*, from the Row State Machine and checks if the robot is entering a new row or exiting from the current one (see table 4.15).

Finally, in the middle-high part of the scheme there is the core of the ROS structure (green circle):

- Row state machine (see 4.5.2) receives robot position, both global and local, and user directives, and gives back the reference velocities for the robot tracks (see *Tracks Ref* in the scheme), making it possible to navigate within the orchard rows and carry out the assigned missions. See table 4.12 for the details about published and subscribed topics.

4.6 Conclusions

In this chapter, software structure implemented on the robot on-board computer, composed of user interface and control system, has been described. At this point the system architecture has been completely defined, in terms of control algorithms (see chapter 3), software implementation (chapter 4), and prototype development (chapter 2). In the following chapter, the carried out experimental tests are presented, in terms of goals, robot setup and, of course, results.

Chapter 5

Experimental tests

In this chapter the practical results obtained during the research work are described. In particular, experimental goals, setup, and results, about both robot motion and autonomous navigation, obtained in the field will be set out.

5.1 Introduction

In the previous chapters a navigation algorithm has been proposed, for robot autonomous movement within a structured agricultural environment. Then, also an experimental platform has been described, developed with the aim of testing and validating the robot control law and the implemented software scructure.
The goal of this chapter is to describe the experimental tests carried out in the field and present the obtained results. For first, the goals of this activities are listed and deepen (see section 5.2), in order to motivate the experimental tests and define the most important opints and concept behind them. Then, a description of the experimental setup is proposed (see section 5.3), showing both the robot setup for the different tests and the environment conditions. Finally, the carried out experiments are considered, and the related results and considerations are given (see 5.4).

5.2 Goals

The aim of the experimental activities is the validation of the different subsystems composing the developed agricultural platform in a realistic working environment. In order to motivate the carried out tests and properly organize both experiments and results analysis, the most important robot features and components are described, focusing on possible defects or failures.

- *Mechanical subsystem*
  This subsystem is composed by the frame, the tracks, and the quadrilateral structure for tools positioning. It is important to test the frame robustness, as far as the center of gravity position, which is affected by the battery position and the presence of any tool mounted in front of the vehicle and determines the robot stability and traction capacity.
- *Electric subsystem*
  In order to define the electric and mechanical power characteristics of the robot, it is foundamental to carry out experiments in which the vehicle acts in different ground and speed conditions. In this way, for first the traction capacity can be tested, as far as the motor size. Inverters and motors temperature can be analysed, in order to see the effort that this components are bearing during the motion. Moreover, the power consumption can be calculated, in order to make it possible to estimate the robot endurance.
- *Software structure*
  All the developed software structure needs to be tested in realistic conditions, since it allows to interact with the robot and control it. In particular, the HMI, Robot Control and communication will be tested, looking for missings or bugs.
- *Navigation algorithm*
  Of course, once the robot is working properly, the localization and control algorithm needs to be tested, in order to see its capacity to control the robot in different conditions and verify its robustness.

The most important features or components have been listed, highlighting possible defects or critical behaviours. In view of this reasoning, a table of outdoor tests have been defined, in order to cover all the features which need to be validated. In the next section (see 5.3) these experiments are described, and the experimental setup for the robot is presented.

5.3 Experimental setup

The goal of this section is to present the robot setup for outdoor experiments, and, of course, to describe the carried out tests, whose results are showed in section 5.4. Figure 5.1 shows how the robot has been prepared for the experimental

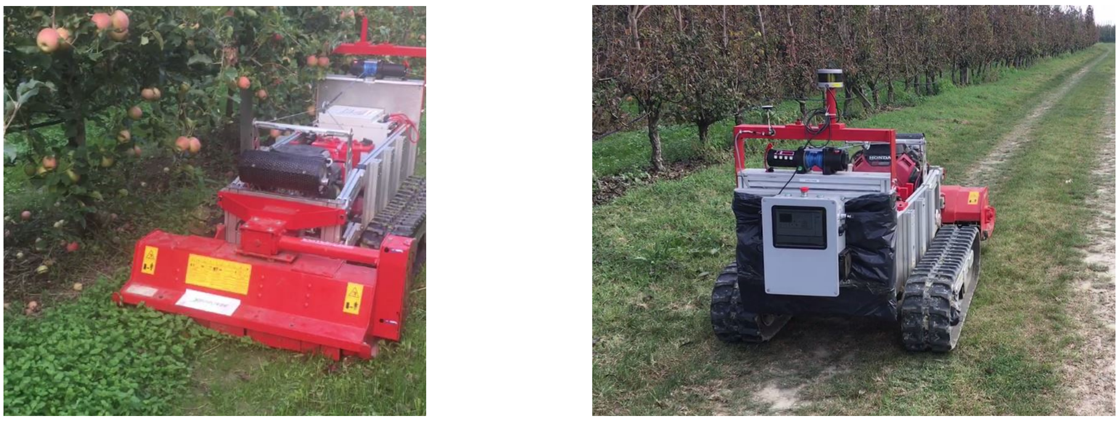


Figure 5.1: Robot equipped with mower (left) and then with LIDAR and low cost GPS (right)

tests. As a sensor suite, the vehicle is equipped with a LIDAR sensor, for providing environment perception, and low-cost GPS receiver, in order to follow robot position on the HMI satellite map and estimate the travelled distance during the experiments. Since in this configuration the battery has only to provide for rover locomotion, the robot is supplied by a single Lithium battery, which is expected to guarantee at least an endurance of some hours. In this way, these tests session will show the robot performances with the minimum amount of energy stored onboard (other sessions are already scheduled, in which the vehicle will be equipped with multiple batteries). In front of the rover a mower is mounted, in order to carry out experiments in the most realistic conditions as possible. Indeed, the tool affects:

- Robot equilibrium, since all its weight is concentrated on the front of the vehicle.
- Steering capacity, caused by the friction between mower and ground and the

consequently torque which opposes the steering action.

- Robot weight, affecting the electric and mechanical power consumption.

In this way, the test results will define the power consumption and the robot steering capacity, with the mower mounted onboard. Of course, through the winch, tool height can be regulated, giving the possibility to test rover effort and power consumption when the mower is placed on the ground rather than lifted. The mechanical power needed by the tool is provided by the endothermic motor, supplied by a gas can.
In order to validate the different subsystems (see section 5.2), from basic features, like simple movements, up to autonomus navigation algorithms, the following tests have been scheduled and carried out:

- Straight and curvilinear movements, at different velocities, for testing torques and requested powers.
- Repetition of the first point with the mower mounted onboard, in order to verify how it affects the power consumption and the robot behaviour.
- Repetition of the first point in different ground conditions, from asphalt to grass, wet grass and mud, in order to see the variation of friction between tracks and terrain and the difference in power requests.
- Durability tests, in which the robot is driven for hours in the field, with the aim of testing the robot endurance, the battery capacity and behaviour during discharge, and verify possible overheatings or malfunctions which do not come out during shorter work sessions.
- Experiments in which the robot is manually driven through the orchard in different kinds of cultivations and the localization algorithm is launched. The aim is to test the reliability of the rows estimation algorithm, before letting the robot to drive itself.
- Finally, once all the components are tested, the whole autonomous navigation system can be activated and tested.

Of course, during all this tests, the software structure, HMI and communication is implicitly verified and bugs and malfunctions are found and corrected on the way. The goal of first experiments is to test mechanical and electric subsystems, i.e. traction capacity, endurance and power consumption. For this reason, during this sessions the robot is manually driven by means of a joypad and just the HMI is launched, while the *Robot Control* is kept switched off. Differently, during autonomous navigation tests the Robot control is active and automatically drives the rover.

5.4 Experimental results

The experimental activities described in section 5.3 have been carried out in october 2018 in an agricultural property in Masiera, a small town near Ravenna, in Italy. It this section the most important experimental results are presented.

Since single tests involve different robot subsystems or features, their description one by one would be quite repetitive and not well-structured. For this reason, just the some foundamental results of the most significant experimental tests are presented, organizing the discussion from the simplest tests to the hardest ones, where robot efforts are pushed to the limit of nominal performances. In particular, the tests described in this section are:

1. **Straight test at constant speed**
2. **Straight test at variable speed**
3. **Steering test**
4. **Stress test in loose soil**
5. **Autonomous navigation within rows**

As introduced in section 5.3, the results are presented starting from the most basic ones, which regards robot moving and traction capacity (see section 5.4.1), and finishing with the most complex ones, like autonomous navigation results (section 5.4.2).

5.4.1 Electric and mechanical features

Straight test at constant speed

This test is carried out moving the robot within different orchard rows, basically mantaining *straight trajectories* and keeping a *constant speed.* The ground is partially wet and covered of *grass*, even if this condition is not so important in this case, since there are no curves and ground friction effects are limited. During this test the mower is mounted onboard, but kept lifted over the ground. Figure 5.2

shows the first important refsult of this section, i.e. *reference speed tracking.* As can be seen, the speed is expressed in km/h rather than in rmp at the motor shaft, in order to give a more realistic idea abou what the robot is doing. This mission lasts for $1800s$, i.e. an half an hour. Neglecting some moments in which the robot is stopped or a curve is asked, the speed is kept constant at about $2.8\,km/s$. In this condition, the motor shaft rotation speed can be calculated as

$$n_{motor} = v_{track}\,\frac{1}{3.6\,wheel_{rad}}\,red_{ratio}\,\frac{60}{2\,\pi} = 2.8\,\frac{32\;60}{3.6\;0.2\;6,28} = 1200\;rpm \qquad (5.1)$$

This value is at the middle of the robot capacities, since the *nominal* motors speed is $3000\,rpm$. As can be seen in figure 5.2, the left track speed is positive, while the

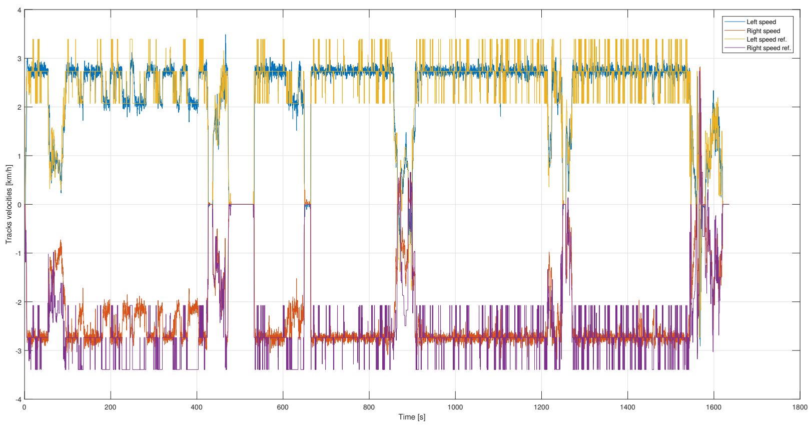


Figure 5.2: Reference speed tracking during straight trajectory at constant speed

right one is negative. This is not an error, but a real operative condition caused by the fact that tracks structures are identical and mirrored respect to the longitudinal axes of the robot, and of course the same happens to the motors. This means that at the left side a positive, i.e. counterclockwise, rotation speed, leads to track positive speed, while at the right side a negative speed, i.e. clockwise, gives a track positive speed.
Figure 5.2 shows the good *speed tracking* behaviour of the robot tracks, proving, as a first results, the accuracy of encoders and drivers parameters tuning, providing the capacity to mantain the desired velocities to the robot.
Figure 5.3 shows the mechanical power consumption at the tracks during the same

experimental test. Power values are calculated as product of motor shaft rotation speed, measured by the encoders, and mechanical torque produced by the motor itself. The latter is the product of injected current, measured by the *drivers*, and its torque constant, which is $0.1\,Nm/A$.
The power at left and right track, except in some moments, appears to be the same, meaning that the robot is well equilibrated, in terms of weight, structure and locomotion system. The consumed power is about $500\,W$ for the two motors, leading to an averall consumption of $1KW$, which is for sure an acceptable value, abundantly within the nominal conditions range. At the part of the test, the left

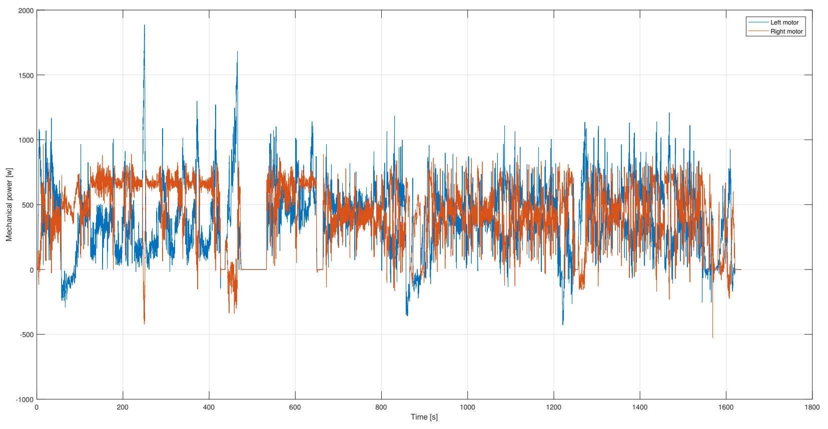


Figure 5.3: Mechanical power asked by the tracks during straight trajectory at constant speed

motor is consuming less power than the other one, effect which can be caused by slightly different ground conditions or steering corrections asked by the human pilot. In a few points of the graph the powers assumes negative values, meaning that the robot is generating energy. These are exactly the moments in which the robot is asked to suddenly reduce its speed and the electric machines applies a negative torque in order to slow down, working as generators. This is a really important feature, because it allows to *recover* energy and recharge the battery, in any situation in which the robot is asked to brake or slow down, as can happen when working in the hills and come down a slope terrain. Moreover, this test highlights the system good behaviour in energy regeneration conditions and confirm, from this point of view, the correct tuning of the motor drivers parameters.
Finally, figure 5.4 shows a comparison between mechanical and electric power con-

sumed by the robot, which values remain around 1 KW. The graph demonstrates

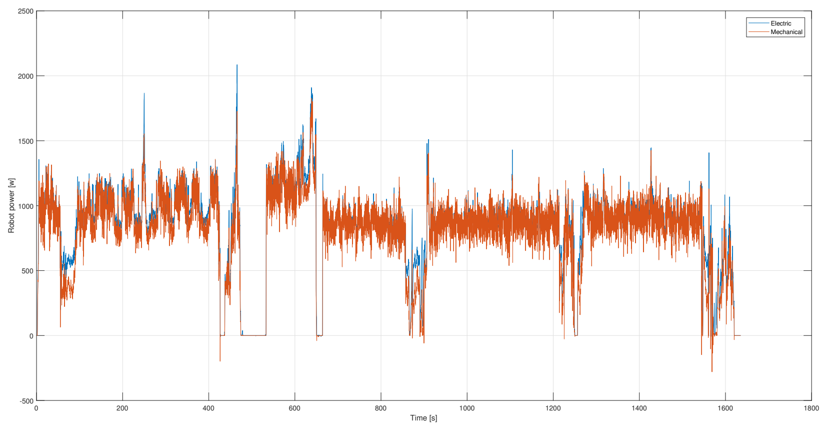


Figure 5.4: Mechanical and electric powers asked by the robot during straight trajectory at constant speed

that the two power values are similar, showing that the efficiency is quite high and in particular close to one. This means that the power management carried out by the driver is actually correct, at least in this conditions. Of course some points are visible, in which the mechanical power is visibly less than the electric one. This highlights a performance worsening, expecially in those cases in which the reference speed is changed, and a better behaviour when the robot is working in constant conditions.

Straight test at variable speed

This test is carried out moving the robot on a straight path at *different velocities*, i.e. 6, 4 and 2 km/h. Each reference speed value is kept for about a minute, than the robot rests for a while and so on. The mower is now leaned on the ground, in order to test robot tracking performances and behaviour when trying to track the reference speed while dealing with ground friction effects caused by the tool. Figure 5.5 shows a very good behaviour in terms of reference speed tracking, even when the motors are about to reach their nominal speed threshold, i.e. 3000 rpm.
Figure 5.6 reports motor *torques* evolution during the test. Since the right motor rotates at a negative speed for giving a positive right track velocity (as already described), of course its torque is negative, too. The graph shows a quite *stable*

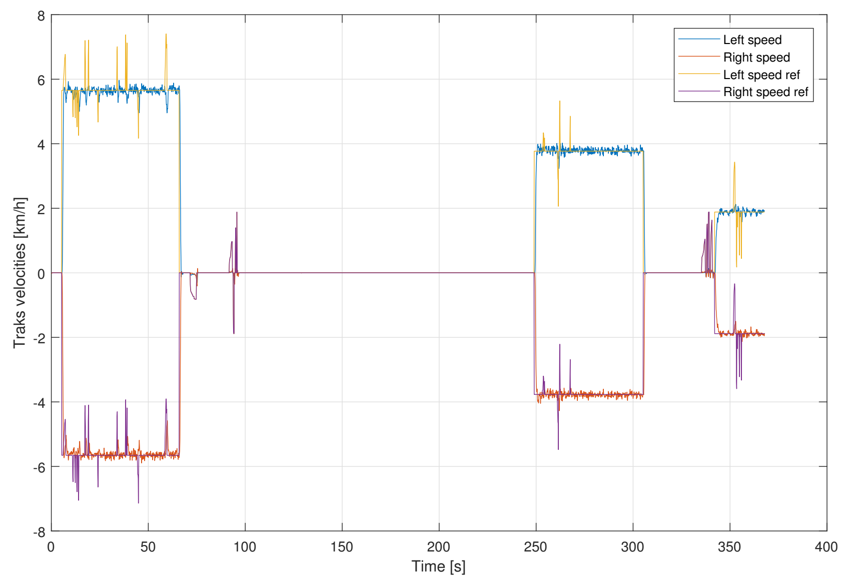


Figure 5.5: Reference speed tracking during straight trajectory at different speed values

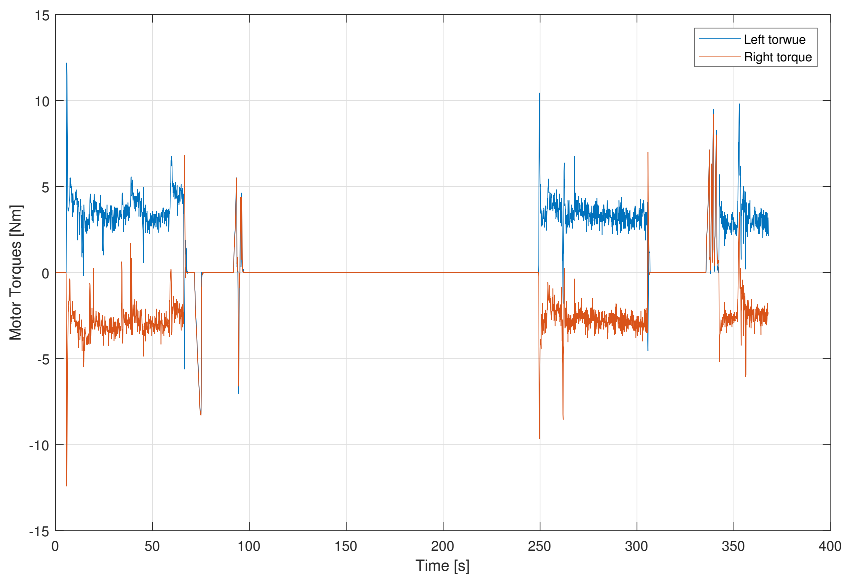


Figure 5.6: Motor torques during straight trajectory at different speed values

behaviour in terms of torque, meaning that the values are more or less the same at different speed conditions. This behaviour is quite expected, in the sense that the torque needed to mantain the desider speed is more or less the same, while of course the mechanical power increases with the velocity, and highlights that friction effects related to the speed are quite limited. The torques, at different speeds, remains around a value of $4\,Nm$. When the reference steps are applied to the robot (at 5 or 250 s, for example), there are of course some positive torque peaks, needed for accelerations. Differently, in the moments in which the robot stops (i.e. 70 or 310 s), the torque peaks are negative, for deceleration, and the robot is generating energy.

Figure 5.7 shows the battery voltage evolution during the test. The nominal

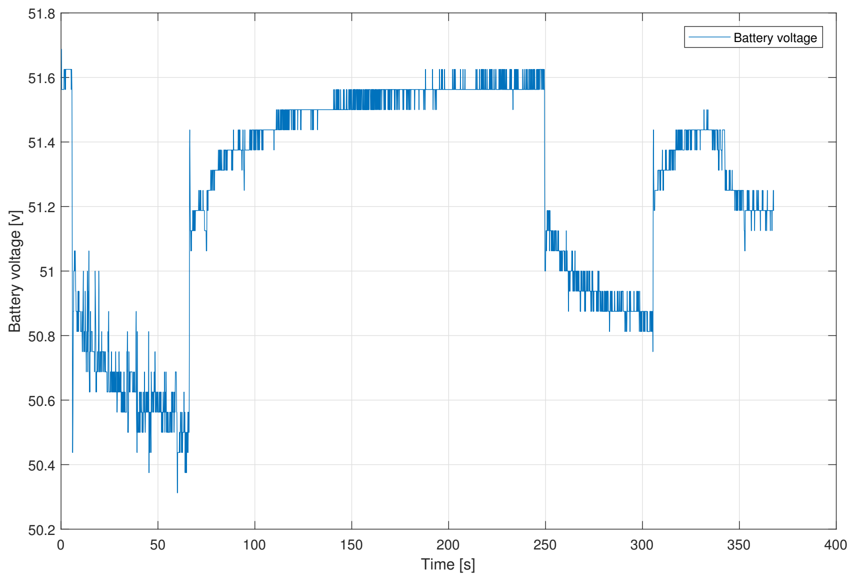


Figure 5.7: Battery voltage during straight trajectory at different speed values

voltage level is about $51\,v$, while the complete charge value is $54\,v$, meaning that the test is carried out in the middle of this two conditions, with a voltage level around $51.6\,v$. It is very interesting to see the *charge* and *discharge* transients, related, respectively, to the time periods in which the robot asks energy to the battery, or it rests, with the voltage returning to its previous level. During the major effort period, i.e. when the robot asks the maximum power to the battery, its voltage level goes under the nominal threshold, reaching a minimum value of $50.4\,v$.

Finally, in figure 5.8 are reported the *temperature* values of the motor drivers. The initial offset between the two temperature values is due to previous experi-

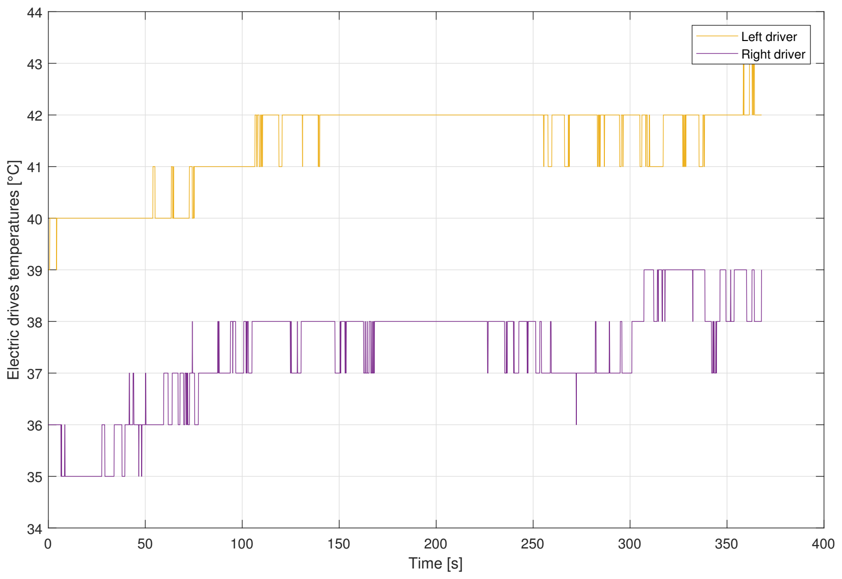


Figure 5.8: Drivers temperatures during straight trajectory at different speed values

ments, in which the left driver has been forced more that the right one. Anyway, the graph shows a quite identical behaviour for the two temperatures, fitting with the test straight trajectory conditions, in which the tracks should be subject to the same, or very similar, effort. Moreover, in the central time period, when the robot is stopped, the values are more or less constant, while increase when the vehicle is moving.

Steering test

This test consists in a *steering* maneuver, in which the robot starts to move forward and then is forced to steer for a certain period. Figure 5.9 shows an instance of this test. As can be seen in the picture, the mower is lifted from the ground. The experiment is quite fast, since the duration is about 100 s, i.e. a minute and an half. Figure 5.10 shows the reference speed tracking during the test. In the first part, from 50 to 70 s, the robot is moving forward, with the tracking moving at the same velocities. Then, a steering towards the right direction is imposed, from 70 to 85 s. Indeed, while the left speed increases, the right one decreases and goes next to zero. Since the two speeds are not equal and opposite, the robot is not steering on its place, but it is describing a curvilinear trajectory.

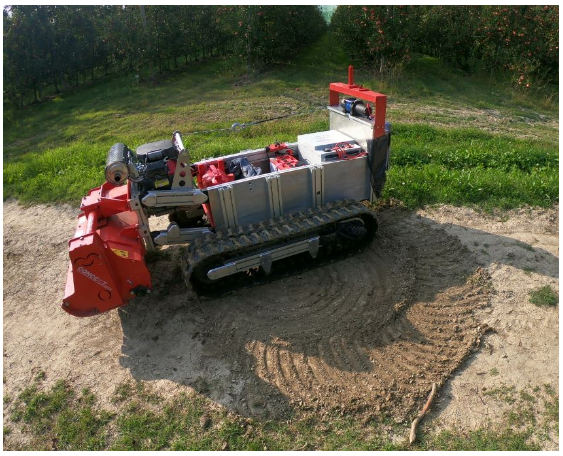


Figure 5.9: Instance of the steering test

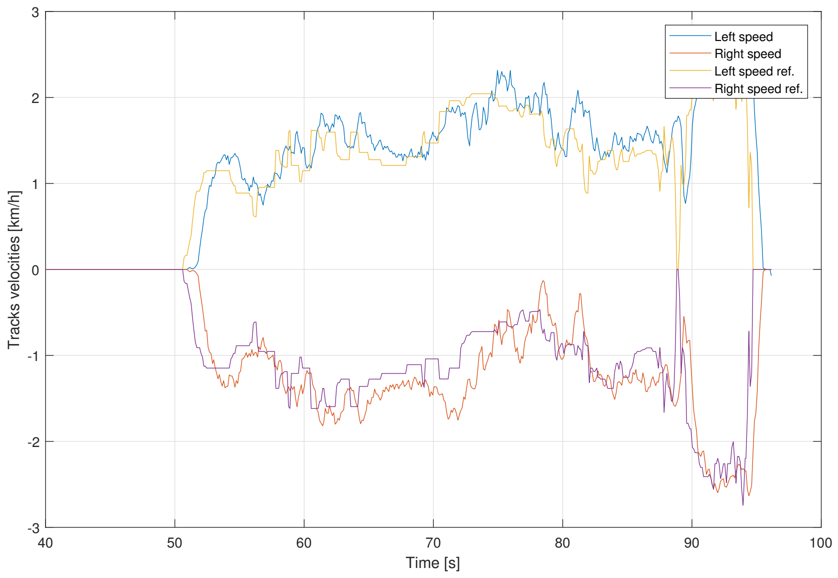


Figure 5.10: Reference speed tracking during steering maneuver

In this case the trajectory tracking appears to be worst, respect to the previous cases. This is true, and related to two factors:

- This test is chronologically prior to the previous ones, and in that moment the torque ramps set inside the drivers had a minor slope.
- The steering maneuver is really power expensive for a tracked vehicle, since the tracks slither on the ground and are subject to a very high frinction.

Figure 5.11 reports the torques on left and right tracks during the maneuver. It

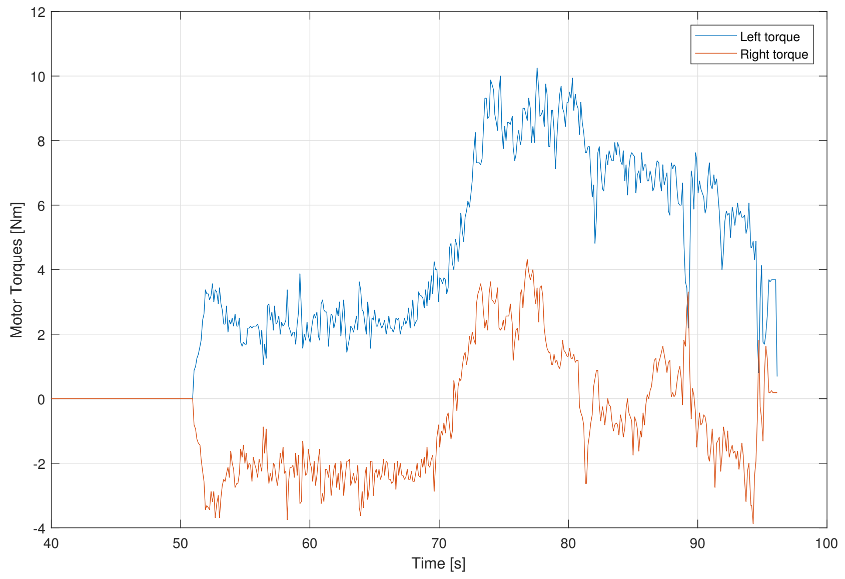


Figure 5.11: Motor torques during steering maneuver

is interesting to see that in the first part the torque values are opposite and quite lower, since the robot is moving forward, while in the second part assume a quite particular behaviour. The left torque increases, in order to let the track mantaining its speed, while the right one remains lower, since the right track has to mantain a lower speed, and becomes *positive*, even if the motor speed is $\underline{\text{negative}}$. This is due to a particular feature of tracked vehicles. Basically, during a steering on the place, the tracks have to move in opposite direction. But one track tends to be dragged by the other one, and so, in order to mantain its desider speed, have to break, applying a negative torque. This interesting effect is visible in the graph between 70 and 80 s, where the right torque reaches positite values.
Finally, figure 5.12 highlights the elctric power behaviour, showing the difference between straight and curvilinear trajectory. In the first case, the power values are around $200\,W$, while during the steering the most stressed motor reaches $1.2\,KW$. As expected, the left track is actively consuming electric power, while the right one, during the steering, tends to absorb a negative power, i.e. *generates* energy.

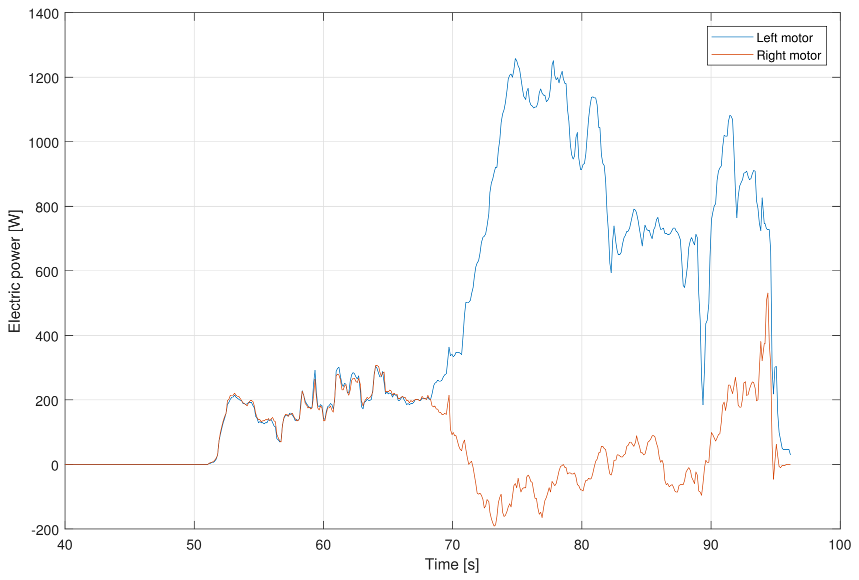


Figure 5.12: Electric powers asked by the robot during steering maneuver

Stress test in loose soil

This test has the aim of testing the robot traction capacity and the system behaviour when reaching its speed and power limits. This test is carried out in a loose soil field, where the ground is more soft and the robot traction capacity is put at risk. Here, the robot is asked to carry out some straight paths at different speeds, starting from $0.6\,km/h$ and reaching almost $3.8\,km/h$. Between the sessions, the robot is stopped for more or less $30s$, in order to better distinguish the different behaviour. The test is quite long, with a whole duration of about $900\,s$.

The peculiarity of this experiment is that the mower is lowered, completely leaning on the ground. In order to give an idea about environment and test conditions, figure 5.13 shows two frames from the ongoing test, with the robot moving at low speed (left side) and high speed (right side). Due to its particulat nature, the terrain tends to accumulate in front of the mower and stop the robot forward movement. Figure 5.14 shows the reference velocity tracking of left and right tracks, proving that, even if the speed signal is quite disturbed, due to the effort, the desired value is mantained also is these extreme conditions. An idea of the tracks efforts is given in figure 5.15, where the motor torques are reported. It is

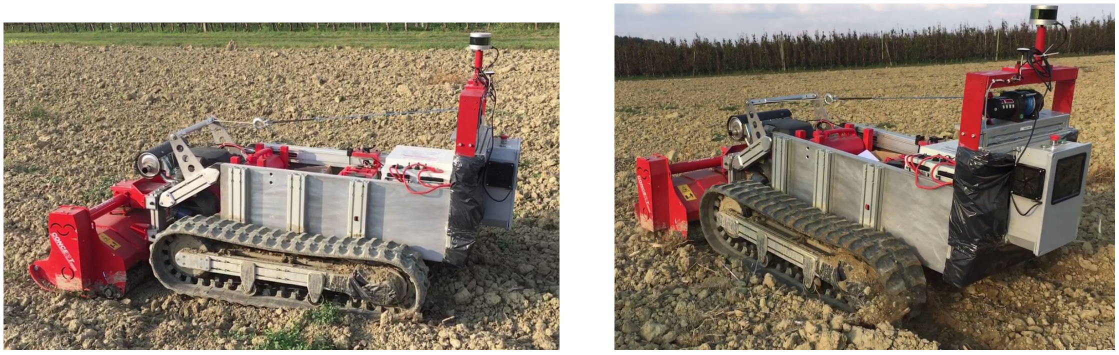


Figure 5.13: Images from the experimental test, with the rover moving at low (left) and high speed (right)

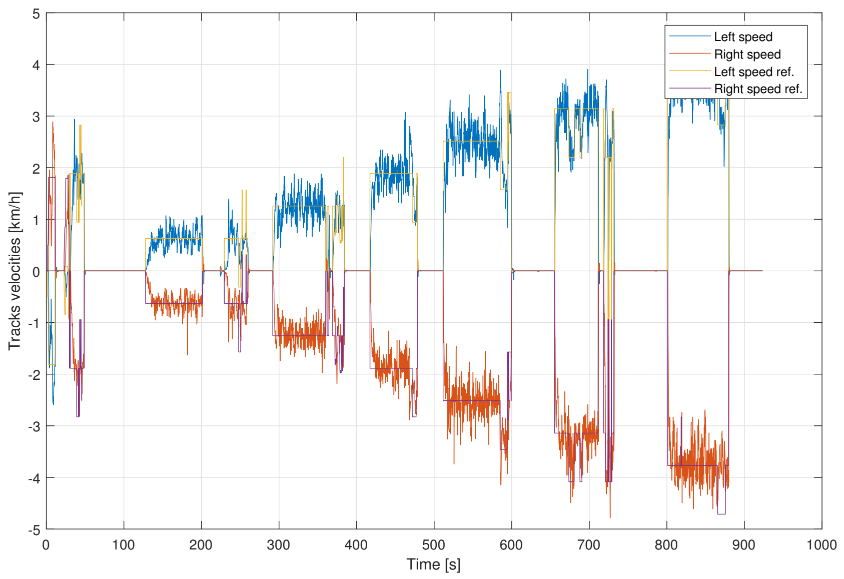


Figure 5.14: Reference speed tracking during stress test

interesting to see that the torques are more or less constant from one speed value to the others, and reaches values of 15 Nm. This numbers are within the safety limits of the motors, but way above the nominal values, meaning that the robot can keep this working conditions only for a limited period of time. Of course this is just a test and the normal operative conditions asked to the robot are very far from these ones, meaning that this results does not limit the expected application field. Differently from the torque, mechanical power increases linearly with the speed. Figure 5.16 shows the mehanical power evolution during the different tests. As can be seen in

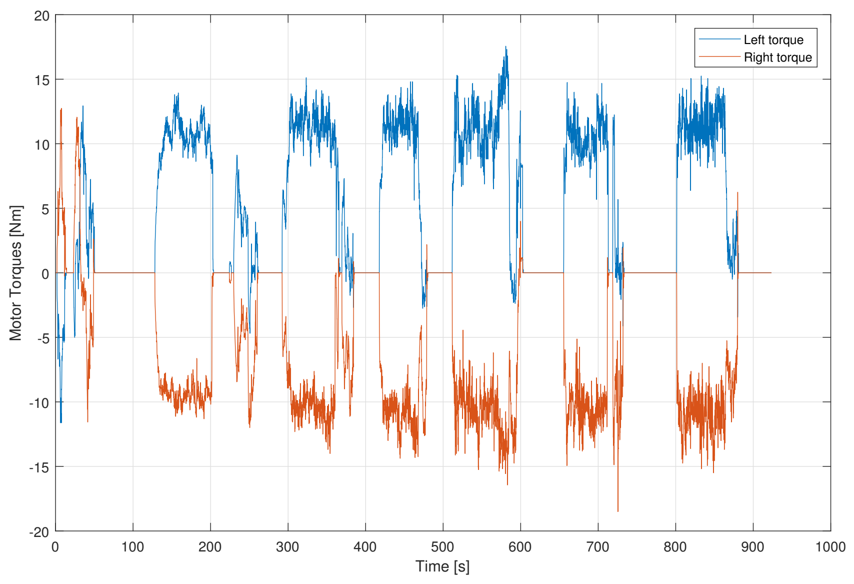


Figure 5.15: Motor torques during during stress test

the graph, the power values are almost the same for the two tracks, as expected when following straight trajectories with the tracks working at the same terrain conditions. The power goes from $500\,W$, in the lower speed case, to $2.5\,KW$, in maximum speed conditions, for *each* motor, getting close to the motors nominal power value of $3\,KW$. Figure 5.17 reports the temperatures evolution both for motors and tracks. The initial temperature offset between the two components is due to previous tests, showing that drivers tends to heat more faster than motors. The graph highlights the same behaviour during the experiment, with the motors temperature quite constant, and the drivers one increasing of about $15\,C$. This effect is due to different heat losses on the two components. Indeed, while the motors are surrounded by metal parts and can easily disperse heat, drivers have been completely covered, on one side, with a plastic cover, in order to prevent water or dust to enter in contact with the electronics. Of course this solution limits the possibility to disperse heat and will be improved at the end of the current experiment session. This is necessary, since the temperature of $75\,C$ reached by the drivers is not so far from the safety limits of these components.

Figure 5.18 shows the DC currents absorbed by left and right drivers and their sum, i.e. the whole current coming from the battery. Of course all the values are

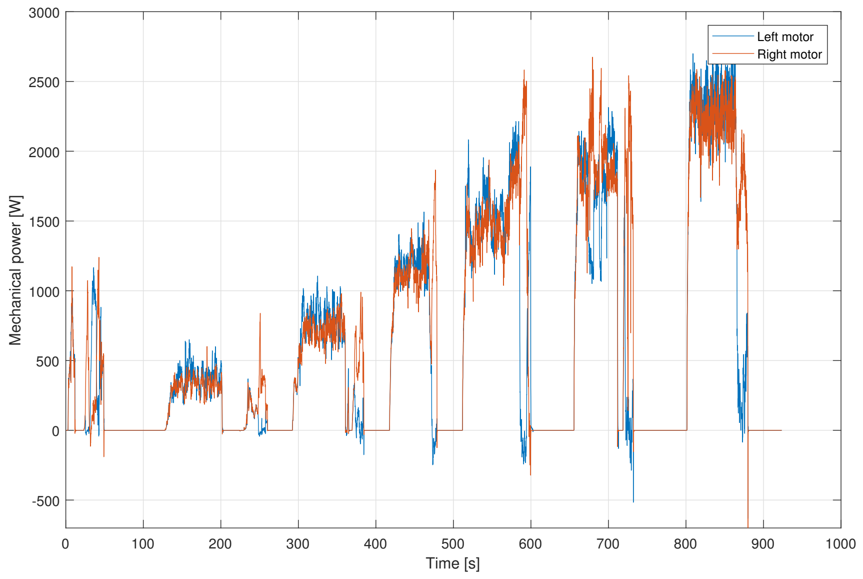


Figure 5.16: Mechanical powers asked by the robot during stress test

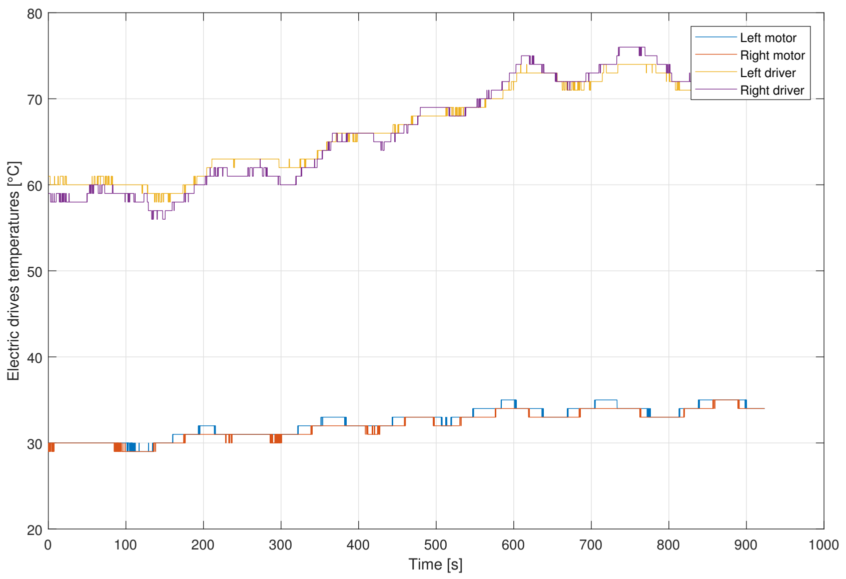


Figure 5.17: Drivers temperatures during stress test

positive, since the currents goes from the battery to the electric system and the motors, except when the speed references are suddenly zeroed and the robot has to break for a while, and the current is generated and injected in the battery. This

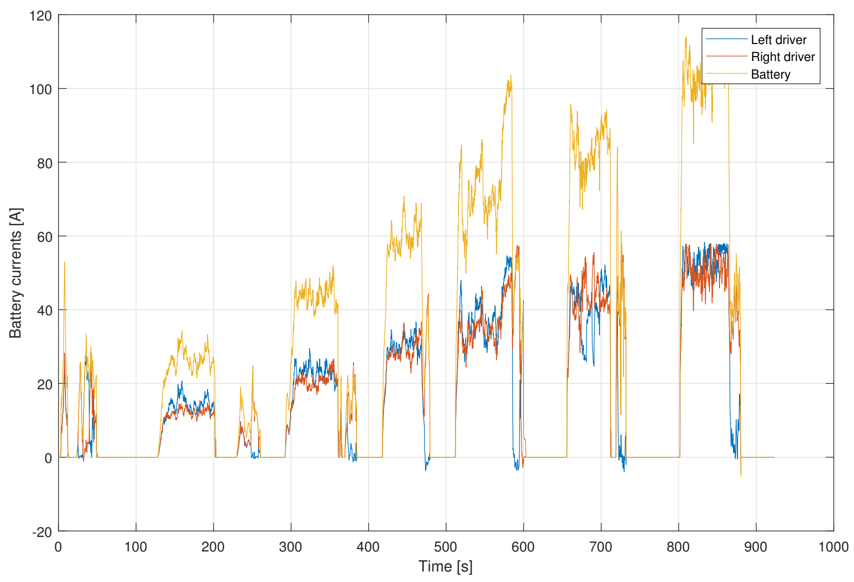


Figure 5.18: Absorbed currents during stress test

effect is more clear at high speeds, when the break action has to be more strong. The current goes from $30\,A$, in the lower speed case, to about $100\,A$, in maximum speed conditions.

Finally, the battery voltage during the experiment is proposed in figure 5.19. Respect to the previous test (see figure 5.7), the voltage decrease is greater and, during the last reference speed step, goes around $2.5\,v$.

As already said, the experimental tests are still ongoing. Unfortunately, it has not been possible carry out a complete *autonomy* experiment, which consist in making the robot move in the orchard, autonomously or not, at nominal speed, starting with a fully charged battery and stopping when it runs out of charge. This test will be carried out during next weeks, giving important results about robot autonomy, battery and driver behaviour when the voltage gets low, and real battery capacity.

In order to give anyway some results about robot autonomy, a projection has been carried out, basically integrating the electric power consumption, detected during an half an hour experiment, in order to get the energy consumed in an hour. Just to give a simple explanation, the experiment at *constant speed* can be considered, in which the power consuming is about $500\,W$ for each motor, i.e. $1\,KW$. So,

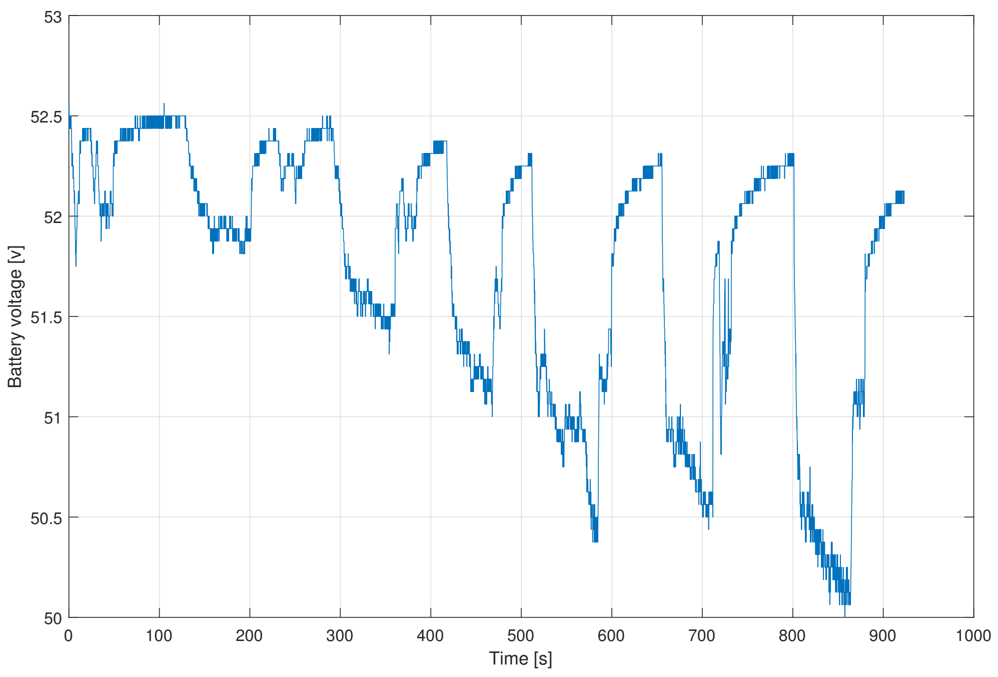


Figure 5.19: Battery voltage during stress test

integrating this result over an hour, namely 3600 s, the result is

$$\int_0^{3600} 1000\,\mathrm{d}t = 3.6\,10^6\,J = 1\,KWh \tag{5.2}$$

Of course this in not a reliable results, but it is useful, in this frame, in order to give an idea about energy consumption and robot autonomy. So, in normal working conditions, the robot is expected to consume 1 KWh. Since the nominal battery capacity is 6.3 KWh, the conclusion is that the autonomy of the robot is expected to be about 6 hours.

This last discussion concludes the first part of experimental results presentation. In the next section, the most important coming outs about autonomous navigation will be proposed and discussed.

5.4.2 Autonomous navigation features

Autonomous navigation within rows

This experiment starts with the robot placed at the beginning of a row in a pear field, with the Robot Control already launched and set in order to carry out an autonomous navigation test within rows. When the system receives the user *start* command, the robot begins to navigate, trying to keep the middle of the row. The test finishes when the robot automativally detects the end of the row and *stops* the mission. Figure 5.20 reports an image from the autonomous naviga-

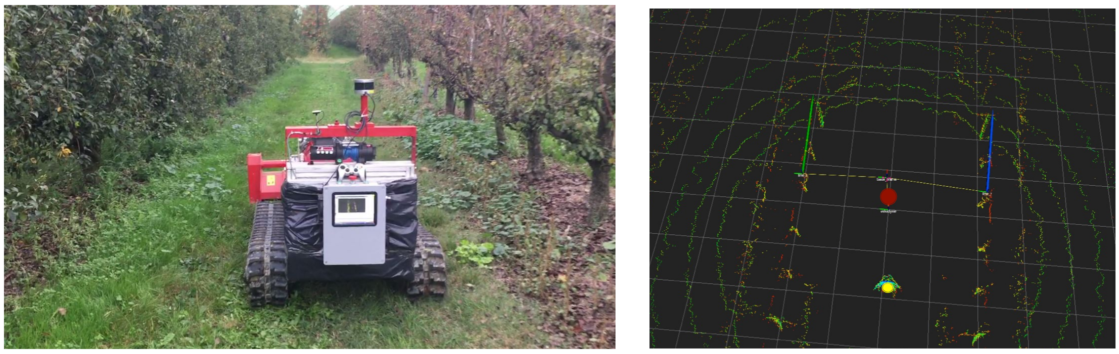


Figure 5.20: Image from the autonomous navigation experiment (left), with the manual controller placed on the robot electronic box, and screen from the onboard computer, showing the lateral estimated lines (right).

tion experiment (left side). Since the robot is automatically driving, the manual controller has been placed on the electronic box. Moreover, a screen of the on-board computer is proposed (right side), in which are visible the lateral estimated lines. In particular, lines are represented by green and blue arrows, while yellow dot represents the human operator and the red one indicated the robot position. The first proposed result consists in the lateral distances between the robot and the row lines estimated by the localization algorithm, which are charted in figure 5.21. The right distance is negative just because in the defined body frame the y_B axes is pointing towards the left (see section 3.2). The two estimated distances are characterized by an oscillation, with a mean value of about $2\,m$ which is constant during all the row lenght, meaning that the lateral distance kept by the navigation system is actually correct. Moreover, the distances sum is about $4\,m$, which results constant, neglecting the small oscillation, and fitting with the real

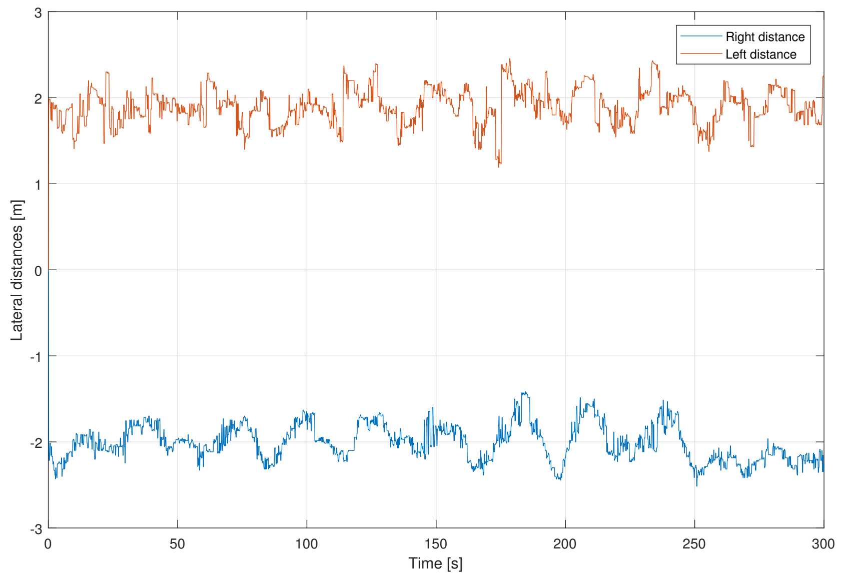


Figure 5.21: Lateral distances between robot and estimated row lines during autonomous navigation

row width value. About the distance noise, this is not related to some algorithm error, but simply reflects *trees* and vegetation shape. This effect is quite natural, because the localization algorithm works on LIDAR data which is not filtered, and of course contains points related to the trees and branches shape. Anyway, the distance feedback is affected by this oscillation, which is about $0.5\,m$ amplitude, with a frequency of about $30\,s$. LIDAR data, or the estimated lines distances could be filtered in some way, or combined for example with a Kalman filter, in order to reduce those ripple effects. But the point is that the computed solution is not wrong, because what is estimated is not the distance from trees logs, but from branches and vegetation. This means that the robot keeps the desired lateral distance from the whole trees, which is actually a positive feature, since it is better to take into account the presence of branches or different trees features and try to avoid them.

Even if it is not visible in the proposed graphics (see figure 5.21), in the right row there are some missing trees. This condition does not affect the proposed navigation approach, i.e. environment perception and localization, thanks to the internal model, which basically creates a *virtual* tree where the real one is missing,

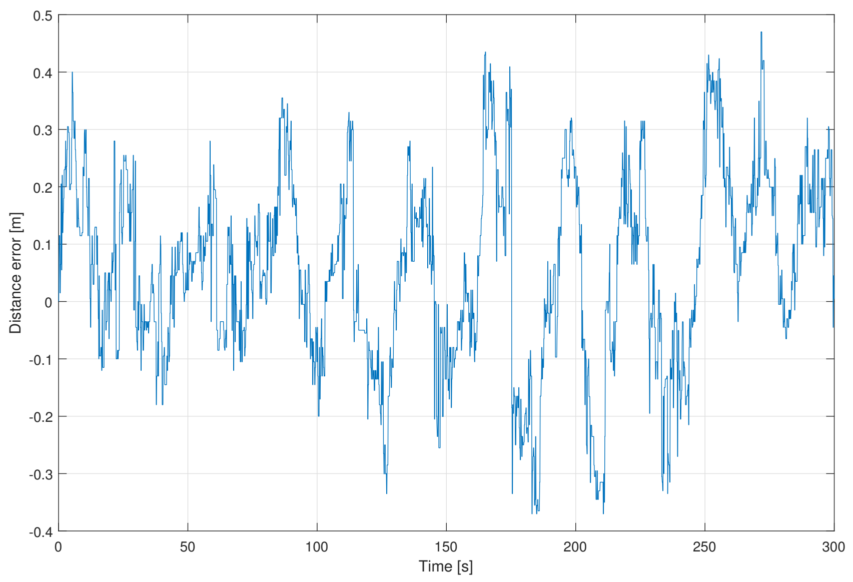


Figure 5.22: Lateral distance error between robot and estimated row lines during autonomous navigation

based on the priori knowledge of the orchard and the previous algorithm iterations results.

Figure 5.22 shows the outer loop distance error computed by the control system (see section 3.8). In this signal the estimated distance oscillation can be recongnized. Of course the regulator is tuned in such a way to neglect this small effect, and apply significant steering actions only in case of considerable distance errors, i.e. if the robot is not aligned with the row or a particularly luxurialt vegetation is detected. Just in this cases the robot steers, following the behaviour of a human driver, which applies small slightly steers in order to avoid a branch or a more protruding tree. The error is calculated respect to the distance between robot and lateral rows, calculated by the *Hough transform* algorithm (see 3.5). Unfortunately, there is no way to measure the precision of this distance, or error, in an analytical way, since LIDAR data represent the only sensor information available during this experimental tests. At the end of last summer the priority has been given to the presented experimental activity, in order to collect data and test the platform before autumn and avoid rain and mud. Due to a lack of time, some features has not been implemented onboard, such as GPS data acquisition, denying

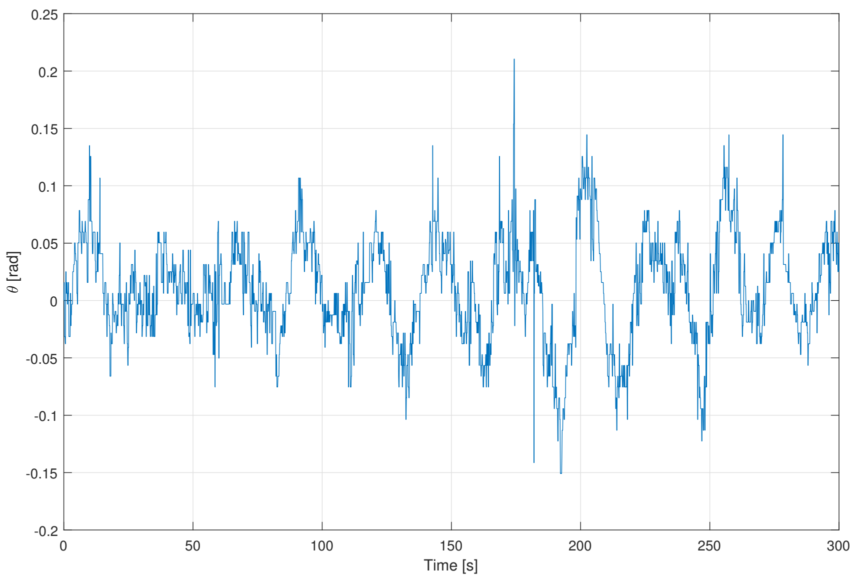


Figure 5.23: Relative orientation between robot and estimated row lines during autonomous navigation

the possibility to use its position information for validating the proposed navigation algorithm. Some possible improvements and a discussion about this lack can be found in chapter 6. About the steering action, the first result is presented in figure 5.23, which reports the relative orientation between the robot and the estimated row lines. Similarly to the estimated distances, this signal is characterized by a zero mean value plus a certain noise. A zero mean value is actually correct, indicating that the robot is navigating parallel to the rows, while the noise presence is due to the fact that the lines estimation tends to follow the trees shapes, leading to a small angle variations. Indeed, the peak values of the angle signal are about $0.1\,rad$, i.e. $5\,°$, which is actually a very limited noise. Finally, figure 5.24 shows the motor reference velocities calculated by the control algorithm during the autonomous navigation experiment. A constant value of translation speed v is imposed, while the automatic steering action keeps the robot at the middle of the rows. So, in order to have a proper translation speed, a reference speed of $1200\,rpm$ is asked, and then the automatic steering action is added, originating the oscillation visible in the graph. Autonomous navigation experiments are currently ongoing. Up to now the robot has automatically travelled about ten rows, which

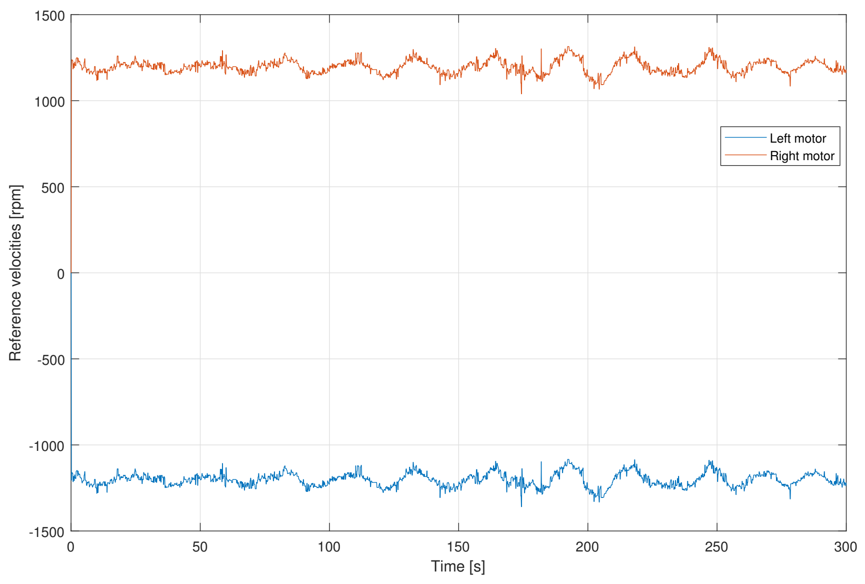


Figure 5.24: Motor reference velocities during autonomous navigation

corresponds to a distance of 2.5 km, both in vineyard and pear fields, without any problem and keeping the middle of the row. A discussion about conslusions or future improvements can be found in chapter 6.

Autonomous navigation outside rows

Regarding autonomous navigation outside row, i.e. automatic row change, experimental tests have not been carried out yet. Actually, the localization algorithm is ready, while the control law is already under development. For this reason, a *complete* row change experiment was not possible to be performed.

The only available results have been obtained during come tests, carried out in a pear and apple fields during the last week, in which the robot has been manually driven from the exit of a row to the entrance of the next one, but with the Robot Control active, in order to see how the localization algorithm was working. Just in order to give an idea about the real test conditions, figure 5.25 shows a picture taken during the experiments, in which the robot has exited the row and is perpendicular to the row lines, in the middle of the change maneuver. In figure 5.26 an *RVIZ* screenshot from the ROS environment is proposed, showing the first

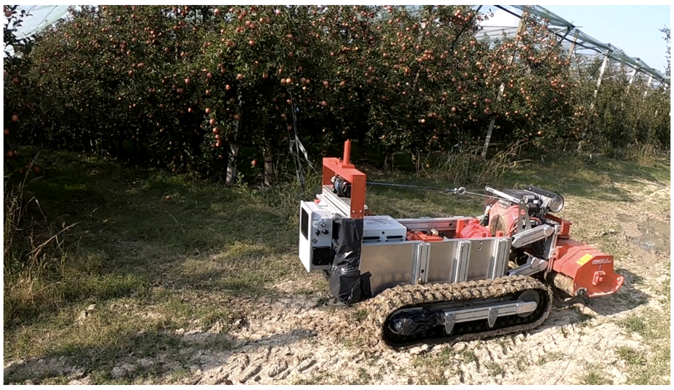


Figure 5.25: Image from a localization experiment, taken during a row change.

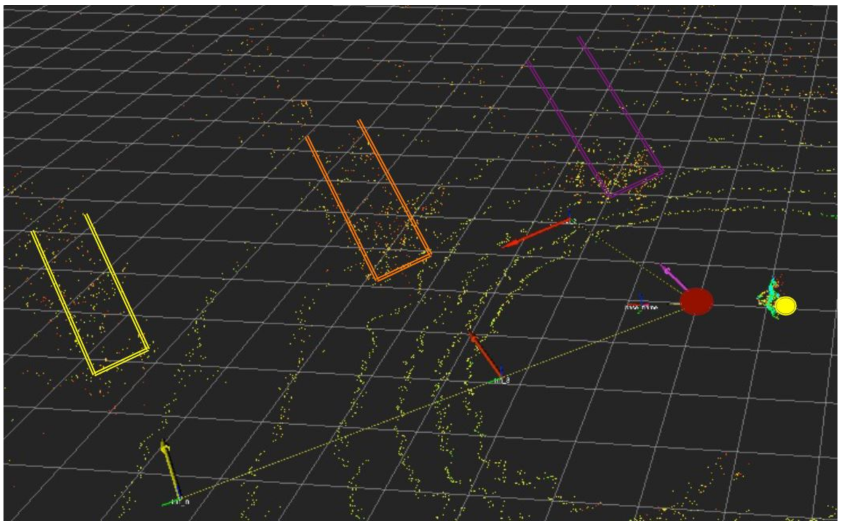


Figure 5.26: Online row lines estimation while the robot is curving in left direction, during a row change.

experimental result about localization during row change. In order to frame rows and robot positions, this items are highlighted in the picture. Red and yellow dots define robot and human operator positions, respectively, while the yellow, orange and violt boxes identifies the orchard rows. The robot is coming from the right side of the picture and has to enter between the orange and violet rows. Since it is positioned in front of the violet row, it perceives just its beginning. Diffrently, it has a better vision of orange and yellow rows. The arrows identify the output of the localization algorithm. The three sides of violet, orange and yellow rows are perfectly detected, while also the rows extremities line is correctly estimated (red arrow), providing the distance from the robot to row entrance. Of course, from the outside of the orchard, the robot perceives just the nearest side of the tree rows (see figure 5.26, where the generated arrows identify just a side of the corrresponding boxes). When the robot is located in front of a row, position in which it can not estimate this feature as a line, the priori knowledge about the environment, i.e. the internal model, is employed, and a *virtual* line continues to be estimated, even if it is momentarily not visible.

Another example is proposed in figure 5.27. In this case, the robot is coming

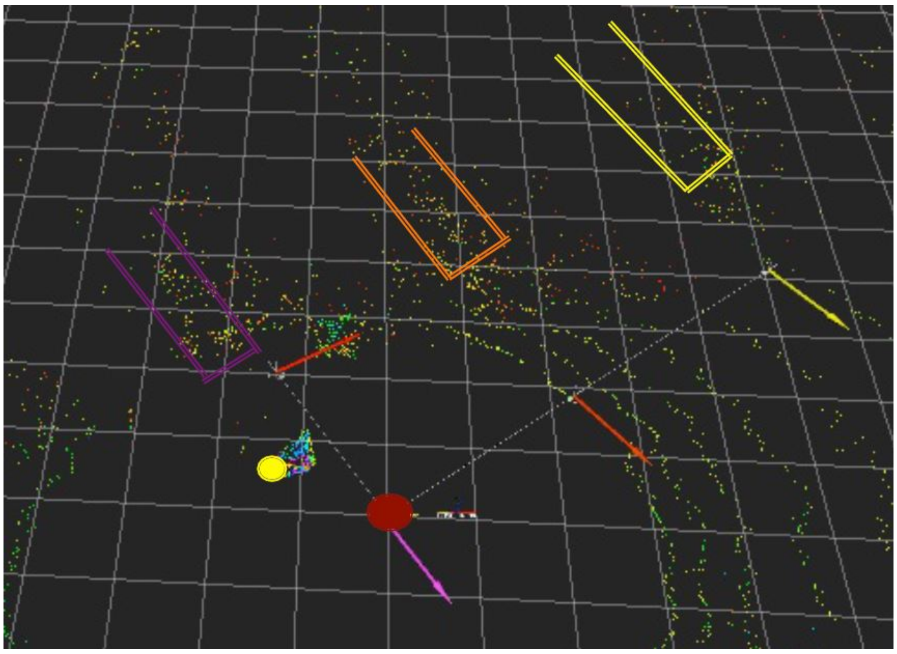


Figure 5.27: Online row lines estimation while the robot is curving in right direction, during a row change.

from the left side of the picture, and has to enter between violet and orange rows. Again, the three parallel lines and the perpendicular one are correctly detected.
Of course, in these examples the developed internal model is playing a foundamental rule. Without it, the standard algorithm could not estimate these four lines, represented in the pointcloud by just a few points. The most significative example is the estimation of violet lines, whose beginning is the only visible part, making it hard to detect their orientation, and yellow lines, defined by just a few points. Moreover, it is interesting to observe that in both the two proposed results, colors assigned to the estimated lines are not casual. In both cases, the violet arrow identifies the closer row, the orange one is in the middle and the yellow one is the

farthest. Moreover, the red arrow defines the rows extremities line position. This mean that the algorithm knows exactly the lines *correspondence* and, starting from this result, robot position is quite easy to calculate.

5.5 Conclusions

In this chapter the carried out experimental activities have been presented, in terms of objectives, tests description, robot setup and, of course, experimental results. The tests session has been organized in order to get data and validate all the robot features or components. The outcomings, both for the electromechanical subsystem and the autonomous navigation algorithm, are quite encouraging. Indeed, the results shows that the robot is able to act in agricultural anvironment and manage an onboard tool, moving at interesting velocities and without exceeding motors or drivers nominal powers. About autonomous navigation, tests demonstrate the capacity of the robot to drive itself within rows and also to localize outside rows.

Chapter 6

Conclusions and Future Developments

This chapter reports the conclusions of the research activity described in this thesis. The results about developed navigation algorithm and experimental activities are breafly described. Finally, possible improvements and future developments are discussed.

6.1 Conclusions and Final Discussions

The research work presented in the thesis frames in the context of precision agriculture, with the goal of developing an agricultural robot able to autonomously perform different tasks and support human operators working in the field. In particular, the problem of letting a tracked robot navigating autonomously in agricultural environment has been addressed. In particular, the work focused on navigation in structured environments, such as orchars organized in rows. For this purpose, a proper sensor suite has been chosen for the robot, in order to provide it the necessary environment perception and neglecting, or exceeding, the concept of autonomous navigation based on absolute position provided by GPS receiver. Indeed, for navigating within orchard rows, what is foundamental is localization with respect to the surrounding environment, in order to act at a certain distance from trees or avoid obstacles or deal with unaspected environment features. For this reason, one of the foundamental robot components is a LIDAR sensor, which provides environment perception and allows to recognize the orchard structure and localize within it, by means of models extraction and post-processing algorithms, as far as detecting ostacles. The most important novelty of the proposed navigation algorithm is the application of an internal model of the orchard structure, i.e. the idea of exploiting the a priori knoledge about the environment. The localization algorithm is based on rows estimation, in the sense that rows are modeled as lines, which are extracted from the data coming from the laser scanner. The internal model application makes the estimation process more fast and robust, since the algorithm does not look for "casual" features in the whole space anymore, but for parallel lines located at a certain mutual distance, and close to the previous iteration solution. Moreover, environment knowledge improves also the control algorithm. Indeed, the robot can navigate using the internal model, i.e. by means of feedforward actions or feedback coming from encoders or IMU, and periodically "taking a look" with the LIDAR for correcting possible position estimation drifts. In this way the computation is more fast, since the laser scanner post-processing algorithms, very heavy from this point of view, do not have to run at high frequency, like the rest of the navigation algorithms. The proposed localization algorithm, developed for acting within rows, has been generalized for position estimation also outside the rows, in order to manage also the row change and give the possibility to autonomously navigating in an entire orchard. With the purpose of testing the proposed navigation algorithm an home-made agricultural robot has been developed in the lab. This vehicle is moved by tracks, which gives it an high traction capacity, together with the ability of moving on uneven terrain, with mud, snow and holes, which characterizes the agricultural environment. The traction system is full electric, with energy provided by a Lithium battery placed in the back side of the frame. The robot is able to carry on and manage different agricultural tools, such as mower or sprayer, like a real tractor. Mechanical power to the tool is provided by an endothermic motor, while its position can be adjusted by means of a winch and proper mechanical structure. Then, a proper software structure has been developed for the robot, composed of an HMI, for human-robot interaction, and an onboard control system, for autonomous navigation. The HMI provides communication within the human operator and the vehicle, giving the possibility to assign missions and monitor their status on a remote computer. Differently, the onboard navigation system contains all the features and algorithms which let the robot managing missions and autonomously acting in the field. Finally, the whole system has been tested during an experiment session, which is, actually, still ongoing. During performed experiments the whole system has been tested, focusing on electromechanical subsystem and autonomous navigation features, but implicitly validating mechanical frame, electronics, sensors, communication system and software structure. Experimental results are quite encouraging and are showing the hoped-for performances and potenzialities of the developed agricultural robot. Indeed, the vehicle easily moves in the field and within orchard rows, also when equipped with the mower. Mechanical power, torque and speed of the motors did not exceed nominal or safety values during experimental tests, showing a correct components dimensioning. Moreover, data projections about system autonomy shows that the robot can work for about five hours, when equipped with just one battery. This means that, using two batteries, the platform can be autonomous for a whole working day. Moreover, even the autonomous navigation gave important results. Now the robot can move autonomously within rows without problems, at the desired translation speed. The autonomous navigation experiments let suppose that the proposed localization and control algorithm are performant and robust enough to drive the robot along an indefinite distance.

6.2 Next Developments and Future of the Project

Even though the interesting results obtained from the described research activity, of course there are still many possible improvements and developments. In order to define a methodical discussion, such improvements will be presented considering first actions immediately applicable and regarding the robotic platform, and then some points related to the future concept of the project and to the possible HMI and software developments.
Looking at the future, an important improvement is for sure the transition to a full-electric prototype, which consists in removing the endothermic motor and provide mechanical power to the onboard tools by means of an electric machine. Also the tools support could be re-designed, in order to remove winch, lead battery and gas can. This would clear a lot of space in the central part of the frame, giving the

possibility to carry onboard two or three Lithium batteries, instead of one, as in the current configuration. In this way, the robot is expected to store enought energy to power both onboard tool and locomotion system. If this evaluation turned out to be wrong, an hybrid solution could be adopted, in which an endothermic motor is mounted onboard in order to recharge batteries and increase robot autonomy.
Also a cover for enclosing the central part of the frame could be really useful, because it would prevent damages to the batteries and protect them from rain and dust, as far as the wirings or other fragile components.
In the current configuration, the robot has not the possibility to manage a sprayer, since there is no tank installed onboard for storing fertilizers and treatments. Since the battery shape can not be modified, the idea could be to exploit the free space over the two tracks and mount onboard two tanks, with a shape suitable for installation in that space. Differently, if the batteries had to be substituted, maybe an unic big tank could be installed in the space at the frame center, with batteries, of a proper shape, placed over the tracks.
About sensors, a proper suite is already present onboard, for localization and navigation. What is still missing is maybe some proximity sensor mounted at the edges of the robot, with the aim of signaling the detection of too close objects and provide some redundancy for collision avoidance. Also a feedback about the height of the onboard tool would be useful, in order to automate the tool vertical position management. In this way the robot could lift a bit the mower when approaching a row change or a generic curvilinear path, saving a lot of energy, in view of the presented experimental results, or regulate the sprayer vertical position in order to adapt to different trees height.
Thanks to experimental activities, the autonomous navigation within rows has been tested and now is a functioning component of the agricultural vehicle. Actually, in view of a complete validation of the algorithm, there is still a missing part. As already mentioned, during experimetal tests, in which the rover had to travel a row keeping a certain lateral distance, the robot trajectory has not been compared with GPS position, or another reference. This procedure would make it possible to verify if the platform is really navigating in the middle of the row or not, and define an error with respect to a certain frame. This point has to be taken into account for future activity, not only during experiments within rows,

but also row change or other phases of autonomous navigation, in order to correctly validating the algorithms. From this point of view, also sensor fusion could help for improving robot behaviour. Working with a position feedback not based anymore just on LIDAR data, but taking into account also encoders, IMU and GPS, would immediately lead to a more precise, or correct, localization. In this way the validation, or comparison problem described above tends to disappear, since the robot would already act on the basis of all sensors data.
Differently, there are many other features that needs to be added or improved, in order to obtain a robot with completely autonomous navigation capacities. The most important features are for sure row change and navigation in open field condition. The localization algorithm for row change, i.e. outside rows, has already been developed and some tests has been carried out, giving quite good results. But the algorithm still needs to be tested even more and better, in order to remove bugs in the code and find possible errors or unexpected behaviours. Moreover, the control law con guiding the robot during row change is currently under development. So, the work for developing localization and control, i.e. navigation, during row change is not finished yet. Autonomous navigation capacities in open field environment would be useful both for performing missions on fields with no trees, like wheat or corn fields, where there is no structure to exploit for localization, and also for autonomously moving from the base to the orchard in which the mission has to be carried out. In this case, localization and guidance would be based on GPS technology, with the LIDAR used just for obstacle avoidance features. Moreover, all this kinds of autonomous navigation features does not take into account the possible presence of obstacles or unexpected objects on the robot path. For this reason, a 3D obstacle detection algorithm based on laser scanner data is currently under development. Some simulations have already been carried out, where a sort of RANSAC angorithm provide recognition of ground plane and object which arise from the latter, with the possibility to distinguish between fixed and moving ostacles. This last features is really important, in order to develop a robotic platform which is, other than capable of autonomous navigation, safety, especially in cases in which the obstacle is moving and could be potentially a human or even a child.
Now the discussion has come to the possible developments about software structure

and, in particular, HMI. All the features named before are of course foundamental, but regards just the robotic platform. A robot with advanced navigation capacities is still just a robot. But if the platform is coupled with an advanced HMI, it assumes the capacity to interact with the human operator, accept missions, evaluate its battery status, advice the farmer about possible actions and treatments, and assumes the capacity to really cooperate with the farmer. For this reason, in order to discuss the possible HMI improvements, it is better to present the future concept of this research project. Then, from this vision, some feasible and more realistic improvements can be extracted and discussed.
The future concept of the project consists of a remote interface, a communication system, a server or data storage center and a certain number of robots, which, in this vision, are just the actuators of a more complex and smart robotic system. Remote interface, server and robots are always connected. In this way, from the HMI the farmer can monitor the operating state of his robots, knowing battery status, problems during the mission, and so on. All acquired data is stored in the server, from framers decisions and plants deseases to information sampled by the robots during work, like terrain humidity, temperature, and so on. This data is elaborated by the computer and is available to all system components. For example, the smart system advices the farmer with weather information, treatments that should be distributed on the basis of season, rain conditions and past history. When the farmer defines a mission, the system computes robot trajectories and evaluates if the battery charge is enought or some breaks are necessary, and, in this case, optimizes the path respect to travelled distance or time. When the battery is running out of charge, the robot automatically goes to the nearest recharge station, with no need of human intervention. From this perpective, the singol automatic tractor presented in this thesis evolves in an heterogeneous robotic team, composed of both rover and drones. Expecially in case of extended farms, multiple rovers could work in parallel on different orchards or cooperate together on the same activity, while drones, which are lighter and faster, could be exploited for inspections of orchards and ground robots and for data collection.
Of course this kind of automation is very far from the current progresses, and also hard to be addressed, at least from a technological point of view. Returning to the actually fisible improvements of the presented HMI, the first development could

the management of complete missions, in which the robot starts from its base, autonomously reaches the selected orchard, carry out its mission and returns home. This kind of results implies the presence of a really advanced user interface and an automatic tractor with high navigation and work capacities in different kinds of environments and cutivations. This would be of course a great result, which our research team hope to reach in the next months.

Appendix A

Sensors

In this section a brief description of the sensors employed in the research project or discussed in the thesis is given. The aim is not to report a detailed or exhaustive description, but only an explanation, for each sensor, of the measures it returns, functioning principle and main merits and defects, in order to make discussions in the text more fluent and comparisons or choises easy to understand.

A.1 Encoders

Encoders are sensors or transducers of *angular position*. They are usually mounted on the motor shaft and, by means of an appropriate acquisition circuit, allow position and angular velocity estimation. Physically, their output is a an electrical, analogue or digital, signal, while the operating principle is based on light modulation (electric photo sensor). Indeed, a beam of light generated by an emitter is modulated, in such a way to discriminate the position of the shaft, and read by the receiver. Finally, data is extracted through the signal conditioning provided by the acquisition circuit.
Encoders can be divided into two types:

- **Absolute**
  The output of the sensors is a series of bits (see figure A.1), which discriminate the absolute position of the motor shaft. Modulation is implemeted by a disk with transparent and opaque areas, arranged on concentric circular crowns. The angular resolution of the sensor is calculated as $ris_{Abs} =$

$360/2^N$, where N is the number of crowns, i.e. the lenght of the output bit word. For robotic applications the required resolution is at least 12 tracks, with a cost that increases proportionally.

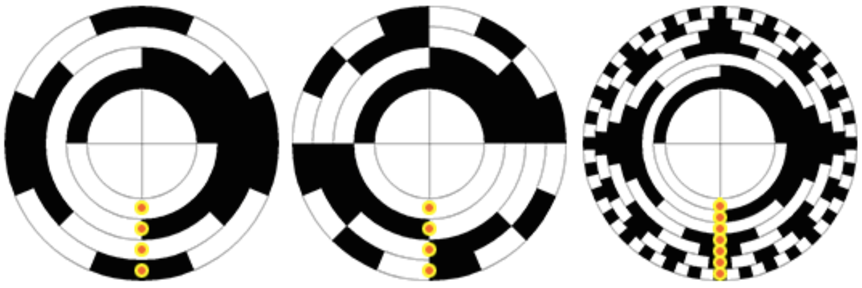


Figure A.1: Absolute encoders schemes, with binary 4 bits (left), Grey 4 bits (center) and 7 bits (right) codings

- **Incremental**
  Consists in a disk with two tracks in which transparent and opaque areas (see figure A.2) are arranged alternately. The presence of two traces allows to discriminate the rotation direction, while the reference mark on the innermost crown if for position zeroing. Defined N as the number of steps, i.e. light / dark areas per revolution, and since signals A and B (vedi figura) are offset by a quarter of a step, the resolution is calculated as $ris_{Inc} = 360/4N$. In this case the cost of the sensor is lower and is not very conditioned by the resolution increase.

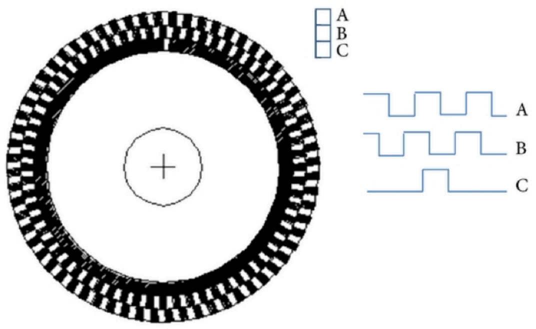


Figure A.2: Incremental encoder scheme, representing channels A and B and the zeroing reference mark

- **Sin/Cos**

Usually employed in modern vector control electric drives, have an operating principle similar to incremental encoders. The difference is that masks modulate the light signal in an *analog* way and so A and B are no longer digital pulses, but sinusoidal signals, mutually out of phase of 90. This analogue information allow to discriminate the rotation direction, obtain a sort of forward / reverse counter, by converting sin / cos to quadrature signals, and to calculate the position of the shaft as $\theta = \arctan(\sin(\theta)/\cos(\theta))$. Exploiting this technique, a large increase in performance is obtained, especially about resolution and velocity estimation at low speeds.

A.2 GPS

GPS (acronym for Global Positioning System) as an absolute, satellite-based, positioning system, which provides a receiver (see figure A.3) informations on its geographical coordinates.It is based on a satellites constellation which orbits the Earth and sends the actual position of each component. The time taken by the

Figure A.3: 3DR (left) and uBlox (right) GPS receivers

satellite to "respond" gives a measure of the distance between the satellite itself and the receiver. By triangulating these informations, is possible to identify the position of the receiver on the Earth. For this reason, the receiver needs at least to communicate with four satellites, in order to calculate its position. GPS system allows localization in any point of the earth's surface that allows an obstacle-free contact to the satellites, regardless of weather conditions and at a very low cost for the GPS receiver (order of Euros or tens of Euros).
On the other hand, position accuracy is limited (about one meter) and lowers with the number of visible satellites. Moreover, is deteriorated by the presence of clouds, vegetation or other objects that limit the receiver visibility. The lack

of precision therefore limits the scope of this technology, which turns out to be unsuitable for navigation in narrow environments or precision readings.

Position errors due to the environment are mainly related to two factors:

- Reception of a satellite signal after it is bounced on one or more surfaces, causing a delay and therefore an error in signal fix or position estimation.
- Alternation of satellites that the receiver sees and manages to lock onto. For example, one of the fixed satellites could be lost, forcing the receiver to repeat the fix procedure or leading to measurement errors.

For these reasons is important that the receiver,during the measurements, locks onto several satellites, in order to have sufficient redundancy and to limit the described effects. Some improvements have been developed over time, in order to achieve better reliability and accuracy of the GPS system. First, after the United States, actually the creators of GPS system in the '80s, different countries have also developed their networks and nowadays there are, among the others, GLONASS (Russian), GALILEO (Europe) and Beidou (China) systems. Thanks to the higher number of satellites and the augmented coverage, the reliability of the, so called, GNSS positioning system increases and the accuracy reaches about 0.7 meters. Then, a further progress has been the RTK (Real Time Kinematic) technology, which enriches the position accuracy by means of external corrections. In this case the receiver is not stand-alone anymore, but communicates with the outside. This tecnique can be implemented basically in two ways:

- Integrating a communication system into the receiver, such as a SIM card, allowing connection to the national network (Netgear, in case of Italy), and corrections receipt.
- Combining the mobile receiver with a *fixed* one. This latter has to be precisely positioned at a well known GPS coordinates place. The idea is to compare *true* and received position in the fixed station, compute errors and corrections and send this ones to the mobile receiver.

Integrated receivers (see figure A.4) are easier to use, since the system is composed of a single element, while the negative aspect is given by the cost, which runs

Figure A.4: Trimble R8s GNSS-RTK receiver

to thousands euro. Differently, systems based on fixed receivers are essentially less expensive, but involves a certain investment for the installation placement of fixed receiver and local network (communication channel between the two devides). In conclusion, exploiting satellite technology very high localization precision can be reached. For example, the positioning error of modern GNSS-RTK receivers is about centimeters. The main drawbacks of this technology are high cost, around thousands euro, and necessity of satellite coverage, which limits the application fields.

A.3 IMU

IMU, acronym for Inertial Measurement Unit, is a device that provides linear acceleration, angular velocity and Earth's magnetic field, by means of accelerometer, gyroscope and magnetometer (see figure A.5). The degrees of freedom indicates the output dimension and depends on the number of sensors integrated in the specific IMU. Applying filtering and integration techniques, rigid body kinematics can be estimated, i.e. knowing its position and speed. An important advantage of IMU, especially for applications on land or agricultural vehicles, is that measures are not disturbed by the slippage of the vehicle on the ground, as happens to encoders, and therefore remain faithful to real displacements. Moreover, measurements are provided at high frequency, at least tens of Hertz, making the IMU suitable for high dynamic applications, such as control of cars or drones. The main disadvantages are due to bias or calibration error. These little deviations are not periodically compensated, but tend to accumulate during the estimation and

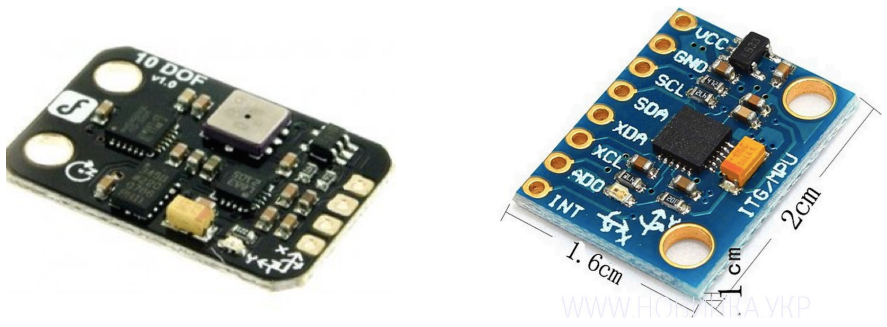


Figure A.5: Ten (left) and six (right) degrees of freedom IMUs

integration processes, leading to significant drifts over time. This sensor therefore needs periodic calibration and corrections, per example by means of a GPS or a proximity sensor, in order to correct the calculated measurement and cancel drifts.

A.4 Ultrasonic sensors

The ultrasonic sensor measures objects *distances* within a certain perception field and is based on the sonar principle. Indeed, the sensor is composed of an emitter and a receiver of high frequency sound waves. Knowing the speed of sound in the air and calculating the time elapsed from sending to receiving, the distance of the object on which the wave bouced be can calculate. In figure A.6 an example of such a sensor is showed, where are visible emitter, receiver, and, in the central-upper part of the image, a 4 MHz oscillator, needed for a correct time count. These sensors are quite inexpensive but, unfortunately, quite under-

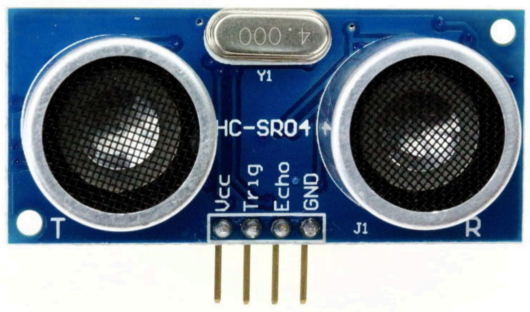


Figure A.6: Example of low-cost ultrasonic sensor

performing, due to the short range and the low accuracy. In addition, the wave emitted by the sensor does not remain point type, but expands in a cone shape, thus limiting the accuracy of the sensor and the possibility to use it for mapping or object shape reconstruction. Finally, due to limited speed of sound in the air, it is not possible to get high frequency sampling. For these reasons, ultrasonic sensors are employed either for simple applications or to provide redundancy.

A.5 Laser Scanner and LIDAR

Laser scanners measure at high frequency the position of hundreds of thousands of points which define the surface of the surrounding environment. The result of the acquisition is a collection of dense points commonly called ”pointcloud” (see figure A.7). Laser sensors are often referreed to as LIDAR (acronym for Light De-

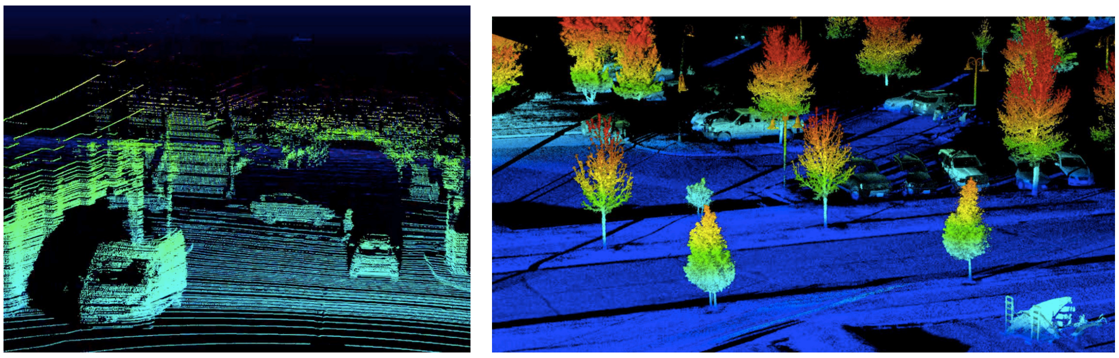


Figure A.7: Example of ground (left) and aerial (right) laser scanner data

tection and Ranging), which is literray the sensing or perception technique based on laser scanner technology. The working principle is the same as sonars or ultrasonic sensors, where the object distance is determined by measuring the time elapsed between the emission of the pulse and the reception of the bounced signal, but using light instead of sound waves.
LIDARs can be both 2D and 3D. In the first case the device is implemented as a single laser beam rotated around its axis, mechanically or by means of mirrors, thus scanning a *plane* in the space. The output is a vector expressed in polar coordinates, representing the distances on the plane cut by the laser beams and the related angles respect to the sensor heading. A 3D pointcloud can be achieved using a gimbal, i.e. an external device that oscillates the sensor (see figureA.8).

By making the scanning plane oscillate, a three-dimensional scan is thus obtained.

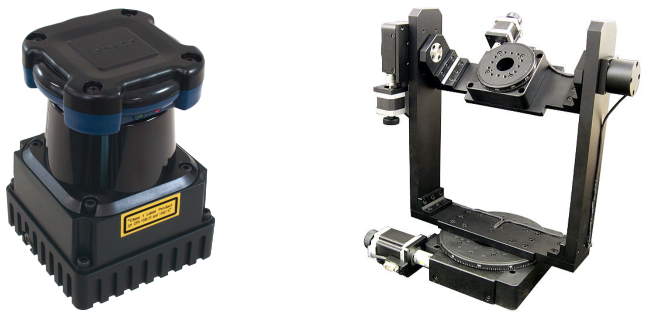


Figure A.8: Hokuyo UTM-30LX-EW planar laser scanner (left) and example of a gimbal model (right)

Differently, in 3D laser scanners (see figure A.9) this principle is already integrated in the sensor, which directly generates a 3D pointcloud.
This sensor is extremely performing and allows to detect at any time the distance

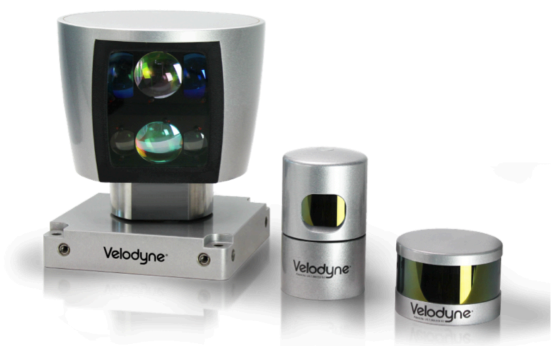


Figure A.9: Examples of 3D Velodyne LIDARs: HDL-64E (left), HDL-32E (center) and VLP-16 (right)

of hundreds of thousands of points, with a range of tens of meters and millimeter resolution, regardless of weather conditions, light or dark. Its applications are innumerable and undergoing very strong expansion: forest and archaeological surveys, environments mapping, speed detection, mapping and obstacle avoidance for automatic vehicles, agricoluture, up to lunar or Mars surface detection, applying the sensor on a orbiting satellite. The main disadvantage of the lidar is the high cost, which runs into thousands or tens of thousands euro.

A.6 Cameras

Cameras are devices capable of capturing, thanks to the light reflected by surroundings, environment images or video sequences, imitating the visual ability typical of the animal kingdom. Once technological development has allowed the development of sufficiently performing computers, it is possible to combine cameras performances and images processing, so extract features, rather than recognize objects or estimate their distances. This marked a turning point in the field of application of these technologies, which are now employed to monitor traffic, in assembly lines for defect detection, surveillance, and in robotics in general, trying to provide robots with a visual ability that is closer and closer to the human one. Cameras, either mono or stereo (see figure A.10), depending on whether they have one or two sensors, are quite inexpensive, small and light devices. This last feature

Figure A.10: An XBOX 360 Kinect, example of common stereo-camera

makes them suitable for many applications where other sensors, such as LIDAR, would be too heavy or bulky, or perhaps expensive. An example is the application on drones, in order to carry out surveillance, but also visual odometry rather than obstacle avoidance, or on the end effector of robotic arms. In addition, cameras are exploited for their unique visual capability. Recognition, according to the shape and color, of objects to choose from or to discard on a production chain or perhaps, recognition of animals rather than ripe fruits agriculture, are just a few of the possible applications.
The disadvantage of the cameras is that the performances are strongly related to weather conditions. In low light, dense fog or dust deposited on the lens, camera loses all its usefulness. This tends to limit their applications, expecially in dusty or outdoor environments.

A.7 Infrared cameras

Infrared cameras are devices capable of detecting from far away the environment infrared energy, i.e. the emitted heat. This kind of camera is not sensitive to meteorological conditions, such as blind or fog. For this reason it is mainly employed for surveillance of fire hazard environments, in any application in which temperature control can prevent faults rather than danger situations, but also for human body detection or traffic monitoring.
This devices could find useful application in precision agriculture, for plant monitoring. Indeed, they can detect in a while plant or ground temperature, and, combined with other sensors data, would give informations about terrain humidity and trees health.

Appendix B

Estimation algorithms

The achievement of this research project objectives, in consequence of the implementative choises regarding sensors and localization approach (see chapter 3), are related to the LIDAR pointcloud post-processing. In particular, the proposed navigation approach is based on the extraction of models from the laser-scanner data, for estimating rows position. In this appendix is reported a description of the working principle of the estimation algorithms which are most used in this application field and have been considered as a starting point of the proposed navigation algorithm. For technical details or a deeper explanation the reader is addressed to the references.

B.1 Introduction

The goals of this research project concern autonomous navigation of a robot in a structured agricultural environment by means of a laser scanner. In literature, different papers or research projects propose navigation techniques based on orchard rows modelling as lines, which are extracted from camera images or LIDAR pointclouds. In this project, the localization approach is based on lines extraction from laser scanner data, since LIDAR has been chosen as navigation sensor (see chapter 3) and lines can effectively represent orchard rows.
Figure B.1 reports LIDAR data collected in different kinds of orchards and shows very disturbed pointclouds, due to different vegetation features which alters trees shape and so the correct row estimation, presence of grass at ground level and

perception of trees belonging to other rows. In this frame is better to recall the

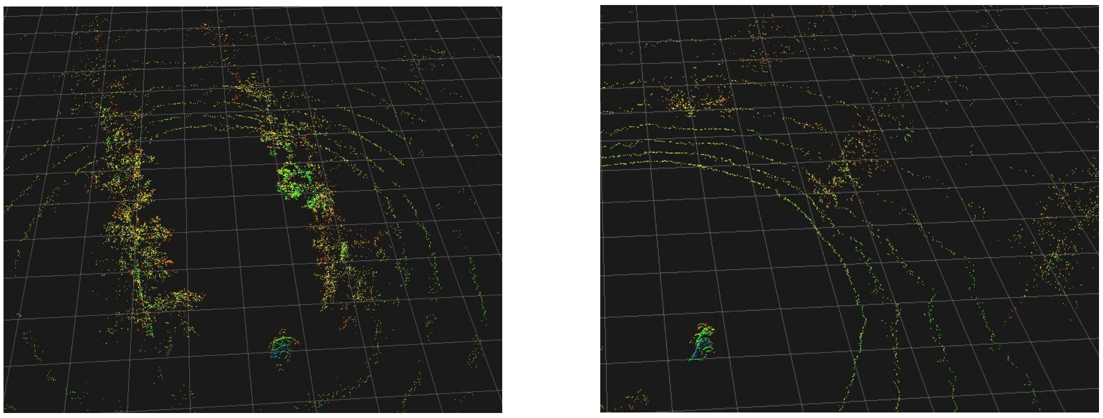


Figure B.1: Realistic LIDAR pointclouds, collected within (left) and outside (right) orchard rows

definitions of *inliers*, points which fits the model, and *outliers*, which are far from the model and tend to disturb its correct estimation. Figure B.2 shows a LIDAR pointcloud collected between two rows, in which inliers (blue dots) and outliers (red dots) are highlighted. The inliers tends to define the two row lines, while the

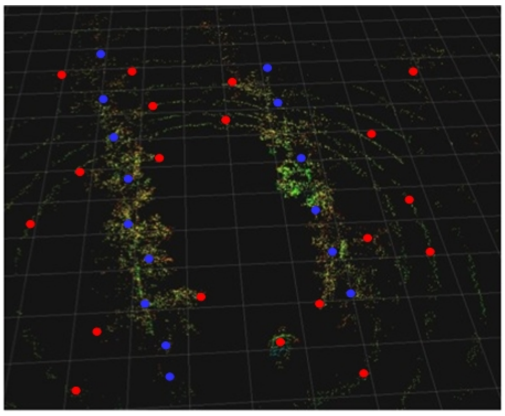


Figure B.2: Discrimination of inliers (blue dots) and outliers (red dots) on real pointcloud data

outliers represent different trees, ground or simple noise, and disturb the rows perception. For this reason simple filtering techniques, such as the *Moving Average Filter*, are not so useful, since their result is strongly disturbed by the outliers. Another approach investigated for this purpose consists in the *Least Squares* estimation. This algorithm estimates lines parameters which minimize the sum of squared errors, or *residuals*, calculated as the distance from the pointcloud entries and the estimated line. Unfortunately, this method is strongly affected by the outliers, which are not recognised and weight in on the line estimation. Figure

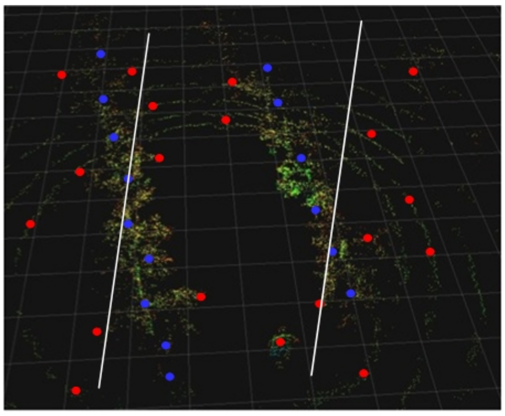


Figure B.3: Least Squares line extraction, affected by the presence of outliers

B.3 reports an example of models (white lines) estimated by the Least Squares algorithm, showing how it is sensitive to outliers, which perturb the lines orientation. For this reasons, the Least Squares algorithm can not be employed in this kind of application. The choise fall on algorithms whose estimation is not affected by outliers, i.e. which are able to isolate them from the inliers. In the next two sections the most used estimation algorithm, in agricultural field and outdoor environment, are presented: RANSAC (section B.2) and Hough Transform (section B.3).

B.2 RANSAC

RANSAC (acronym of RANdom SAmple Consensus) [6] is an iterative method for estimating parameters of a mathematical model from a set of observed data containing outliers. It is a *non-deterministic* algorithm, in the sense that it produces a reasonable result only with a certain probability, increasing with the number of iterations. The algorithm was first published by Fischler and Bolles at SRI International in 198 (citazione!!!). The most basic assumtions of the algorithm are:

- Data consists both of *inliers* and *outliers*.

- Given a set of inliers, there exists a procedure which can estimate the parameters of a model that optimally fits this data.

RANSAC attempts to exclude outliers and find a linear model based only on inliers. This is done by fitting linear models to several data *random samplings* and returning the model which best fits a subset of the input data. Since inliers tend to be more linearly related than a subset of both inliers and outliers, a random subset consisting entirely of inliers will have the best model fit. In practice, there is no guarantee that a subset of inliers will be randomly sampled, and the succeeding probability depends on the proportion of inliers in the input data, as well as the choice of the best parameters.
The algorithm parameters are:
- N, number of iterations
- s, minimum number of points defining the model
- d, distance threshold for defining if a sample belongs to the sampled model
- T, minimum number of close points required by the sampled model

A simple explanation of the RANSAC algorithm is proposed, with the aim of clarifying its working principle in a linear model estimation ($s = 2$). Defined the counters i, for the number of samples, and j, for the iterations, the algorithm evolves through the following steps:

1. Two random points are chosen from the poincloud, defining a line and so a possible model.
2. The number of pointloud items fitting the sampled model, i.e. the points located within a certain distance threshold (d) from the line ($d_i \leq d$, with d_i distance from the line and the $i-th$ sample), are counted.
3. The algorithm checks is the count is over a certain threshold ($c_j \geq T$). In this case, the sampled model could be correct one and its parameters and count are saved.
4. The procedure is repeated until the number of iteration reaches its maximum ($j \leq N$).
5. The counts related to the diferrent models are compared. The maximum count defines the chosen model, i.e. the output of the algorithm ($max(c_j), j = 0, ..., N$).

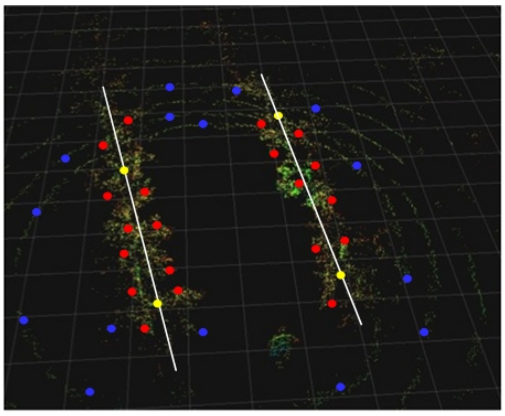


Figure B.4: RANSAC algorithm iteration, where the points close to the sample are counted

A breaf study about the RANSAC success probability is now proposed, in order to give an idea of the parameters and conditions by which it is affected. Given

- e = probability that a point is an outlier
- s = number of points in a sample
- N = number of samples (we want to compute this)
- p = desired probability that we get a good sample

The probability that at least one sample was not contaminated (at least one sample of s points is composed of only inliers) can be computed as

$$1 - (1 - (1 - e)^s)^N = p \tag{B.1}$$

So, the number of samples necessary to reach the probability p can be calculated as

$$N = \frac{\log(1 - p)}{\log(1 - (1 - e)^s)} \tag{B.2}$$

Figure B.4 represents an i-th iteration of the RANSAC algorithm, in which the sampled model correponds to the desired one. Two random points (yellow dots) defines the sampled model (represented by white lines). The points close to the sampled models are couted (red dots), while the other ones are neglected (blue dots). It is important to highlight that the probability to find the correct model is not unitary, but increases with the number of iterations and is affected by the presence of outliers.

Table B.1 reports the necessary number of samples in order to reach a success probability $p = 0,99$ (i.e. asking for at list a random sample free of outliers), in case of different proportion of outliers (e) and number of samples composing the model (s). In conclusion, RANSAC is robust for model estimation, respect to

$s \setminus e$	5%	10%	20%	25%	30%	40%	50%
2	2	3	5	6	7	11	17
3	3	4	7	9	11	19	35
4	3	5	9	13	17	34	72
5	4	6	12	17	26	57	146
6	4	7	16	24	37	97	293

Table B.1: Number of iteration required for reaching a success probability p in presence of an outliers proportion e

outliers presence or noise in the input data. As a main drawback, the probability to obtain the correct model is not unitary and trongly depends on the number of iterations, sample dimensions and percentage of outliers. Consequently, the algorithm parameters have to be tuned from time to time, on the basis of the application, quality of the samples and desided results.

B.3 Hough Transform

The *Hough Transform* is an algorithm for extraction of lines or other parametric curves from images. It was introduced in 1962 and used for the first time for finding lines in images in 1972 [5]. The algorithm has the goal of finding the location of lines, circles or other models, provided that their parametric equations are known. It gives robust detection in spite of noise or outliners presence. Even if the algorithm was borned for working on camera images, it can be exploited also for lines or model extractions for laser scanner data.
In case of line detection, the solution is of course the line position, which is expressed in polarcoordinates and parameterized in terms of distance and angle. This means that the algorithm performs a research within a certain grid, defined by the ranges of the two parameters and their discretization quantums. The latter defines both solution precision and research computation time.

The most important algorithm parameters are:
- Δ_d, quantum for distance discretization

- Δ_θ, quantum for angle discretization
- n, number of points composing the pointcloud
- d_{max}, d_{min}, which identifies the distance range in the research grid.

A simple example of lines detection from laser scanner data by means of the Hough transform is proposed below, in order to clarify its working principle. Defined the counter i, indicating the point currently considered, the algorithm evolves through the following steps:

1. An empty matrix is created, for collecting the votes for different lines. The matric indexes are distance and orientation, while the cell value contains the number of votes.

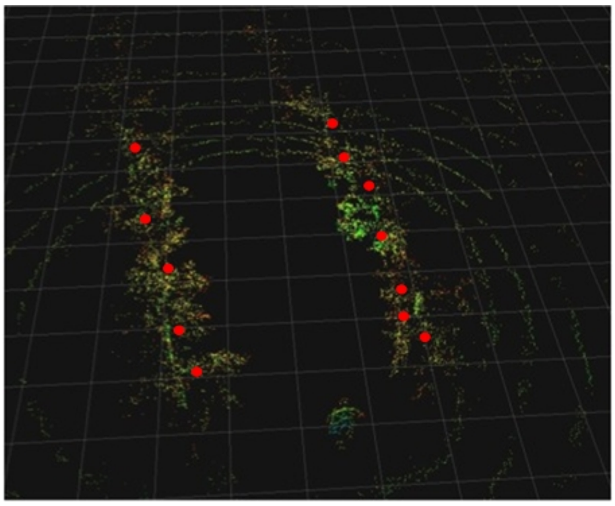


Figure B.5: Pointcloud items considered by the Hough transform algorithm

2. Considered the $i-th$ point, among the n present in the pointcloud, the following procedure is repeated. An example is shown in figure B.5, where, on a real laser scanner pointcloud sampled within two rows of a plum field, some $i-th$ points are highlighted by red dots.
3. A beam of lines going through the $i-th$ point is generated, oriented from 0°to 179°and separated by the quantum Δ_θ. Figure B.6 shows how the algorithm generates the beam of lines, intersecting some of the pointcloud items.
4. Each line of course intersects some cells of the discretized space. For each cell intersected by each line, the corresponding vote in the matrix cell is incremented. The simple example in figure B.6 shows an image of this procedure

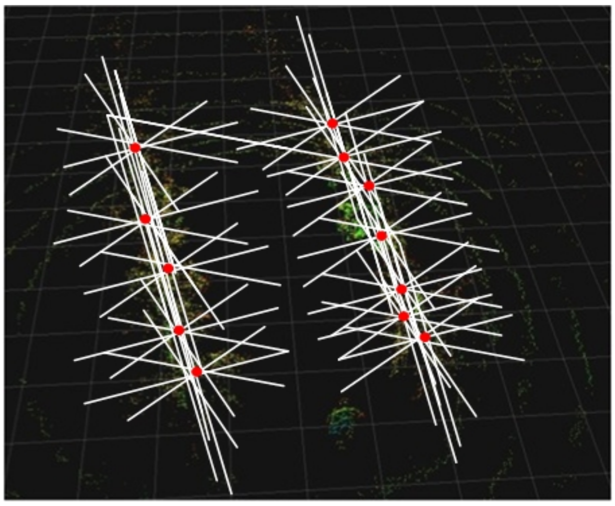


Figure B.6: Beams of lines generated by the Hough transform algorithm

result, highlighting that the space points more intersected by generated lines are the ones which identifies the desired models, i.e. the row lines.

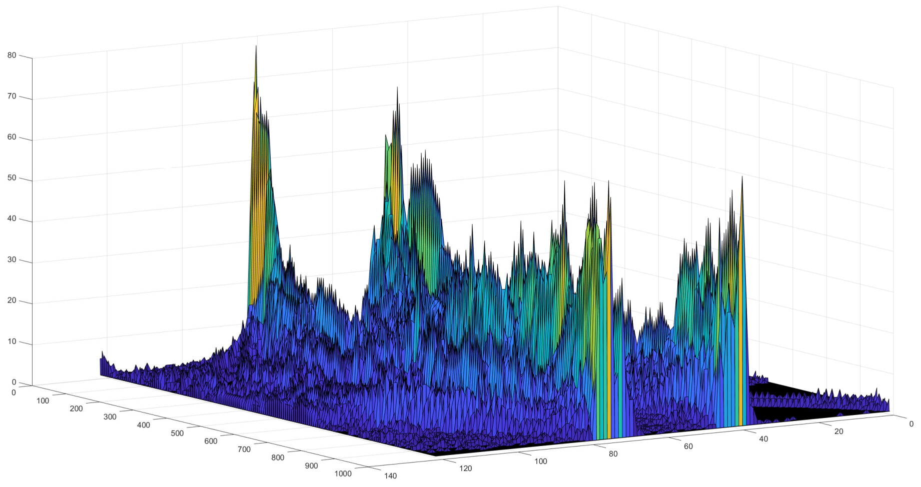


Figure B.7: Representation of a possible resulting vote matrix.

5. When the procedure is finished, the algorithm looks for the highest peaks in the matrix, i.e. for the points that receives the highest number of votes. Figure B.7 reports a vote matrix obtained running the Hough transform on LIDAR data collected at the entrance of an orchard row. The x axes represents the angle θ, discretized in 1000 cells, the y one the distance,

discretized in 140 cells, while on z direction there are the matrix cells value, i.e. the votes.

In the last step of the procedure, the solution selection can be of course polarized, on the basis the desired lines characteristics. For example, the algorithm can look for a single line, rather then two or three, or select lines with a certain mutual orientation, like parallel or perpendicular ones.

Differently from RANSAC, in the Hough transform the number of cycles is not a parameter which can be tuned, but depends on input data dimensions, meaning that the solution will be for sure present in the vote matrix. As main drawbacks, the computation time is quite high and, in case of very noisy data, the vote matrix could result *dirty* (see figure B.7) and could be not obvious to identify the solution.

Appendix C

Software data structures

This appendix describes data structures stored in the *Robot Control* software and containing informations about robot and environment. In particular, they contain the foundamental parameters of orchards, where the robot can act, and vehicle itself. In this way, the Robot Control knows informations about orchards available robotic platforms. When the user connects to the HMI and defines a mission, it choose also robot and orchard, whose parameters are automatically loaded on the Robot Control. At this point the control system is ready to act in the selected field with the choosen robotic flatform. In the current impementation, these parameters are defined in *yaml* files, which are recalled in the launch files when the Robot control is started and whose data is loaded in the *ROSParam server*. Then, during initialization, each *node* can take from the server the parameters it needs, based on mission directives received from the HMI, and fill its *field* or *robot* structures. In the following two sections, the structures related to *field* and *robot* are presented, describing the contained parameters and explaining why the latter are important for mission management and navigation.

C.1 Field parameters

Field parameters are foundamental both for mission management and trajectory calculation and for localization. The *field structure* contains informations about orchard geometry and cultivation. When the robot receives a mission, it needs for first to define a path which leads the vehicle from the beginning of the first row to the end of the last one. For this reason, informations about *number* and *lenght* of rows (see left side of figure C.1) are foundamental, in order to allow the robot to calculate the best trajectory, in terms of time or distance, and estimate the amount of energy needed for that work. Localization needs informations

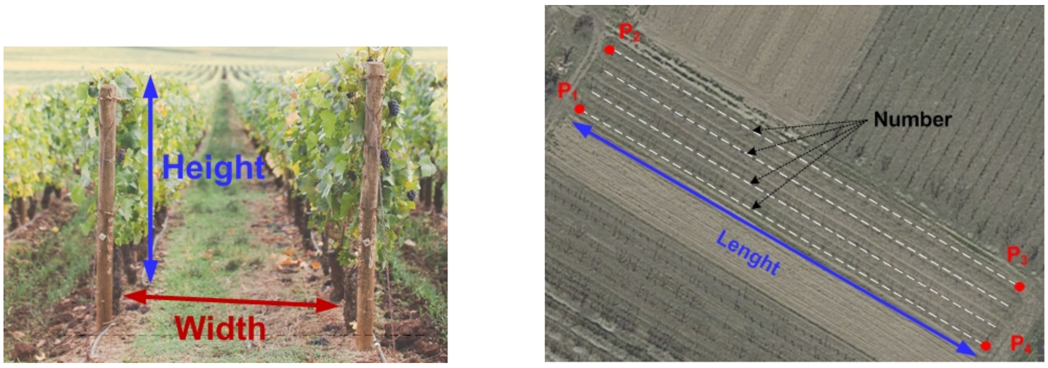


Figure C.1: Representations of the field row structure informations

about row *width*, foundamental for extracting the right row lines and exploiting the *internal model* embedded in the algorithm. Moreover, also the kind of cultivation is an important parameter, since implicitly contains informations about trees, like trunk dimensions or canopy features. This data is used for tuning the row lines estimation algorithm, making it capable of working with completely different kinds of cultivations. Finally, some foundamental *GPS* coordinates (see points P_1, P_2, P_3, P_4 on right side of figure C.1), for example related to the extreme rows of the orchard structure or to the rows extremities, help the Robot Control in estimating the entrance or exit from a row or giving redundant informations about vehicle position.

C.2 Robot class

Localization, position estimation and control algorithms need to know about some foundamental robot parameters. Informations like *wheelbase* and track wheel *radius* are employed by the control algorithm for estimating vehicle kinematics and calculating correct reference speeds, while *height* or *lenght* of the vehicle are foundamental in order to estimate its volume and evaluate if a certain obstacle can be bypassed or for avoiding collisions during stricht maneuvers (see figure

C.2). Moreover, robot *weight* is foundamental for estimating friction effects and

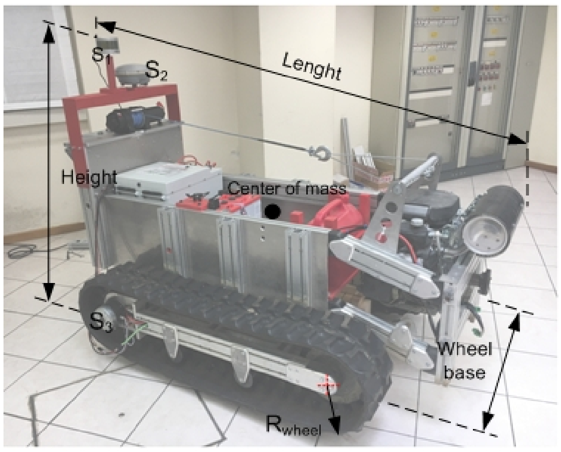


Figure C.2: Representations of the robot structure informations

evaluating the power consumption. Finally, informations about onboard *sensors* tell the localization or position estimation algorithms on which informations can rely, rather than neglect.

Bibliography

[1] Paulo Peixoto Alireza Asvadi, Cristiano Premebida and Urbano Nunes. 3d lidar-based static and moving obstacle detection in driving environments: An approach based on voxels and multi-region ground planes. *Robotics and Autonomous Systems*, 83:299–311, September 2016.

[2] Albert-Jan Baerveldtrn Bjorn Astrand. A vision based row-following system for agricultural field machinery. *Mechatronics*, 15:251–269, 2005.

[3] Marcel Bergerman Bradley Hamner, Sanjiv Singh. Improving orchard efficiency with autonomous utility vehicles. *American Society of Agricultural and Biological Engineers*, June 2010.

[4] Luigi Villani Giuseppe Oriolo Bruno Siciliano, Lorenzo Sciavicco. *Modellistica, pianificazione e controllo.* 2008.

[5] Richard O. Duda and Peter E. Hart. Use of the hough transformation to detect lines and curves in pictures. *Communications of the ACM*, 15:11–15, January 1972.

[6] Martin A. Fischler and Robert C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24:381–395, June 1981.

[7] Roberto Tazzari Nicola Mimmo Flavio Callegati, Alessandro Samor‘ and Lorenzo Marconi. Autonomous tracked agricultural ugv configuration and navigation experimental results. In *Workshop: UAVs for precision agriculture and smart farming: state-of-the-art and open challenges*, May 2018.

[8] A. Bugra Koku E. ilhan Konukseven Gokhan Bayar, Marcel Bergerman. Localization and control of an autonomous orchard vehicle. *Computers and Electronics in Agriculture*, 115:118–128, July 2015.

[9] Erhan i. Konukseven Ahmet B. Koku Gokhan Bayar, Marcel Bergerman. Improving the trajectory tracking performance of autonomous orchard vehicles using wheel slip compensation. *Biosystems Engineering*, 146:149–164, June 2016.

[10] Mürvet Krc Halil Durmus, Ece Olcay Günes and Burak Berk Üstündag. The design of general purpose autonomous agricultural mobile-robot: "agrobot". In *Fourth International Conference on Agro-Geoinformatics (Agro-geoinformatics)*, September 2015.

[11] Noboru Noguchi Hao Wang. Autonomous maneuvers of a robotic tractor for farming. In *Proceedings of the 2016 IEEE/SICE International Symposium on System Integration*, December 2016.

[12] Han-Sil Kim Hwan-Seok Choi, Ok-Deuk Park. Autonomous mobile robot using gps. In *International Conference on Control and Automation*, June 2005.

[13] Juan I. Nieto James P. Underwood, Gustav Jagbrant and Salah Sukkarieh. Lidarbased tree recognition and platform localization in orchards. *Journal of Field Robotics*, 32:1056–1074, May 2015.

[14] Anis Koubaa. *Robot Operating System (ROS): The Complete Reference (Volume 1)*, 2016.

[15] Don Kim Luis Serrano and Richard B. Langley. Multipath adaptive filtering in gnss/rtk-based machine automation applications. In *IEEE Xplore*, July 2010.

[16] Q. Zhang M. Kise and F. Rovira Mas. A stereovision-based crop row detection method for tractor-automated guidance. *Biosystems Engineering*, February 2005.

[17] Ji Zhang Gustavo M. Freitas Bradley Hamner Sanjiv Singh Marcel Bergerman, Silvio M. Maeta and George Kantor. Robot farmers. *IEEE Robotics and Automation Magazine*, March 2015.

[18] Brian Gerkey Morgan Quigley and William D Smart. *Programming Robots with ROS: A Practical Introduction to the Robot Operating System*, 2015.

[19] Wyatt Newman. *A Systematic Approach to Learning Robot Programming with ROS*, 2017.

[20] Vijay K. R. Chenchela Venkata Siva Prasad. CH Palepu V. Santhi, Nellore Kapileswar. Sensor and vision based autonomous agribot for sowing seeds. In *International Conference on Energy, Communication, Data Analytics and Soft Computing*, 2017.

[21] R. Keinprasit S. Khoomboon, T. Kasetkasem and N. Sugino. Increase a standalone gps positioning accuracy by using a proximity sensor. In *IEEE Xplore*, May 2010.

[22] Liqiong Tang Samuel J.O. Corpe and Phillip Abplanalp. Gps-guided modular design mobile robot platform for agricultural applications. In *Seventh International Conference on Sensing Technology*, 2013.

[23] R.F. Stengel. *Flight Dynamics*. Princeton University Press, 2004.

[24] Carl Wellington and Anthony (Tony) Stentz. Online adaptive rough-terrain navigation in vegetation. In *Proceedings of the 2004 IEEE International Conference on Robotics and Automation*, April 2004.

[25] Tamar Flash Yael Edan, Dima Rogozin and Gaines E. Miles. Robotic melon harvesting. *IEEE Transactions on Robotics and Automation*, 16(6), December 2000.

[26] Kentaro Nishiwaki Masahiro Saito Yutaka Kikuchi Yoshisada Nagasaka, Katsuhiko Tamaki and Kyo Kobayashi. Autonomous rice field operation project in naro. In *Proceedings of the 2011 IEEE International Conference on Mechatronics and Automation*, August 2011.

[27] M. Schoenfisch J. Billingsley. the successful development of a vision guidance system for agriculture. *Computers and Electronics in Agriculture*, 16:147–163, August 1997.