POLITECNICO DI MILANO

Facoltà di Ingegneria
Scuola di Ingegneria Industriale e dell'Informazione
Dipartimento di Elettronica, Informazione e Bioingegneria

Corso di Laurea Magistrale in
Ingegneria Informatica

Un modello per tesi di laurea magistrale al DEIB

Relatore:

PROF. MATTEO MATTEUCCI

Correlatore:

????

Tesi di Laurea Magistrale di:

GIOVANNI BERI
Matricola n. 852984

Anno Accademico 2016-2017

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

ACRONYMS

GRAPE	Ground Robot for vineyArd Monitoring and ProtEction
ROS	Robot Operating System
LIDAR	Laser Imaging Detection and Ranging
UGV	Unmanned Ground Vehicle
ICC	Instantaneous Center of Curvature
IMU	Inertial Measurement Unit
ECHORD++	European Clearing House for Open Robotics Development Plus Plus
API	Application Programming Interface
EKF	Extended Kalman Filter
SLAM	Simultaneous Localization And Mapping
AMCL	Adaptive Monte Carlo Localization

1 BACKGROUND AND TOOLS

In this chapter we are going to describe the general concepts this thesis deals with, together with the main tools we used to address the project. Since this thesis is in the frame of GRAPE project (see Chapter 2), most of them are typical of the robotic field and, more specifically, of the agricultural robotics. This last field should be seen in the wider context of the so-called *E-agriculture*; to give a precise definition of this term, we make reference to the FAO (Food and Agriculture Organization) definition[1]:

> *E-agriculture, or ICTs in agriculture, is about designing, developing and applying innovative ways to use ICTs with a primary focus on agriculture. E-agriculture offers a wide range of solutions to agricultural challenges and has great potential in promoting sustainable agriculture while protecting the environment.*

The GRAPE project, that will be described with further details in chapter 2, is about the design and realization of an Unmanned Ground Vehicle (UGV) with control and operative task in a vineyard environment, so in this chapter we'll deal with topics concerning software development in robotics, estimation of the state of a robot, autonomous navigation

ROBOT OPERATING SYSTEM

ROS is the *robotic middleware* we used to develop the software components of the system described in this thesis. We decide to use it because of its great modularity, the availability of a very large number of packages, well documented Application Programming Interface (API)s and an active community. Moreover, ROS is a very widespread system, so its power and versatility are well known in the field of software development for robotics. Citing words from its official website[2], these are ROS main features:

> *It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also pro-*

1 http://www.fao.org/fileadmin/templates/rap/files/uploads/E-agriculture_Solutions_Forum.pdf

2 http://wiki.ros.org/ROS/Introduction

vides tools and libraries for obtaining, building, writing, and running code across multiple computers.

ROS is actually a *meta-operating system*, that is, it's not an operating system in the traditional sense (it requires to be run on top of an another operating system; currently, the only officially supported OS is Linux Ubuntu), but it provides a peer-to-peer network that processes can use to create and process data together. This network is implemented through TCP, and it's called *Computation Graph*. In this section, we're going to describe ROS with more detail, with particular emphasis on the different techniques that nodes can use to communicate among them. Keep in mind that, even if these techniques differs a lot, they all are strongly typed *i.e.* in order to define a channel (with *channel* we now mean one of the technique that we are going to describe. It's not the name of a specific communication tool) you also need to define the types of message that are going to be exchanged through it. ROS already defines a lot of useful message types (*e.g. LaserScan.msg*, *PoseWithCovarianceStamped.msg*), grouped by domain (*e.g.*, *Sensor_msgs*, *Geometry_msgs*). However a simple message definition language is provided, and users are encouraged to define their own message types to make them as self-explanatory as possible.

ROS MASTER Even if the Computation Graph is a peer-to-peer network, a central process, called **ROS Master**, is required to exist, to provide naming and registration services to all the user processes In this. Once the processes have located each other through the services offered by the Master, they can communicate peer-to-peer without involving a central entity;

NODES The processes that are in the Computation Graph are called **nodes**, and they are the atomic units of the computational graph. The ROS API are available in C++, Python and Lisp, but C++ is the most widely used. One of the aims of ROS is to be modular at a fine-grained scale, so a complex task should be achieved through cooperation of several different nodes, each with quite narrow tasks, rather than one large node that include all the functionalities. Nodes can use different techniques for communication, depending whether the message is a part of data stream or it is a request message (*i.e.* a response message is expected) and, in this last case, on the (expected) duration and complexity of the computation of the response.

TOPICS Topics implements a *publish-subscribe* paradigm, are they the easiest way that nodes can use to communicate with each other, and basically are named channels, characterized by the type of the messages that are sent through it. When a node *publish* a message on a certain topic, the message is read from all the nodes that previously *subscribed* to that topic, interfacing with the Master. Note that:

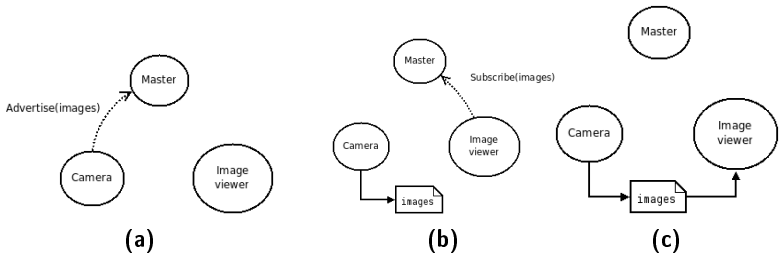


Figure 1.1: *Three phases of the setup of communication of nodes throught topics.*

- this technique leads to a strong decoupling between publishers and subscribers to a topic, because a publisher node is, from an high-level perspective[3], not even aware of the presence of subscribers, and vice versa.
- the relationship between publishers and subscribers is *many-to-many*, *i.e.* multiple nodes can publish on a topic, and multiple nodes can subscribe to a topic.

We can easily conclude that this method is very suitable for passing streams of data (*e.g.* the handler of a LIDAR streams its measurements over the network, or a node publish the velocities commands for the wheels of a robot), but there is no notion of a *response* to a message, so it's not suitable for *request-response* communication.

SERVICES Services are defined by a name, and a couple of message types that describe the *request* type and the *response* type. Each service is offered by a Service server to any Service client that perform a call. So, Services implement an inter-node communication that is very similar to traditional function calling in most common programming languages (*e.g.* C++, Java), in the sense that:

- Service calls are blocking
- using Services, the inter-node communication is *one-to-one*

These properties make Services suitable for punctual (in opposition to data stream) inter-node communication, such as: request of parameters values to another node, ask a node that handles a camera to take a picture, ask a node that perform navigation task to clear the current map.

ACTIONS While Services, with their resemblance to traditional function calls, can address pretty well the problem of *one-to-one* inter-node communication, they can be quite unsatisfying if the computation required to produce the response is demanding in term

3 Actually, publisher nodes always know the list of nodes subscribed to their topics. But this is only used in connection phase, and to avoid a situation where a node publish on a topic with no subscribers, for the sake of efficiency.

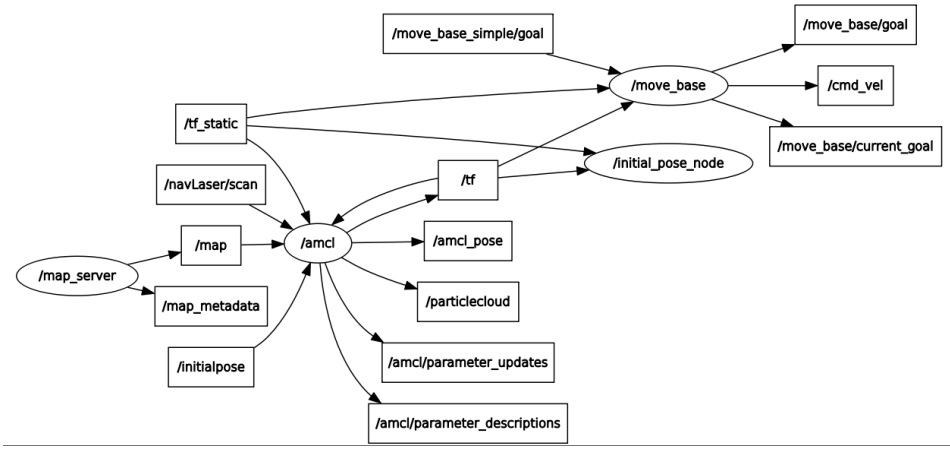


Figure 1.2: *An example of the ROS Computation Graph, visualized with* rqt*: nodes are represented with circles, rectangles represent topics, and arrows go from a node to a topic it publishes on, or from a topic to a node subscribed to it. It's easy to recognize the* many-to-many *relationship.*

of execution time (*e.g.*, navigation of a robot from one point to another in an environment), because the caller is stuck at the line with the Service invocation until the end of the procedure. Services show their weaknesses also in situations where it could be useful to observe the intermediate results of the computation triggered by the request (*e.g.*, a very complex manipulation procedure). **Actions** are very suitable in this context because, at the cost of a more complex implementation, provide an asynchronous and fully preemptable remote procedure call, with the possibility of monitoring intermediate results if needed, and a native exit status to check the state (active, rejected, preempted, succeeded, aborted...) of the execution. Differently from Topics and Services, Actions are not native in ROS, and their functionalities are built on top of the other ROS messaging systems (See figure 1.4). Asynchronicity is provided by the use of callbacks.

A final consideration about the ROS mechanism for recording, and playing back, data published on topics. This software module is call *rosbag*, and takes advantage of the fact that ROS is aware of all the messages exchanged through its messaging system. This tool gets very useful in a project like GRAPE, where is not trivial at all to simulate or replicate in a laboratory environment the physical context of a vineyard, and of course meaningful data are required in order to correctly validate algorithms and tools. The paradigm is very simple: in every moment you can invoke a *rosbag record* command, that records every single message exchanged on the ROS topics; data are logged in a single file with *.bag* extension, that can be lately filtered and/or compressed. Bag files can be played back in a very easy way invoking *rosbag play* command; the playing of the bag is implemented with a single node that streams messages on all the topics existing in the bag

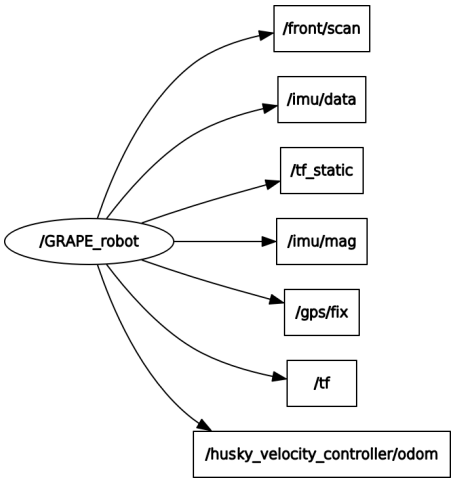


Figure 1.3: *Visualization of the ROS Computation Graph with* rqt *in the context of a data bag being played: there is a single node* (GRAPE_robot) *that simultaneously plays recorded topics messages.*

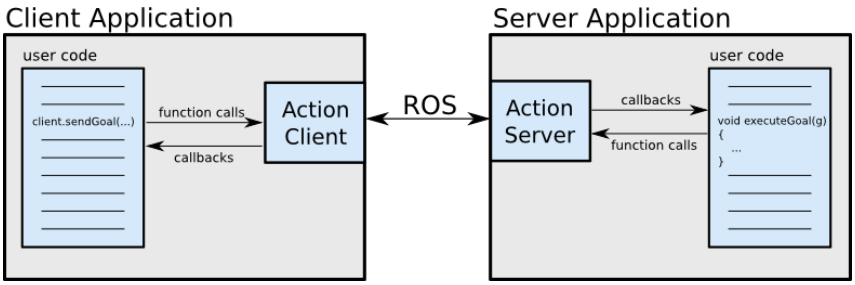


Figure 1.4: *Sketch of the implementation of actionlib, through ROS topics and a callback system.*

(see Figure 1.3). The module also provides an option to use *simulated time*, *i.e.* playing the bag exactly with the same message timing as in the real recorded session. In this situation, the decoupling induced by *publish-subscribe* is a strong advantage, since you can substitute all publishing nodes with a single one, with no consequences[4].

TF: THE TRANSFORM LIBRARY

tf is a ROS library, which task is very important to understand in order not to get lost in the next sections and chapters. The goal of *tf* is:

> *" [...] provide a standard way to keep track of coordinate frames and transform data within an entire system such that individual component users can be confident that the data is in the coordinate frame that they want without requiring knowledge of all the coordinate frames in the system" (Foote, 2013)*

4 Except in some pathological cases: for example, a node can require the list of active nodes in the computational graph, and the response would be different when the data are recorded and when the corresponding bag is played.

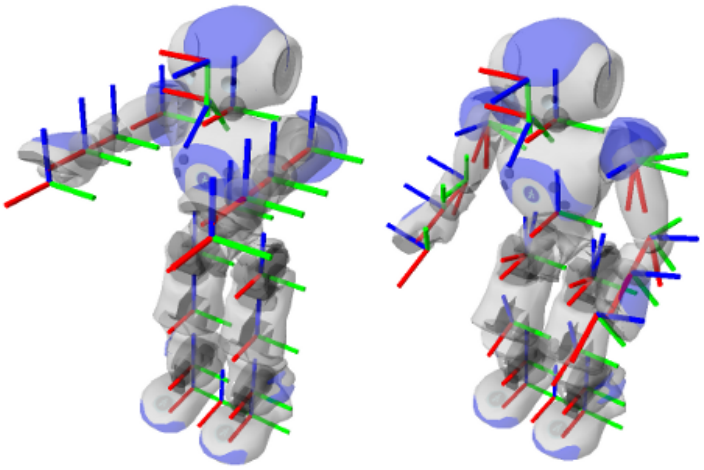


Figure 1.5: *A robot in 2 different positions, with* tf *frames in evidence: x-axis is red, y-axis is green, z-axis is blue. The frames are the same in both configuration, but the transformations* (i.e. *rototranslations) between them are different.*

The utility of such a component is straightforward, even in quite simple robotic systems. We'll describe here a situation we stumbled upon exactly in the development of the GRAPE project, where of the utility of *tf* is very easy to understand; you'll be able to better contextualize this example after you've read Chapter **??**. In this example a LIDAR, mounted on top of the final joint of a robotic arm, acquires data while the arm is moving in order to create a point cloud that will be processed later. To get a meaningful point cloud, it's mandatory to keep track of the movement of the LIDAR with respect to a point with speed equal to zero (*e.g.* the base link of the arm, or the base link of the whole robot), and this gets even more difficult because of the multiple (6 in our specific case) joints of the arm; but this problem can be easily addressed by means of *tf*.
Note that frames are very useful for two main tasks:

- represent the configuration of the robot, by assigning frames to the significant physical elements of the robot (*e.g.* joints, sensors, wheels). Since *tf* graph is a tree, a root element of the robot should exists; in a typical configuration, the frame name is *base_link*, it's placed in the geometrical center of the robot, and all the other frames linked to physical components of the robot belong to the subtree with root *base_link*.

- represent the position of the robot in an environment. A typical example is the frame *odom*, that is typically centered in the point where the robot is located when it's switched on. Another example is the frame *map*, that is used as a reference in case the robot is localized not only with respect to its initial point, but in a given map.

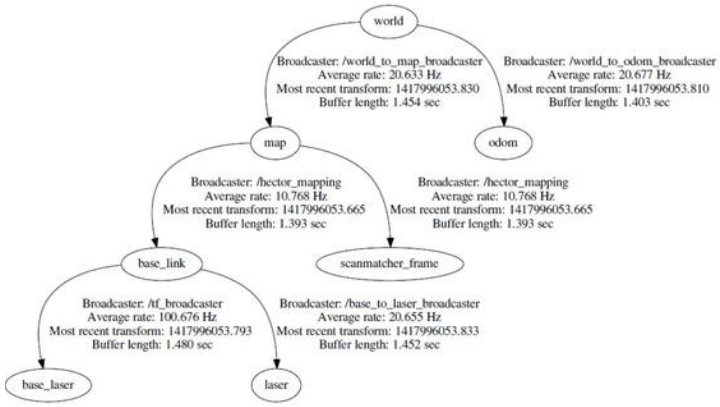


Figure 1.6: *An example of* tf *tree*

We are now giving a a bit more techical detail about *tf*. *tf* implementation relies on ROS topics (see Section 1.1) and achieves the goal mentioned before by building an oriented graph where vertices are reference frames, and edges are transformations (rototranslations) between frames. *tf* does not assume a constant structure and, if a path exists between two reference frames in the graph, the direct transformation between them can be computed by composition of transformation. Since, in general, multiple paths between 2 vertices can exists in a directed graph and this could lead to ambiguity in computing the transformation between two reference frames, the graph is forced to be acyclic. Disconnected subgraphs are allowed, but of course transformation between vertices that belong to different subgraphs cannot be computed. The main components of the library are:

- ***tf* broadcasters**: they are simple software components, that publish a transformation between two reference frames every time an update is available. Different broacasters does not sync together the publishing phase
- ***tf* listeners**: they are more complex components, because they take into account that broadcasters are not synced. Since both transformations and queries to *tf* graph are stamped, listeners make use of queues to store the most recent transformations, and they interpolate old values using SLERP (Spherical Linear intERPolation) to return a transformation for which there is no measured value at the requested timestamp.

In figure 1.5 you can see a graphical representation of the reference frames tracked in a Nao Robot, while figure 1.6 shows an example of *tf* graph visualized with visualization framework *rqt*.

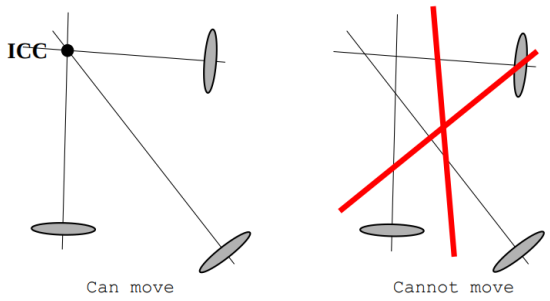


Figure 1.7: *If the axis of all the wheels intersect in a single point, it's called ICC and the robot can move without slipping*

ODOMETRY

The problem of odometry, *i.e.* estimation of the position of a robot in an environment is harder than it could seem. Formally, odometry estimation is the problem of estimating over time the tuple:

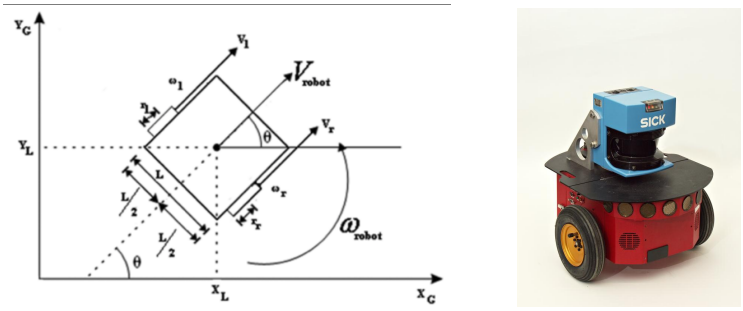
$$< x, y, z, \theta, \dot{x}, \dot{y}, \dot{z}, \dot{\theta} > \tag{1.1}$$

given the measurement of some motion sensors. To better understand the complexity of the problem, let's analyze an extremely simple model: a robot with a single, freely rotating wheel. In this frame, assuming rotary encoders on the wheel, we can think about measuring directly the wheel speed and integrate these measurement to get the traveled distance, and measure the variation in the orientation of the wheel to get the position. But actually there are a lot of imperfection that can lead to error, for example:

- wheel can be non perfectly perpendicular to the ground
- the friction between the floor and the wheel might not be enough to avoid slippage (expecially)
- there is no such thing as a perfect sensor, so the use of encoders introduce an error

Even if all these concurrent causes seem negligible, you have to take into account that the errors sum up over time, so an error of a few millimeters per meter might become significant over time.

Moreover, the probability of slippage gets higher in systems with more than one wheel, because, because for a system with multiple wheels to move without slippage, a point must exists around which all the wheels can move along a circular path. This point is called Instantaneous Center of Curvature (ICC) (see Figure 1.7), and can be easily identified by looking for the intersection of the axis of all wheels. If the intersection exists in a single point, it's called the ICC.



(a) (b)

Figure 1.8: *On figure 1.8a, the scheme of a differential drive motion model; in figure 1.8b, an example of a differential drive robot (Pioneer 3DX).*

But even if the odometry estimated from the wheels is not a good solution if used alone, it can be used as a starting point for other, more complex, method. For this reason we are now going to describe how to estimate the odometry starting from the wheels encoders in our specific robot. This computation is different according to the **motion model** of the considered robot. As we'll see in Section **??**, the robot we used is the Husky platform (see Figure 1.10) from Clearpath Robotics, that moves with a *skid steering* kinematics, that is a derivative of *differential drive* kinematics. Thus, we're going to describe these two motion model with more detail.

Differential drive robot

In a differential drive system, the movement of the robot is only based on two separately driven wheels, placed on either side of the robot, on the same axis (see Figure 1.8), and optionally a central, non-actuated caster wheel for stability. The two side wheels are not steerable, so the changes of direction are realized through application of different speed to the two wheels. For example, intuitively, if the wheels move at the same speed and in the same direction, the robot will move straight; if the wheels move at the same speed but different directions, the robot rotates in place. By recalling the definition of ICC, we observe that, if the wheels are correctly aligned, a differential drive robot always have a well-defined ICC and the slippage of the wheels is not very accentuated.

At each instant in time, since the ICC is well-defined, both the left and right wheel follow a path that moves around ICC at the same angular speed ω, and thus:

$$\begin{cases} \omega(R + \frac{L}{2}) = v_r & (1.2) \\ \omega(R - \frac{L}{2}) = v_l & (1.3) \end{cases}$$

where L is the distance between the center of the two wheels, v_r and v_l are, respectively, the linear velocity of the right and left wheel, R is the signed distance between the ICC and the midpoint of the wheels. Note that the only parameter constant through time is L, since it's a physical property of the robot structure, while all other parameter evolve during the movement.
By combining 1.2 and 1.3, we get:

$$R = \frac{L}{2}\frac{(v_r + v_l)}{(v_r - v_l)}, \qquad \omega = \frac{(v_r - v_l)}{L} \tag{1.4}$$

Observing these results we can validate the intuitive impressions about particular cases made a few lines above:

- if $v_r = v_l$, the curvature radius is infinite, because the robot is moving straight.
- if $v_r = -v_l$, the robot is moving around the midpoint of the wheels

We give now some details about odometry computation. Let's assume, in a certain moment $t = t_0$, that the robot pose is (x, y, θ). We assume that in the time interval $t_0 \rightarrow (t_0 + \delta t)$ the values v_r and v_l are constant; if we observe figure 1.9 under these condition, we have:

$$ICC = (x - R\sin\theta, y + R\cos\theta) \tag{1.5}$$

Write now the expressions for (x', y', θ'):

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos(\omega\delta t) & -\sin(\omega\delta t) & 0 \\ \sin(\omega\delta t) & \cos(\omega\delta t) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x - ICC_x \\ y - ICC_y \\ \theta \end{bmatrix} + \begin{bmatrix} ICC_x \\ ICC_y \\ \omega\delta t \end{bmatrix}$$

With this procedure we have identified 3 of the elements of the target tuple (see expression 1.1), but we still have to retrieve x,y and θ. For this reason we consider that by assuming an initial pose (x_0, y_0, θ_0), knowing that:

$$V(t) = \frac{(v_r + v_l)}{2} \tag{1.6}$$

where $V(t)$ represent the overall speed of the robot, and assuming to know the functions $v_r(t)$ and $v_l(t)$ *i.e.* the linear speed of the wheel in time, we can calculate (x, y, θ) by integrating the speed of the robot over time, that is:

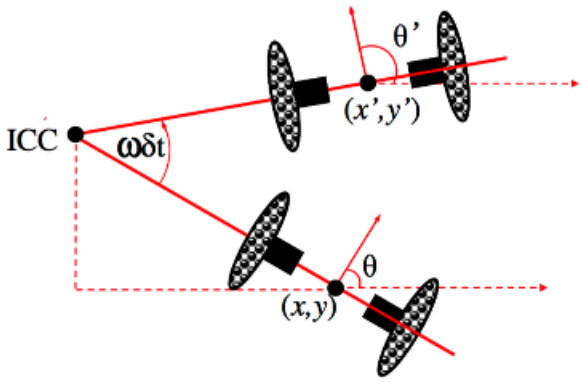


Figure 1.9: *Differential drive:* $(x, y, \theta) \rightarrow (x', y', \theta')$

$$x(t) = \int_0^t V(t)\cos(\theta(t))dt \tag{1.7}$$

$$y(t) = \int_0^t V(t)\sin(\theta(t))dt \tag{1.8}$$

$$\theta(t) = \int_0^t \omega(t)d\omega \tag{1.9}$$

and, in our specific case we can write it as function of v_l and v_r, that are the quantities that are directly measured on the wheels.

$$x(t) = \frac{1}{2}\int_0^t (v_r(t) + v_l(t))\cos(\theta(t))dt \tag{1.10}$$

$$y(t) = \frac{1}{2}\int_0^t (v_r(t) + v_l(t))\sin(\theta(t))dt \tag{1.11}$$

$$\theta(t) = \frac{1}{L}\int_0^t (v_r(t) - v_l(t))(t) \tag{1.12}$$

So the tuple required by the odometry calculation is now complete.

Skid-steering

However, observing image 1.10, it's very easy to contest that the Husky platform used in GRAPE project is not similar to the model described in previous section, because the number of actuated wheels is four instead of two. However, the motion model of the Husky is more similar to the differential drive because:

- wheels are not steerable

Figure 1.10: *Husky platform from Clearpath Robotics is the platform used for the development of GRAPE project.*

- being $v_{f\star}$ the speeds of the front wheels, $v_{r\star}$ the speeds of the rear wheels, $v_{\star r}$ the speeds of the right wheels, $v_{\star l}$ the speeds of the left wheels, we always have:

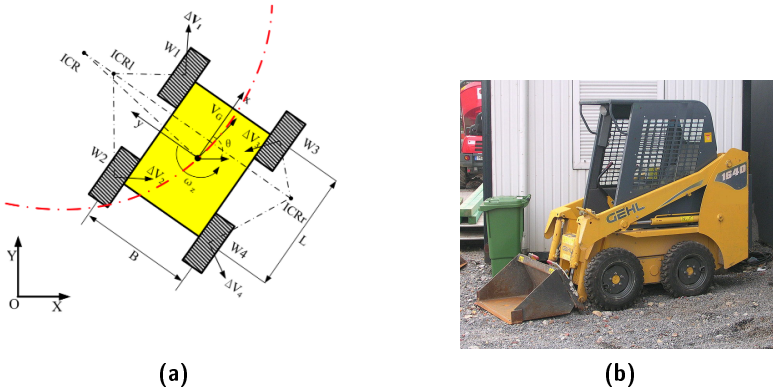
$$v_{fr} = v_{rr} \tag{1.13}$$

$$v_{fl} = v_{rl} \tag{1.14}$$

This type of motion model is called *skid steering*, and is often use in real-world applications (see figure 1.11b) because of its simple and robust mechanical structure that leaving more room in the vehicle for the mission equipment. In addition, it has good mobility on a variety of terrains, which makes it suitable for all-terrain missions. But, of course, it also present a variety of problems and weakness. For example, if you recall what we told about ICC in Section 1.3, it's clear that the ICC of such a model will always be at the infinite, since wheels are organized in 2 parallel rows that cannot steer. Thus, skid steering robots can't turn without slipping of the wheels! From a theoretical point of view, the main consequence is a a complexity in obtaining an accurate kinematics and dynamic model of *skid steering* (Yi et al., 2007). On the other hand, in the context of GRAPE project this leads to two major practical problems:

- the slippage of the robot leads to higher power consumption of the electric engines with respect to the a system with explicit steering (Shamah, 1999). This is something to be taken into account in the sizing phase of the power system of the robot.
- the estimation of the odometry is going to be much more imprecise than in the differential drive case, for the error introduced by the slipping of the wheels.

Actually, there is a quite simple way (Wang et al., 2015) to model with an acceptable approximation the skid steering as an equivalent differential drive system, where the distance between the wheels is



(a) (b)

Figure 1.11: *On figure 1.11a, the scheme of a skid steering motion model; in figure 1.11b, an example of a skid steering vehicle.*

obtained multiplying the original distance for a factor χ that is function of the physical structure of the robot This approximation relies on some assumptions:

- the mass center of the robot located at the geometric center of the body frame.
- the two wheels of each side rotate at the same speed.
- **the robot is running on a firm ground surface, and four wheels are always in contact with the ground surface.**

We cannot guarantee that conditions 1) and 2) will hold in our context, but we are almost sure that condition 3) is **not** going to be verified, given the condition in which our robot is going to operate (vineyard terrain), so this is another reason for the wheel odometry not to be very precise. Thus, the integration of several sensors beyond the wheels encoders is essential to correct the wheel odometry.

SENSOR FUSION

First of all, clarify what we mean for sensor fusion, following the definition of Elmenreich, 2002:

> *"Sensor fusion is the combining of sensory data or data derived from sensory data in order to produce enhanced data in form of an internal representation of the process environment. The achievements of sensor fusion are robustness, extended spatial and temporal coverage, increased confidence, reduced ambiguity and uncertainty, and improved resolution."*

Let's analyze this last list, element by element, recalling the analysis of Remagnino, Monekosso, and Jain, 2011:

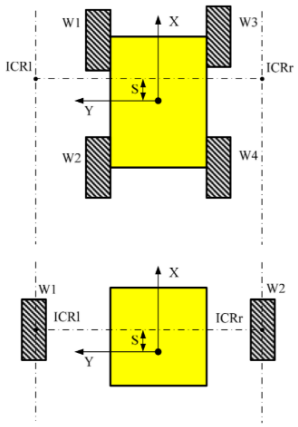


Figure 1.12: *The equivalence of differential drive motion model and skid steering model according to Wang et al., 2015*

- *robustness*: redundancy generated by the presence of multiple sensors make the robot more resistant to partial failures
- *extended spatial and temporal coverage*: disseminate sensors are more likely to measure the dimension concerned with temporal continuity and from several positions
- *increased confidence*: measures confirmed by more that one sensor are given a larger weight
- *reduced ambiguity and uncertainty*: joint information reduces the set of ambiguous interpretations of the measured value
- *robustness against interference*: by increasing the dimensionality of the measurement space (*e.g.*, measuring the desired quantity with optical sensors and ultrasonic sensors) the system becomes less vulnerable against interference.

Even if the name could be misleading, sensor fusion is different from *multi-sensor integration* in the sense that multi-sensor integration only consists in the simultaneous use of disparate sensor sources in order to accomplish a goal task. Sensor fusion techniques make a step further, and aim to the construction of a single common representation, using all the available sensor sources. The difference between multi-sensor integration and sensor fusion is described graphically in image 1.13, to underline that sensor fusion actually provides one single representation of the state of the environment that is used as a single input stream by the control application, while in multi-sensor integration the application uses each of the sensor data streams as direct input. In this context, we are assuming the definition of environment, from Thrun, 2002: pose of the robot, velocity of the robot, and velocity of the joints, configuration of actuators, location and fea-

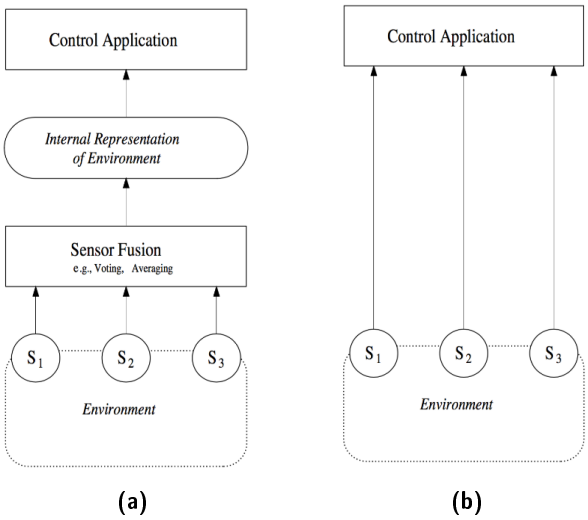


Figure 1.13: *Sensor fusion 1.13a vs multi-sensor integration 1.13b*

tures of surrounding objects, location and velocity of moving object and people.

In ROS ecosystem, several open source solutions exist that implements *sensor fusion-based* odometry; we tried two of them, which gave proof of good functioning in the past: **ROAMFREE** (Cucci and Matteucci, 2013, Calabrese, 2014, Cucci and Matteucci, 2014), a graph-based algorithm, and **Robot Localization** (Moore and Stouch, 2014, Dudek, Szynkiewicz, and Winiarski, 2016, Mohanty et al., 2016), based on Kalman filters.

After the experimental campaign in Casciano Terme, ROAMFREE turned out to be harder to configure for our specific problem, while Robot Localization had some specific documentation about the its usage in the required context[5], so we opted for this last one. Further details about our usage or Robot Localization will be given in Chapter 3, while we are now giving a few hints about its general functioning.

Robot Localization

Citing Robot Localization documentation[6]:

> *"Robot_localization is a collection of state estimation nodes, each of which is an implementation of a nonlinear state estimator for robots moving in 3D space. It contains two state estimation nodes, ekf_localization_node and ukf_localization_node". In addition, robot_localization provides navsat_transform_node, which aids in the integration of GPS data."*

5 http://docs.ros.org/kinetic/api/robot_localization/html/integrating_gps.html

6 http://docs.ros.org/kinetic/api/robot_localization/html/

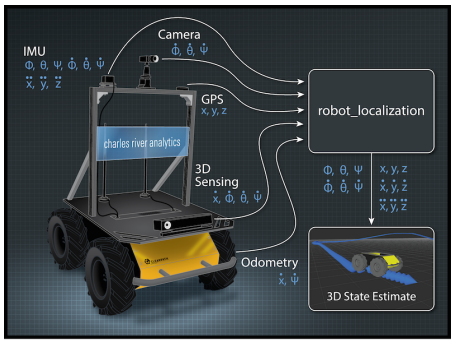


Figure 1.14: *A scheme of Robot Localization senor fusion*

The Robot Localization algorithm is implemented specifically for ROS, and it's widely adopted by ROS community for its documentation, and its easiness of use and configuration. It fuses an unlimited number of sensor sources, for each of which you can specify a configuration vector given in the frame id of the input message *i.e.* you can specify the message fields to be fused in the global estimate. For example, since GPS output is not so precise about altitude estimation, you can specify that only latitude and longitude parameters to be fused in the global estimate.

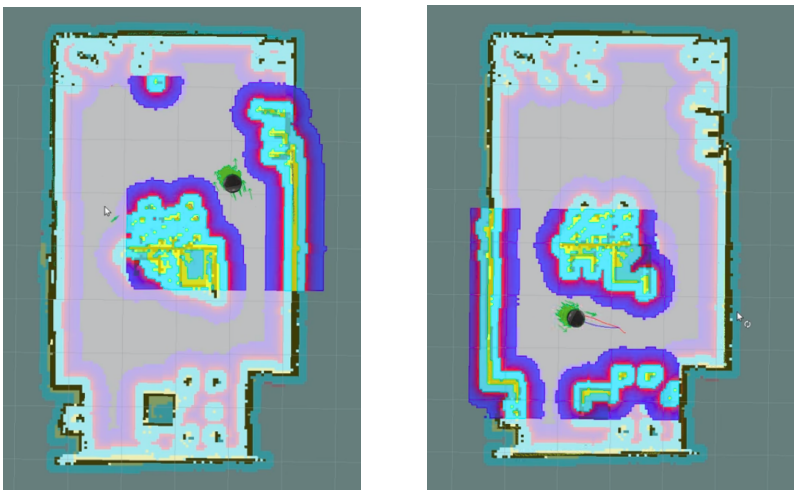
The state variables considered by Robot Localization are:

$$< x, y, z, \psi, \theta, \phi, \dot{x}, \dot{y}, \dot{z}, \dot{\psi}, \dot{\theta}, \dot{\phi}, \ddot{x}, \ddot{y}, \ddot{z} > \tag{1.15}$$

where ψ, θ, ϕ correspond to the Euler angles *roll, pitch, yaw*. The node accepts as inputs several types of ROS messages:

- *nav_msgs/Odometry*: the odometry estimation output by another node can be used entirely as input; this is the case of the odometry estimated only using wheels encoders, and it also allows for cascaded localization nodes. This will be useful in the GRAPE project, as we'll se in Chapter 3.
- *sensor_msgs/Imu*: the output of an IMU sensor *i.e.* a device composed by accelerometer, gyroscope, and magnetometer
- *geometry_msgs/PoseWithCovarianceStamped*: an estimate of the position and orientation of the robot, together with the timestamp of the measure and its covariance matrix
- *geometry_msgs/TwistWithCovarianceStamped*: an estimate of the linear and angular velocities of the robot, together with the timestamp of the measure and its covariance matrix.

Moreover, the Kalman filter implemented in Robot Localization is capable of *continuous estimation*: if for some reason no data are received from the various sensors for a large enough amount of time, the filter



(a) (b)

Figure 1.15: *Local and global costmap visualized with* RViZ*. As you can see, the local costmap (in brighter colors) is built around the robot, and moves together with it.*

keeps producing odometry estimation exploiting an internal motion model. Remember that Robot Localization can be configured also to publish the *tf* transformation associated to the estimated odometry. This is one of the usage of *tf* described in Section 1.2.

NAVIGATION STACK

In Computer Science, a software stack is a set of programs that collaborate, in order to achieve a common goal. Since we are talking about ROS ecosystem, the programs are ROS nodes, and in the case of Navigation Stack, the common goal is to provide a modular out-of-the-box navigation system for UGVs. It was originally developed for Willow Garage's *PR2* robot, but its usage can be easily extended to differential drive and holonomic wheeled robots. The planner, Move Base, is known to be a very good solution for indoor navigation (Marder-Eppstein et al., 2010), so we were not sure about its performance in an outdoor environment as required in GRAPE project. Luckily, it turned out to be a very good solution even in our situation, so it was integrated in the final navigation system. We are now giving a brief description of the main building blocks of the Navigation Stack (figure 1.16):

ODOMETRY SOURCE A topic from which read the pose estimated from the odometry system *e.g.* simple wheels odometry, output of a sensor fusion node as Robot Localization or ROAMFREE.

SENSOR SOURCE A topic from which read the data coming from the laser sensors (LIDAR) that are probing the environment, for obstacle avoidance, localization and possibly mapping tasks.

AMCL Implementation of a probabilistic localization system, based on the Monte Carlo localization, as described in Fox et al., 1999

MAP SERVER A ROS node where the map is published as a single topic message. This block is not used if Move Base is used in mapless mode

BASE CONTROLLER This is the only output topic of the Navigation Stack, and of course it contains speed commands for the base of the robot.

LOCAL AND GLOBAL COSTMAP They represent the information about the obstacles on the 2D plane of the ground, with a certain inflation radius that represent the size of the robot base. Each cell of the gridmap is associated to a certain cost that measure "how hard is it" to traverse that cell of the gridmap; possible values to represent the severity of obstacles are *Lethal obstacle*, *Inscribed*, *Possibly circumscribed*, *Freespace*, *Unknown*. The **global** costmap represent whole environment as is build from a known map, while the **local** costmap is built using only incoming laser scans. Local map is, in general, a scrolling window that moves in the global costmap in relation to robot current pose, and it's continuously updated.

LOCAL AND GLOBAL PLANNERS Global planner takes as input the global costmap, and traces the path with lowest cost from current position of the goal position; the local planners instead takes care of continuously update (if required) the global plan in the light of the incoming laser measurements. This combination is essential in case of unexpected obstacles.

MOTION PLANNING

Motion planning is a term used in robotics for the process of breaking down a desired movement task into discrete motions that satisfy movement constraints, and possibly optimize some aspect of the movement. This process is essential for autonomous systems, that accepts high-level description of tasks and should require no further human intervention; in this way the user can focus on *what* he wants done, instead of *how* to do it (Latombe, 2012). Motion planning can be applied to any kind of mechanical device equipped with actuators and sensors under the control of a computing system (*e.g.* a robotic arm that perform *pick-and-place*, a differential drive robot that has to reach a goal in an environment). Motion planning includes (among

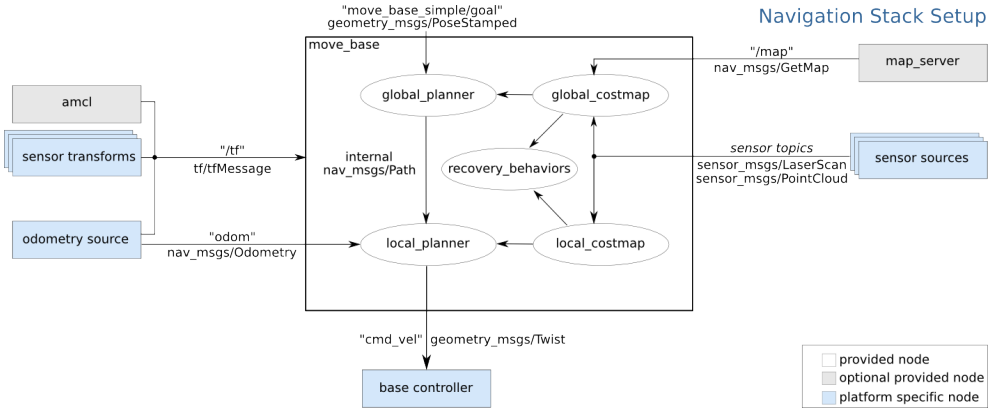


Figure 1.16: *A scheme representation of the Navigation Stack; see the color legend for classification in provided, optional provided, and platform specific nodes.*

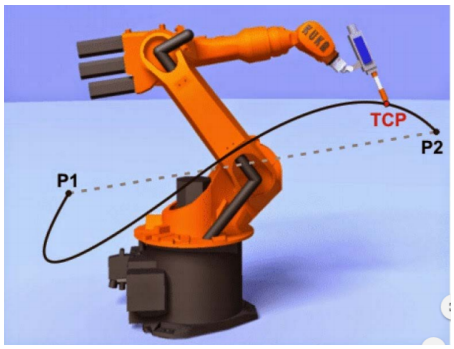


Figure 1.17: *A graphical representation of the trajectory planned for moving a robotic arm from start (P1) to goal pose (P2), satisfying geometrical constraints of the robot.*

other): obstacle avoidance, computation of collision-free paths, building reliable sensory-based motion strategies.

More concisely, the basic problem of motion planning is, given:

- a start pose of the robot
- a desired goal pose (position, orientation)
- a geometric description of the world
- a geometric description of the robot (both as geometric shape, and as kinematic constraints due to kinematic motion model of the robot)

finding a path that moves the robot gradually from start to goal while never touching any obstacle.

Even if the movement of the robot has to be executed in the real world, the planning of the motion is typically computed in the *configuration space*, or *C-space* (Lozano-Perez, 1983). Formally, let: the robot A, at a certain position and orientation, be described as a compact

subset of $W = \mathbb{R}^n$, $N = 2$ or 3, and the obstacles $B_1, \ldots, B_n$ be closed subsets of W. In addition, let F_a and F_w be Cartesian frames embedded in A and W, respectively. F_a is a moving frame, while F_w is fixed. A **configuration** q of A is a specification of the position T and the orientation Θ of F_a with respect to F_w. The **configuration space** of A is the space C of all the configurations of A. Usually, configuration space is high dimensional; a configuration is expressed as a vector of positions and orientations, so the robot in configuration space is always represented as a point. We distinguish the configuration space from the *workspace*, that is the physical environment in which the robot move. A few examples:

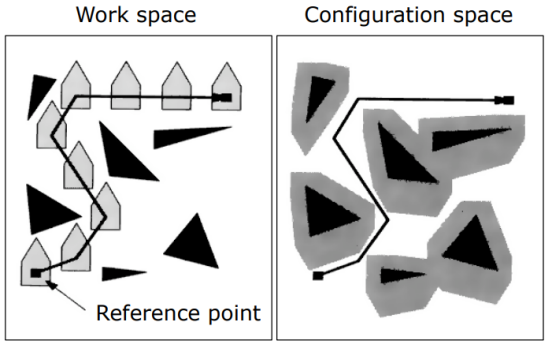
- the robot is a single point and the workspace is a 2-dimensional plane; C is a plane, and a configuration can be represented using two parameters (x, y).
- the robot is a 2D shape that can translate and rotate, the workspace is a 2-dimensional plane; C is 3-dimensional and a configuration can be represented using 3 parameters (x, y, θ).
- the robot is a solid 3D shape that can translate and rotate, the workspace is 3-dimensional; C is 6-dimensional, and a configuration requires 6 parameters: (x, y, z) for translation, and Euler angles (ψ, θ, ϕ).
- the robot is a fixed-base manipulator with N revolute joints and no closed-loops; C is N-dimensional.

Note that, as physical workspace, the configuration space is splitted in free space (C_{free}), and obstacle space (C_{obs}). From a practical point of view, *C-space* can be drawed by virtually sliding the robot shape along the edge of the obstacle regions, inflating them of the area covered by the robot. You can see some graphical examples of this procedure in Figure 1.18.

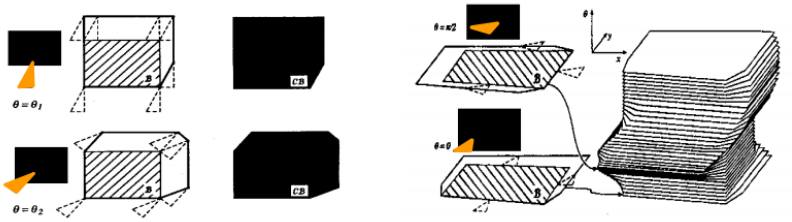
In simple configuration spaces, *grid-based search* algorithms can be used. With this approach, a grid is overlaid to configuration space, and assume that each configuration is identified with a grid point. With this approach, the problem is reduced to a graph search and exact algorithms like Dijkstra and A*, or suboptimal algorithms like A*, ARA*, AD* can be used.

But this approach gets easily unfeasible (for example, A* has complexity $O(b^d)$ (Hart, Nilsson, and Raphael, 1968), with b branching factor and d depth of shortest path), so **sample-based approaches** are currently state-of-the-art planning algorithms. Sampling bases approaches, in a nutshell:

- are more efficient in most practical problems but offer weaker guarantees

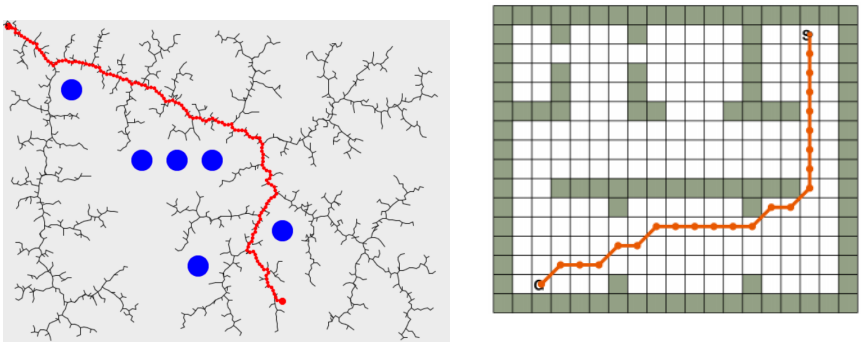


(a)



(b)

Figure 1.18: *The graphical construction of C-space from workspace, by sliding robot shape along the borders of obstacle regions. In Figure 1.18a you can see the procedure in 2D, in Figure 1.18a in 3D. Note that in 3D you only need to compute 2D procedure for each θ, and then stack all obtained images.*



(a) (b)

Figure 1.19: *Graphical representation of the ways of proceeding of a sample-based planner (1.19a), and of a grid-based planner (1.19b)*

- are probabilistically complete: increasing computing time, the probability tends to 1 that a solution is found if one exists (otherwise it may still run forever)
- performance degrades in problems with narrow passages

In sample-based planning there isn't a explicit characterizing of C_{free} and C_{obs}, but only a collision detection algorithm that probes C to see whether a certain configuration lies in C_{free} or not. An example of sample-based planning algorithm is **Rapidly exploring random trees** (LaValle, 1998); its pseudocode is shown in Algorithm 1.

Algorithm 1 RapidlyExploringRandomTrees(q_{goal})

```
1:  G.init(q_0)
2:  iterNumber ← 0
3:  N ← 100                                   ▷ for example
4:  repeat
5:      if iterNumber mod N ≠ 0 then
6:          q_rand ← RANDOM_CONFIG(C)
7:      else
8:          q_rand ← q_goal
9:      q_near ← NEAREST(G, q_rand)
10:     G.add_edge(q_near, q_rand)
11:     iterNumber ← iterNumber + 1
12: until q_near = q_goal
```

In the context of GRAPE project, motion planning is an essential component in two different modules:

1. Navigation module: **Move Base**, described in Section 1.5 is a motion planner, that makes use of grid-based planning algo-

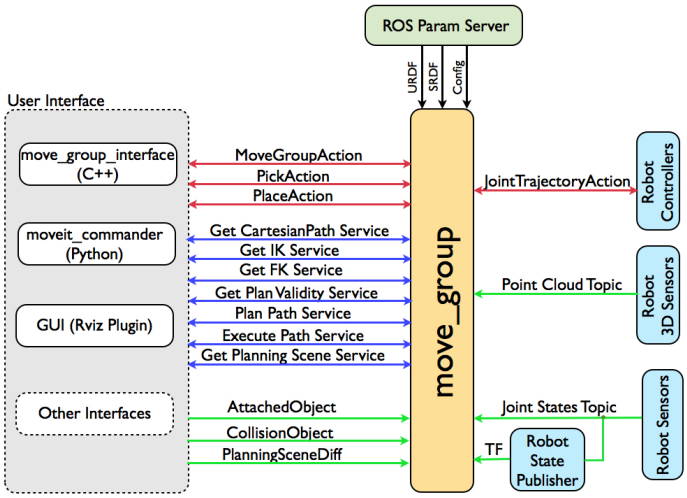


Figure 1.20: *A block scheme representing the high-level MoveIt! architecture.*

rithm, and is used for the navigation of the robotic base in the vineyard environment

2. Manipulation module: **MoveIt!**[7] is a state-of-the-art sample-based planner, and is used to plan the motion of our robotic arm in manipulation tasks (see Figure 1.20 for a block scheme of *MoveIt!* architecture).

7 http://moveit.ros.org/

2

THE GRAPE PROJECT

In this chapter, we are going to give a description of the project this thesis work is part of, with particular emphasis on the parts that were specifically addressed in the thesis work. For the description of the project, we make reference to its official proposal.

PROJECT DESCRIPTION AND GOALS

GRAPE is an experimental project of European Clearing House for Open Robotics Development Plus Plus (ECHORD++) and, as hinted in the first part of Chapter 1, it focuses on vineyard farming activities and aims at setting up a robotic manipulation platform able to support lead users to develop a variety of farming applications. In effect, the main goal of GRAPE is not the complete development of an industrial platform, but, coherently with the research scenario, the primary objectives are:

1. the development of example applications in the pre-mentioned context, exploiting and improving the so-called *key enabling technologies*[1] in robot navigation, perception and manipulation of the project partners; the resulting capabilities are likely to allow to make it easier for small and medium enterprises working in the fields of agricultural robots and, more in general, plant protection.

2. increase of robot acceptance by farmers and agronomists: since the very last goal of this project is not solely about pure research but about the enterprise world too, particular attention is given into the realization of a robotic platform that could be accepted by potential end users. For this reason, a constant interaction between the project partners and the potential stakeholders (*i.e.* vinegrowers) is mantained also in development phases.

The example applications have been selected in order to challenge perception and action capabilities, to achieve vineyard monitoring, navigation and manipulation tasks. This decision is taken in the scope of turning traditional farming into precision farming; the realization of such a turn would allow for both the decrease of the chemical load

1 Key Enabling Technologies are a group of six technologies: micro and nanoelectronics, nanotechnology, industrial biotechnology, advanced materials, photonics, and advanced manufacturing technologies. They have applications in multiple industries and help tackle societal challenges; they are consedidered crucial in the creation of advanced and sustainable economies.

(a) (b)

Figure 2.1: *A vine before (2.1a) and after (2.1b) having leafed out.*

in food and environment, and an improvements of profits and yield for farmers, that would get a return for their investment (Herring, 2001). The introduction of precision farming tecniques leads indeed to a lot of advantages, for example early detection of plants diseases, or application of pesticides and fungicides with high precision and only when/where needed.

It's clear that some of these high-precision tasks are still too complex to be automated, and current state-of-the-art in research and technology have been proved to be not yet mature to give rise to a commercial product able to compete with a skilled agricultural agent. The goal of GRAPE is creating the enabling technologies to allow agricultural companies to develop vineyard robots to keep filling the gap that exists with respect to traditional methods.

GRAPE project specification identifies these two application examples:

- **Vineyard monitoring and autonomous navigation**: the developed robotic platform must be able to autonomously navigate the vineyard and localize itself into a map of the environment (downstream a mapping phase), to be able to monitor the state of the vineyard (*e.g.* foliage and grapes inspection). The localization and navigation parts, however, are a key point for any kind of tasks for an UGV field robot, because of the probability of high slope ground, rugged terrain morphology and unstructured map. We'll see the characterization and analysis of the problem of navigation in a rugged outdoor environment in chapter 3.

- **Autonomous application of pherormone dispensers**: pherormone dispensers are used for *mating disruption* tecniques, to protect grapevines from grape moths, by disrupting the bugs' reproductive cycle making use of synthesized sex pherormones. Also in this case, the most relevant challenge is brought by the unstructured environment, that makes the sensing and manipulation tasks significantly harder. However, note that pherormones deployment task gets easier if you think that the timing of the reproductive cycle (on which of course we have no control

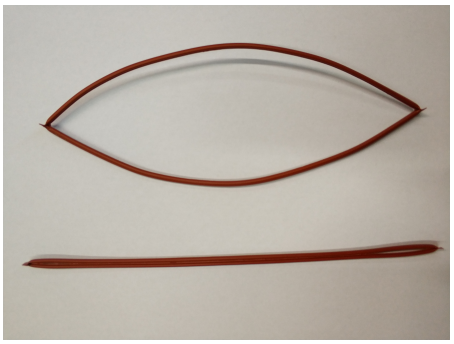


Figure 2.2: *Detail of our dispenser type.*

at all) forces this operation to be performed in a season where grape plants are pruned, and have not leafed out yet (Carde, 1995). Of course, the absence of foliage and grapes makes this task significantly easier (see Figure 2.1).

For what concern the second goal of the project *i.e.* a major commitment about the acceptance of the robotic system in a field that's still strongly led by traditional methods, its realization consists mostly in the creation of an user-friendly interface (possibly for smartphones or tablets) for the GRAPE system, that allows an user to:

- constantly and easily visualize the position and activity of the robot
- perform supervisory control on the platform *e.g.* selection of the plants for the dispensers deployment
- in case of a failure of the UGV in an autonomous task, provide a teleoperation interface to carry out the operation

The whole project was also to be designed with and adequate degree of modularity, in order to naturally support the possible extension to a fleet of robot able to operate in parallel. Note that, as described in Section 1.1, this goal gets very easy by the usage of ROS framework.

The GRAPE projects involved three partners, each with different assignments, and different responsibilities in the project context: **PoliMi**, **Eurecat** (a technology center located in Catalonia, that operates in many industrial and research fields, including robotics), and **Vitirover** (an agricultural robots company, located in France). To make the different teams interact and test the compatibility of the different parts of the system, some communal sessions of integration and tests, that we'll call *integration weeks*, have been carried out within the duration of the whole project. In this thesis we are going to focus mostly on the experiments and results carried out during the last GRAPE integration week, held in March 2018.

(a) (b) (c)

Figure 2.3: *Different shape of pherormone dispensers available on the market. The model used in GRAPE is the one depicted in image 2.3c*

(a) (b)

Figure 2.4: *Other field robots developed in EU projects in the last 6 years: 2.4a Vinerobot, 2.4b Vinbot.*

CONTEXTUALIZATION OF THESIS WORK IN THE PROJECT

Since the timespan of the GRAPE project as a whole covers 18 months, we took advantage of the preceding work described in Serrano et al., 2017, that is a description of the ongoing work around the GRAPE project. As stated in the paper, some results were already accomplished in a few subjects:

- in-depth analysis of the main challenge and obstacles in the project, and requirement extraction
- identification and obtainment of the specific hardware components (mobile platform, robotic arm, LIDARs, IMUs, cameras, network equipment, computational unit)
- segmentation of the macro objectives into smaller subtasks
- results of intermediate experiments carried out during the first integration week in Garriguella, Spain. The experiments mostly concern the mapping, localization and navigation tasks

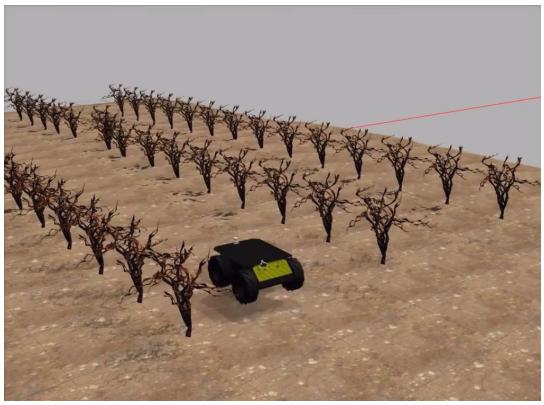


Figure 2.5: *Simulation of the Husky robot in the vineyard environment. Simulation built using Gazebo simulation software.*

Figure 2.6: *The 3D printed vine tree mockup, used in validation phase of the dispenser application task algorithm.*

- creation of a simulated vineyard and Husky environment using Gazebo simulation tool (see Figure 2.5)
- collection of data generated during the same integration week, making use of ROS data logging tool (as described in Chapter 1.1)
- creation of a 3D printed mockup vine tree (see Figure 2.6), to make the laboratory work more efficient and meaningful for the development and validation of the parts that would interact directly with the vine trees (*i.e.* dispenser application task). The mockup was created from the laser scans of a real vine tree in Figueres (ES).

The problems tackled in this thesis are a subset of the ones assigned to PoliMi in the project proposal. The problems we faced are the following, and will be described in depth in next chapters:

- design of the overall software architecture in the ROS framework (see Chapter **??**)

(a) (b)

Figure 2.7: *Other field robots developed in EU projects in the last 6 years: 2.7a Rhea, 2.7b CROPS.*

- revision and enhancement of the mapping, localization and navigation systems (see Chapter 3)
- design and part of the implementation of the pherormone dispenser application (see Chapter **??**)

RELATED PROJECTS

We are now presenting a short list of EU projects related to robotics for precision farming, with direct or indirect application to vineyards. All these projects have been developed in the last 6 years, and demonstrate the growing interest of the scientic community and potential end users to this application and research field.

VINEROBOT Automated measurement and monitoring of parameters such as grape yield, vegetative growth, water status and grape composition in vineyards by use of an UGV (see Figure 2.4a) endowed with artificial intelligence with sensing technologies. Chlorophyll-based fluorescence, RGB machine vision and thermography technologies are used to monitor vineyard on-the-go (Diago, Tardaguila, et al., 2015).

VINBOT Creation of an all-terrain autonomous mobile robot (see Figure 2.4b) with a set of sensors capable of capturing and analysing vineyard images togethere with 3D data, by means of cloud computing applications, to determine the yield of vineyards and to share information with the winegrowers, in order to mix the grapes of homogeneous quality to efficiently market a range of wines by quality and price. (Lopes et al., 2017).

RHEA Design, development, and testing of a new generation of automatic and robotic systems for both chemical, mechanical and thermal effective weed management focused on both agriculture and forestry, and covering a large variety of European products including agriculture wide row crops and forestry woody perennials. RHEA aims at diminishing the use of agricultural

chemical inputs in a 75%, improving crop quality, health and safety for humans, and reducing production costs by means of sustainable crop management using a fleet (see Figure 2.7a) of heterogeneous robots, both ground and aerial, equipped with advanced sensors, enhanced end effectors and improved decision control algorithms (Santos, Ribeiro, and Fernandez-Quitanilla, 2012).

CROPS Design and development of an highly configurable, modular and clever carrier platform that includes modular parallel manipulators and intelligent tools (sensors, algorithms, sprayers, grippers) for high value crops like greenhouse vegetables, fruits in orchards, and grapes for premium wines. The CROPS robotic platform (see Figure 2.7b) is capable of site-specific spraying (targets spray only towards foliage and selective targets), selective harvesting of fruit (detects the fruit, determines its ripeness, moves towards the fruit, grasps it and softly detaches it), reliable detect and classificate obstacles and other objects to enable successful autonomous navigation and operation in plantations and forests (Oberti et al., 2014). Further development in CROPS projects also includes Sweeper project (Ringdahl et al., 2016).

3

LOCALIZATION AND NAVIGATION SYSTEMS IN GRAPE

The studies of **outdoorNavigation** that analyze main issues in autonomous mobile robot navigation, identify three main problems related to outdoor autonomous navigation:

- the unstructured environment where the actions take place, in opposition to the clear and definite one that is typical in indoor navigation (flat floor, right angles, smooth surfaces)
- multiple sensors required, to be aware enough of the surrounding environment, in order to take decisions
- moving obstacles, like for example pedestrians or cars in a city road

The system developed in the context of GRAPE project suffers critically mostly of the first two point listed above. Indeed, the vineyard environment, even if characterized by a high degree of regularity in the global structure, due to the presence of parallel rows of trees, presents strongly difficulties regarding lack of structure in the terrain configuration (steep, bumpy, clay-rich, muddy, according to the weather conditions and the intrinsic composition and morphology of the terrain) and in the obstacles (non-straight surfaces, different reflectivity and lighting conditions). The problem of the fusion of the data coming from the disparates sensors mounted on board of the robot was already addressed from a theoretically point of view in Section 1.4, and it's the major component in the localization system of the robot. Note that, being our robot an agricultural UGV, the problem of moving obstacles is not particularly relevant, since they are most likely to be humans, aware of the robot presence and functionalities, or small wild animals (*e.g.* foxes, hares, cats) in remote cases, that however have a very low probability of approaching the Husky base during movement.

In the next section we are going to analyze with detail the configuration of both the odometry and navigation systems running on the UGV. It will be clear that both the problems, even if eventually solved using *off-the-shelf* solutions, were all but simple problems and required a real understanding of the numerous problems that arose.

ODOMETRY SYSTEM

The configuration of a robust odometry system for vineyard navigation has been a workpoint since the beginning of the project, because

of the expected (and actual) strong instability of the wheel odometry. To the reasons mentioned at the beginning of the Chapter, we recall also the intrinsic lack of precision due to skid steering kinematics of the Husky base. We can describe the search for a robust odometry system for the UGV in three main phases:

1. As hinted in Section 1.4, in the early phases of GRAPE project, the designated sensor fusion framework was **ROAMFREE**, that gave proof of good functioning in the past (see Section 1.4 for precise references). ROAMFREE platform provides multi-sensors pose tracking and it is designed to be flexible and to adapt to every kind of mobile robotic platform. The tracking module is based on Gauss-Newton minimization of the error functions associated to sensor readings. In order to improve generality, physical sensors are abstracted with *logical sensors*, which are characterized only by the type of the proprioceptive measurement they produce. Another ROAMFREE module provides instead self-calibration modules, using error models for each sensor category to provide on-line correction of common sources of distortion, bias and noise (*e.g.* hard and soft magnetic distortion, sensor displacement or misalignment). The usage of ROAMFREE framework was specified in the project proposal, thus it was of course the default choice for the sensor fusion framework in the first *integration week* in Garriguella (ES), before the beginning of this thesis work. The ROS implementation of ROAMFREE is still experimental, anyhow a configuration was identified to fit the requirements. Unfortunately, the same configuration turned out to perform poorly during the second integration week in Casciana Terme (IT). This was caused by concurrence of:
   - presence of a significant slope in the vineyard, in opposition to the flat (even if bumpy) terrain in Garriguella
   - difficulty to handle inaccurate configuration of magnetometer and IMU
   - overall trickiness in the configuration procedure, being ROAMFREE integration with ROS still experimental

   The low performances of this framework in this context were pretty clear, so we opted for a more tested and consolidated solution, **Robot Localization**.

2. The configuration of Robot Localization requires, for each fused sensor, a configuration vector, in which specify which components of the input estimate should be fused into the final pose estimate. Note that the the configuration vector is given in the *frame_id* of the input message: for example, consider a velocity sensor that produces a *geometry_msgs/TwistWithCovarianceStamped*

message with a *frame_id* of *velocity_sensor_frame*. We now assume that the transform would convert X velocity in the *velocity_sensor_frame* to Z velocity in the *base_link_frame*. To include the $\dot{X}$ data from the sensor into the filter, the configuration vector should set the $\dot{X}$ velocity value to true, and not the $\dot{Z}$ velocity value.

In table 3.1 you can see the configuration vectors for the sensors taken as input from Robot Localization. Note that, for example, GPS only gives a contribute about x and y, since the altitude estimation of GPS is often not precise, and it provides no information about the robot orientation. Our configuration of Robot Localization relies on **two** different Extended Kalman Filter (EKF) nodes in cascade:

- a *local* localization node, that fuses continuous data, and estimates the position and orientation of the *base_link* frame of the robot inside the *odom* frame *i.e.* the frame fixed in the initial position of the robot. In other words, the task of the node is to publish *odom* $\rightarrow$ *base_link tf* transform (see Section 1.2 for recall).
- a *global* localization node, that also fuses low frequency sensors *i.e.* GPS, and estimates the position and orientation of *odom* frame inside *map* frame. The *map* $\rightarrow$ *odom* trasfomation "adjusts" the position of frame *odom* in order to contextualize the position of the UGV inside the map. Otherwise, the robot pose would be tracked only with respect to its initial position (origin of *odom* frame), but no positioning inside a map would be provided.

In Figure 3.1 you can see the computation graph of the aforementioned odometry system: you can observe that:

- both nodes write on *tf* topic the respective *tf* transforms
- the local node takes IMU (*/imu/data* topic) and wheels (*/husky_velocity_controller/odom* topic) estimate for the fusion task
- the global node takes as input the pose estimate from GPS (*/odometry_gps* topic), and the pose estimate from the local node (*/odometry/ekf_local_filtered* topic), to implement the cascade scheme.

3. Even if the cascade produced a pretty precise odometry estimate, the odometry computation was shrunk in a single EKF node for two main reasons:
   - as we'll describe in Section 3.2, the navigation system switched to a mapless configuration, so the *tf* transform *map* $\rightarrow$ *odom* became useless

	x	y	z	ψ	θ	ϕ	ẋ	ẏ	ż	ψ̇	θ̇	ϕ̇	θ̇	ẍ	ÿ	z̈
Wheels	✓	✓	✓			✓	✓	✓	✓							
IMU				✓	✓	✓					✓	✓	✓	✓		
GPS points	✓	✓														

Table 3.1: *Configuration vectors for the sensors fused for odometry estimation, using Robot Localization.*

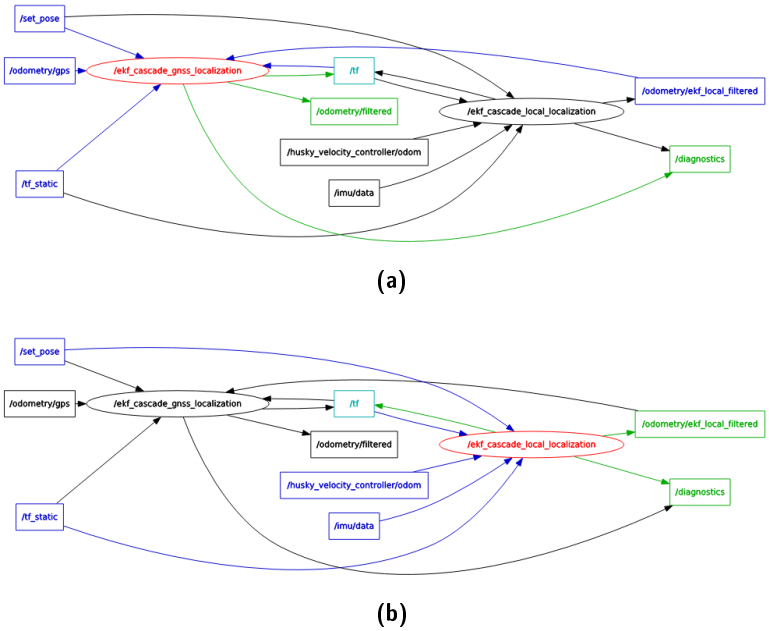


Figure 3.1: *Computation graph of the cascade of Robot Localization nodes; in blue, the input topics, and in green the output topics, of the nodes highlighted in red.*

- the EKFs cascade configuration led to a short delay in the odometry computation (~200-300 ms). This caused some problems with the obstacles detection implemented through LIDAR data, that were processed with negligible delay, and especially when the robot moved from the end of a row of vines to the beginning of another one.

NAVIGATION SYSTEM

We already anticipated in Section 1.5 that the reference navigation framework is **Move Base**, that is an *off-the-shelf* ROS software stack for autonomous navigation (see Figure 1.16 for a graphical representation of the main software modules of Move Base). Since, as already stated, this framework usually provides very stable autonomous navigation in indoor mapped environment, the first attempt was to shift

the same configuration in the vineyard environment. In this Section we'll explain the steps from this initial naive solution to the final accepted one.

Navigation in mapped environment with AMCL

Before the robot could be able to navigate the vineyard, the navigation in mapping environment required a preparatory *mapping* phase; in particular, this is a SLAM problem, that is the computational problem of constructing or updating a map of an unknown environment, while simultaneously keeping track of an agent's (the UGV) location within it. Note that *localization* is a different task with respect to *odometry estimate*, because the first aims to estimate the position of the robot inside a map, while the latter aims to estimate the position of the robot with respect to its initial pose. More technically, in the ROS framework *odometry estimate* algorithms publish the *tf* transform from *odom* frame to *base_link* frame, while *localization* algorithms publish the one from *map* to *odom*.

Even if SLAM is a *chicken-and-egg* problem (localization would be easy in a known map, under some assumptions, and mapping would be easy if the robot position is assumed known), several algorithms exist that solve, even if approximately, the SLAM problem. In the early phases of GRAPE, previous to this thesis work and described in Serrano et al., 2017, three different SLAM algorithms were tested in our specific environment: *Gmapping*, *Google's Cartographer* and *KartoSLAM*, and **Gmapping** algorithm was selected because of its performance and efficiency in terms of computational load. In Figure 3.3 you can see the map created in Mas Llunes during the last integration week, using Gmapping. The accomplished accuracy of the maps was satisfying, but actually a lot of problems occurred in the main phase, that is the autonomous navigation of the robot in the mapped environment. The most relevant problems that emerged were the following ones.

AMCL IN THE VINEYARD MAP

As you can see in Figure 1.16, an optional node called **amcl** exists, that provides a ROS implementation of AMCL algorithm (**amcl**). AMCL is a localization algorithm, that uses a particle filter to provide a *belief i.e.* the robot's estimate of its current state, through a probability density function distributed over the state space. A single particle represents an hypothesis about the real pose of the robot in the environment (see Figure 3.4); thus, regions in the state space with many particles correspond to a greater probability that the robot will be there, and regions with few particles are unlikely to be where the robot is.

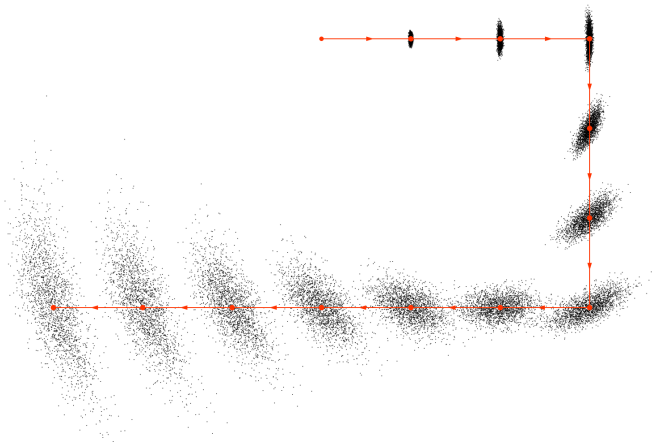


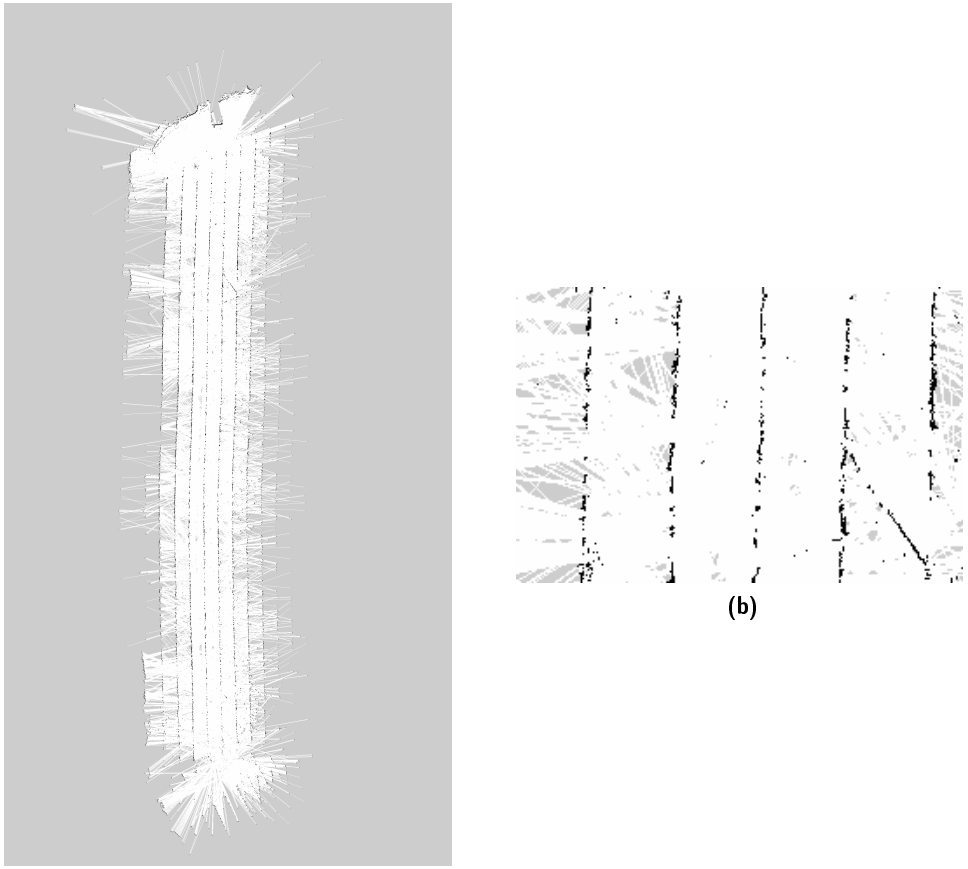
Figure 3.2: *Evolution of the belief computed by AMCL, taking into account the motion update only (ignoring the sensor update).*

In AMCL each particle is assigned a weight that correspond to the probability that, had the robot been at the state of the particle, it would perceive what its sensors have actually sensed (in particular, *amcl* ROS package is only capable to deal with laser scans as sensor input); this weight is used in the algorithm to select the particles that are going to survive through iterations. For this reason, the algorithm requires both a sensor model of the used sensors, and a motion model of the involved robot, in order to update the *belief*. In Figure 3.2 you can see several steps of *belief* update, only based on the motion model of the robot. Pseudo code of Monte Carlo Localization is provided in Algorithm 2.

Algorithm 2 Monte Carlo Localization(X_{t-1}, u_t, z_t)

1: $X_t = \emptyset$ ▷ X_t is belief about the robot state
2: $\overline{X}_t = \emptyset$
3: for $m = 1$ to M: ▷ M is number of particles
4: $x_t^{[m]} = \text{motion\_update}(u_t, x_{t-1}^{[m]})$ ▷ u_z is actuation command
5: $w_t^{[m]} = \text{sensor\_update}(z_t, x_t^{[m]})$ ▷ z_t are sensed data
6: $\overline{X}_t = X_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$
7: endfor
8: for $m = 1$ to M:
9: draw $x_t^{[m]}$ from $\overline{X}_t$ with probability $\propto w_t^{[m]}$
10: $X_t = X_t + x_t^{[m]}$
11: endfor
12: return X_t

AMCL with laser scanners provides a very effective localization in feature-rich environments, like in typical indoor environments with very sharp corners and well defined obstacles, as the ones



(a) (b)

Figure 3.3: *Map of the Mas Llunes vineyard in Garriguella (ES), computed during the last integration week using Robot Localization and Gmapping. In 3.3b, a magnified detail.*

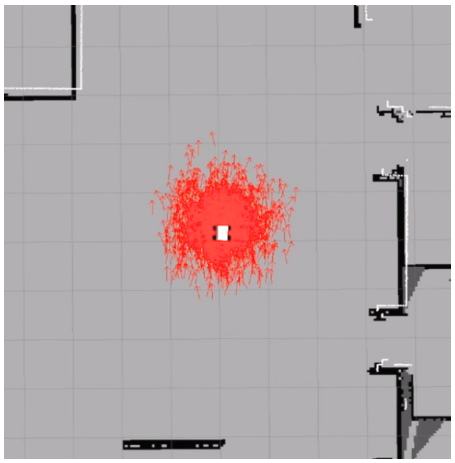


Figure 3.4: *Belief about the robot estimate using AMCL, through a particles cloud where each particle represents a hypothesis about the real pose.*

in Figure 3.4. However, as hinted at the beginning of this Section, a few tests were sufficient to show the weakness of this localization algorithm in our context, due to the high repetitiveness of the vineyard map. In Figure 3.3b you can see a magnified detail of the map of Mas Llunes vineyard, computed using Robot Localization as odometry system, and Gmapping for SLAM; most of the obstacles detected by the frontal LIDAR are the trunks of the trees, together with poles and high weeds. It's intuitive that if the map in which the robot moves presents a lot of repeated patterns, the *survival of the fittest* paradigm that the *sensor update* step of AMCL should implement is not very effective. Moreover, note that the bumpiness of the terrain leads the plan that the LIDAR scans to change significantly its orientation in time. So, for example, given a position in the map, the sensed obstacles over time in this location could be different portions of the same vine, then the shape is likely not to be constant.

ROW CROSSING PROBLEM

The vineyard environment also presented another criticality, due to the average distance between the vines belonging to the same row; indeed, because of the height of the LIDAR used for obstacles detection, the trunks of the plants are detected while the horizontal branches of the vines often are not. If this situation occurs, the space between the trees is sensed as free of obstacles, even if the horizontal branches of the vines are an actual obstacle to the UGV. The first attempt to solve this problem was to simply provide an higher inflation radius (see Section 1.6) to the obstacles, in order to fill the gaps between a plant and the adjacent ones. However, this naive solution couldn't find a trade-off to the extreme situations:

- a too small inflation radius is not able to provide a solution to the problems *i.e.* fill the gaps between adjacent trees
- an inflation radius large enough to fill the aforementioned gaps, makes the navigation almost impossible, because even a small obstacle among two rows (*e.g.* some weed) completely occludes the path.

Insertion of prohibition layer

Given the impossibility to solve the problem of row crossing with the standard inflation radius of the obstacle layer, we found a very effective solution that relies on the very simple global structure of the vineyard. Indeed, looking at the map in Figure 3.3a, it's easy to notice that the obstacles that cause the row crossing problem are of course aligned, and parallel to each other, so the idea was to implement a series of "virtual fences" corresponding to the vine rows. The imple-

mentation was possible thanks to a feature of *costmap_2d*, the package of ROS responsible of the creation of the occupancy grid maps. Indeed, this package implements a **layered** costmap: this means that, for example, the global and local costmaps (see Section 1.6) live in two different layers, that are computed independently:

- the global costmap is computed from the obstacles described in the occupancy grid map, possibly inflated of a chosen radius
- the local costmap is computed in a spacial window around the robot position, from the obstacles detected by the laser scans. The obstacles possibly are inflated of a chosen radius

Since an arbitrary number of custom layers can be added by implementing C++ base class *costmap_2d::Layer* of Move Base package, two additional layers were introduced, in order to include the virtual obstacles of the rows both in global and local costmap:

- **Global prohibition layer:** this includes the virtual fences in the global costmap, and it's useful for the global planner to take them into account for the creation of the global navigation plan
- **Local prohibition layer:** this includes the virtual fences in the local costmap, and it's useful for the local planner to take them into account for the creation of the local navigation plan. This is required because otherwise the local planner might over-optimize the global plan and, however take advantages of the gap between the plants. This situation is comparable to an indoor planning problem in which a local planner detects an open door that was shut at mapping time: even if the global path doesn't take advantage of the door gap, the local planner might choose to traverse it if it reduces the motion execution cost.

Switch to mapless navigation

After several effort in Move Base parameter tuning, the role of AMCL in the poor performance in navigation tasks was clear. Since the map in navigation task is useful in terms of:

- **global planning**: it provides a set of obstacles that helps the robot to optimize the global path in planning phase. See for example Figure 3.5: if no map was provided, the global plan would have been a straight line between the start and the goal states, and all the obstacle avoidance lies on local planner. This of course can lead to a performance degradation, because the knowledge of some obstacles in advance often brings to much more efficient global plans.
- **localization**: in order to provide an estimate of the robot pose in the environment (for example, through AMCL)

- **visualization**: to better contextualize the position of the robot for human users.

Given the problems caused by AMCL, we decided to switch to a **mapless** navigation configuration, because we were able to compensate all the advantages of the mapped navigation using other tools or methods:

- **global planning**: as we already pointed out in Subsection 3.2.2, even if the vineyard environment is strongly unstructured at fine-grained level (for the reasons explained at the beginning of this Section), it has a fixed and very simple global structure given by the parallel vine rows, as you can see in vineyard maps (Figure 3.3). Thus, the inclusion of the virtual fences layers only in the global costmap is a simple, but very effective approximation of the vineyard global map.
- **localization**: in both our field test locations (Casciana Terme (IT) and Garriguella (ES)), we could rely on high-precision RTK GPS (**rtk**) as input for the UGV odometry system. Even if RTK correction worldwide coverage is not guaranteed, several companies exist that provide this service. For example, see Figure 3.6 for a map of RTK stations provided in Italy by the sole SmartNet Italy[1] company. Moreover, since our navigation tasks are executed outdoor, the assumption of an high-precision GPS system is reasonable and we only need to specify the virtual fences positions through the GPS coordinates of their endpoints to navigate the virtual fences costmap layer.
- **visualization**: in this configuration our UGV is always geolocalized; so, we can contextualize the robot position for human users by overlaying to the robot position the corresponding satellite images, using for example the *AerialMapDisplay* plugin for RViZ.

Since we were able to compensate for all the advantages of the navigation with a map, we decided to switch to a mapless configuration, at the cost of adding the assumption of high-precision GPS and geolocalized virtual fences at disposal. Note that, in this configuration, the *localization* task is not relevant anymore since there is not an actual map to localize into; then the *tf* transform from *map* to *odom* is reduced to the identity transformation.

Switch to Timed Elastic Band as local planner

Also the standard local planner of Move Base, a ROS implementation of Dynamic Window Approach (**dwa**), showed some weak points in

1 http://it.smartnet-eu.com/

Figure 3.5: *Example of global plan that takes into account the map obstacles.*

Figure 3.6: *RTK stations provided by SmartNet Italy.*

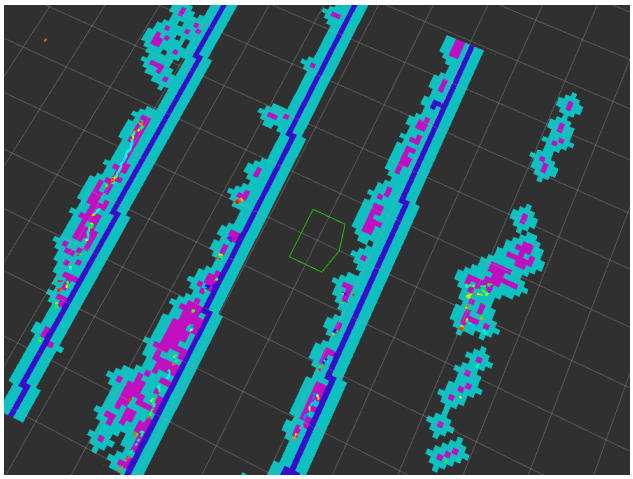


Figure 3.7: *Example of local costmap, in the final configuration: you can distinguish the virtual fences (blue), the obstacles detected by the LIDAR (pink), and the inflation layer (light blue). The green polygon represents the robot footprint.*

the vineyard context. In Figure 3.7 you can see an example of local costmap, computed during an autonomous navigation tasks in Garriguella: the virtual fences are highlighted in blue, the obstacles detected by the frontal LIDAR in pink, and the inflation layer in light blue. Since the detected obstacles are mainly tree trunks, they are detected as small, non continuous obstacles. The inflation of such obstacles leads to a jagged border of the resulting obstacle cells set. If, for example, the detected obstacle was a straight wall, the resulting obstacle layer would have been much more straight and sharp. Dynamic Window Approach had some problems to deal with such a large amount of dead ends, and often got stuck in the obstacle contours (see Figure 3.8). Thus, we switched to another *off-the-shelf* local planner, compatible with the Move Base local planner interface: an implementation for ROS of the *Timed Elastic Band* algorithm (**teb**). This algorithm, at the cost of an higher computational load, provided much more robust navigation.

AMCL as input to sensor fusion

In the late phases of the GRAPE project, some efforts were made to, however, take advantage in some way of AMCL algorithm potential. In particular, the alignment with the sensed laser scans could lead to the following benefits:

- better estimate of the robot orientation
- compensate for the measurement errors of the GPS, since the virtual fences are specified through GPS coordinates of the endpoints

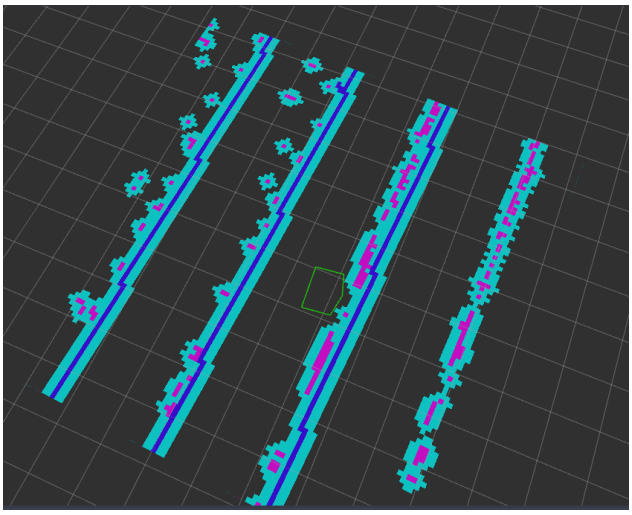


Figure 3.8: *Example of local plan computed by DWA that get stuck in the jagged border of the obstacles.*

We already highlighted the strong drawbacks due to using AMCL as localization algorithm, so the attempt was to disable its publication of the *tf* transform from *map* to *odom*, and fuse its estimated pose together with the other sensor inputs in the Robot Localization node. In this way the weaknesses of AMCL alone are likely to be compensated from the other sensors' estimates. If this configuration had resulted effective, switching back to a navigation with map would have been a reasonable choice. Unfortunately, due to the lack of time (final review of GRAPE project was held in March, 21st, 2018), this configuration was not sufficiently tested.

However, the tests that were performed were sufficient to highlight some difficulties in the coexistence of GPS and AMCL as input sources in the Robot Localization nodes, probably for these reasons:

- Low publishing rate of both AMCL and GPS (~1 second)
- Pose estimates always discordant between them, maybe for some misconfiguration or bias of GPS module

For this reason, the output pose of Robot Localization suffered of constant jumps between the estimate of GPS and the one of AMCL, instead of keeping both of them into account. In Figure 3.9 an example is provided of discordance between the estimate from AMCL (represented by the particles position) and the one from GPS (corresponding in the figure to the actual estimated pose).

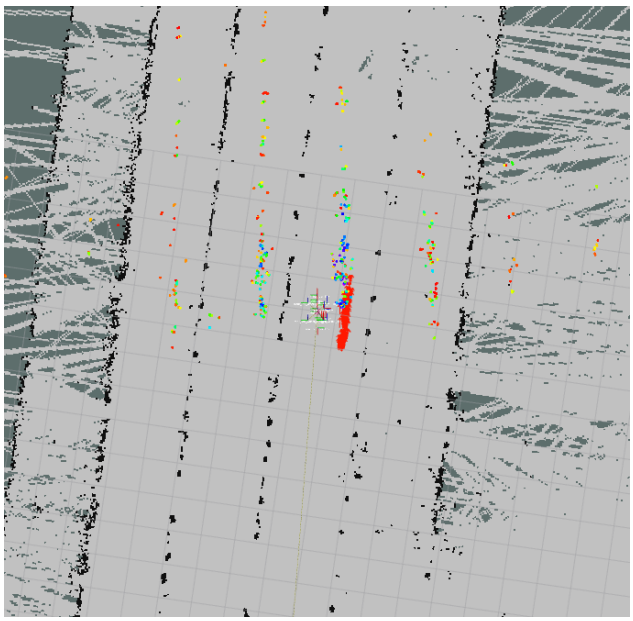


Figure 3.9: *Example of conflict between the pose estimated from AMCL (displayed by the particles positions), and the final Robot Localization output (displayed by tf frames).*

4

EXPERIMENTAL RESULTS

On the field sessions were a crucial point in the validation of GRAPE system, for two main reasons:

1. Robotics is a field that, by definition, requires interaction with the physical world, both the components of the robot itself, and the environment. Thus, simulation could strongly speed up the the development in certain phases, but the creation of a robotic system cannot opt out of experimental sessions. This consideration is especially true in agricultural robotics field, because a simulated environment or a laboratory environment are extremely likely to be too different from the actual robot environment to be significant.

2. As explained in Section 2.1, the research team included members from both Politecnico di Milano and Eurecat, that are located in two different countries. Even if this should not have been a big problem, the division of labor created significant inefficiencies because:

   - even if the sensor fusion and navigation parts were assigned to Politecnico, the Husky UGV was property of Eurecat, so it was in Barcellona
   - even if the analysis of the vine scan in order to detect the most suitable deployment point was in charge of Eurecat, the arm was property of Politecnico so it was in Milano.

   This problem was mitigated by the logging tool of ROS (*rosbag* package we mentioned in Section 1.1), that demonstrated its potential in such a situation: *bags* recorded during (non-autonomous) navigation sessions of the Husky were very useful to test the sensor fusion system, while *bags* of the laser scans sensed during execution of the scan motion in front of the mockup vine (see Figure 2.6), owned by Politecnico team. However, the usage of *bags* in our context showed two main weaknesses, the first of which is is specific of our context, while the second is intrinsic to the use of recorded data:

   - autonomous navigation algorithms cannot be tested by means of recorded data *bags* because, during an autonomous navigation session, the data that are sensed while using a specific navigation algorithm depend on the choices of the algorithm itself; the same reasoning does not apply to neither sensor fusion nor point cloud analysis tasks. For this

reason, the only way to test a navigation algorithm except for *on-the-field* experiments is via simulation by means of a physical simulator. Some effort were spent in this direction before the beginning of this thesis work (see Section 2.2), but very few *on-the-field* tests have been enough to show that the simulated environment was way too different from the real one for the test to be significant.

- validation of algorithms by means of recorded data significantly increases the risk of overfitting *i.e.* adapting the model only to perform well with the data in the (few) *bags* at disposal, and not new new data coming, for example, from an *on-the-field* experiment. The best ways to decrease the probability of overfitting are:
  - use as many bags as possible in testing and validation phase
  - test in the real environment as often as you can. Remember, however, that sometimes overfitting can occur also in real environment, in less obvious way. For example, the ROAMFREE configuration executed in Mas Llunes during the first integration week was absolutely ineffective when applied in Casciano Terme vineyard, because it fitted too tightly the features of Mas Llunes vineyard thus it didn't represent a general solution.

SENSOR FUSION

NAVIGATION

MANIPULATION

(a)

(b)

Figure 4.1: *Pose estimation in time using raw wheels odometry (Figure 4.2), and using GPS (Figure 4.1b).*

Figure 4.2: *The odometry estimated by Robot Localization sensor fusion framework, fusing (among others) the pose estimated by the wheels (see Figure 4.2) and by the GPS (see Figure 4.1b)*

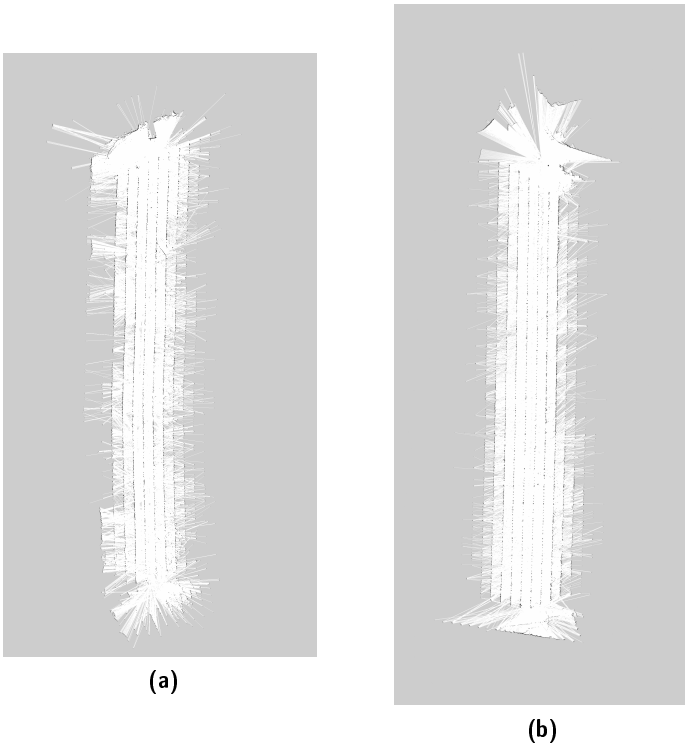


Figure 4.3: *Output of mapping phase, using Gmapping as SLAM algorithm and Robot Localization as sensor fusion framework.*

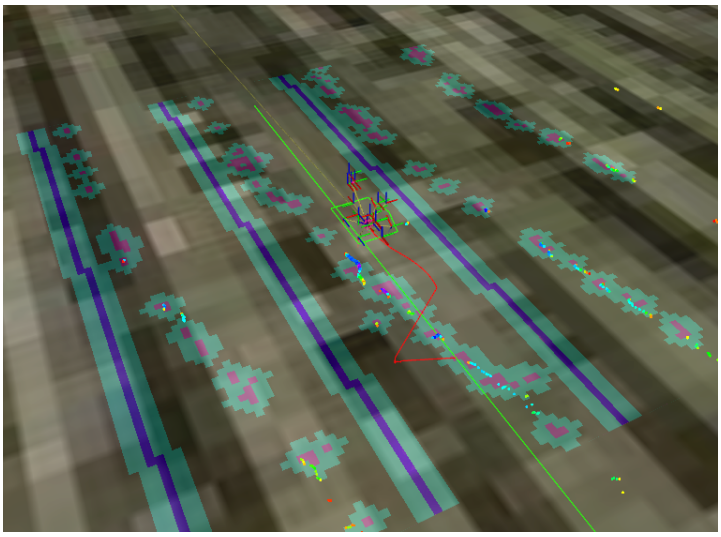


Figure 4.4: *Visualization through RViZ of the Husky performing autonomous navigation, with the local costmap overlaid to the satellite images of Mas Llunes vineyard. You can see the global (green line) and local (red line) navigation plans, the* tf *tree of the robot and the Husky footprint used by Move Base for the interaction with the costmap.*

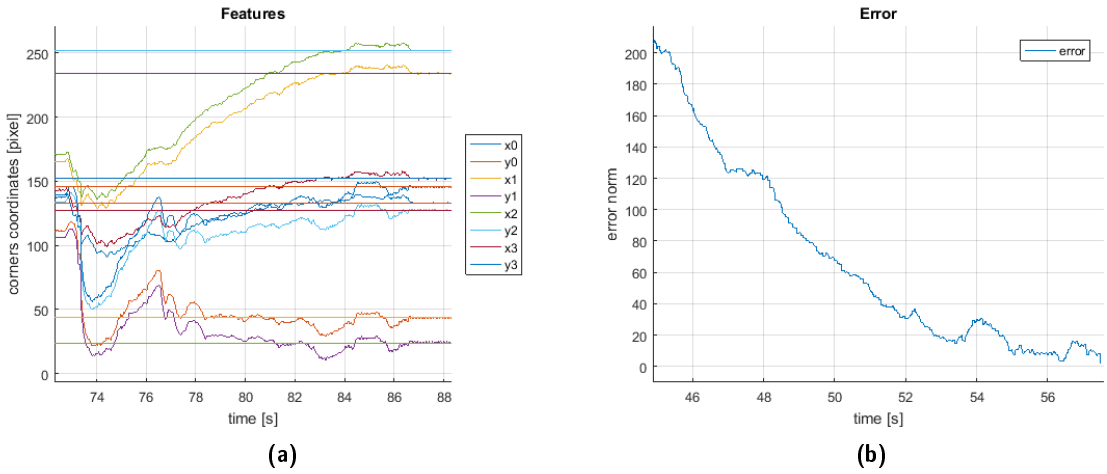
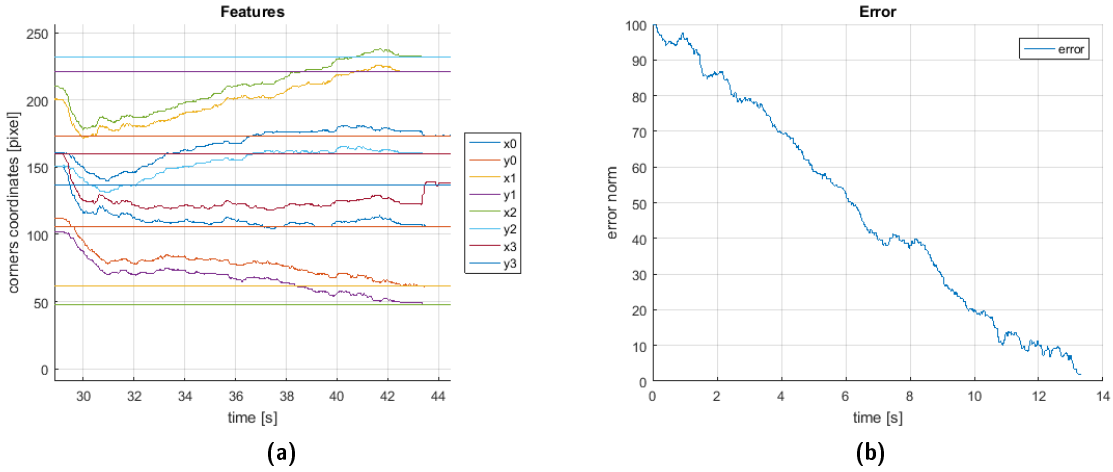


Figure 4.5: *Evolution in time of the image coordinates of the corners of the tracked marker (see Figure 4.5a) during visual servoing driven deployment of a dispenser on a nail. In Figure 4.5b, the evolution in time of the module of the error on the tracked features.*



(a) (b)

Figure 4.6: *Evolution in time of the image coordinates of the corners of the tracked marker (Figure 4.6a) during visual servoing driven grasping of the dispenser from the feeder. In Figure 4.6b, the evolution in time of the module of the error on the tracked features,*

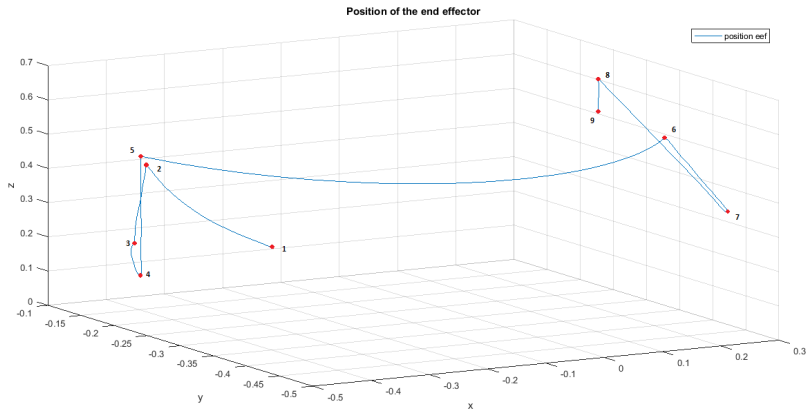


Figure 4.7: *Evolution of the 3D (x, y, z) position of the end effector of the Jaco^2 during dispenser deployment using MoveIt!*

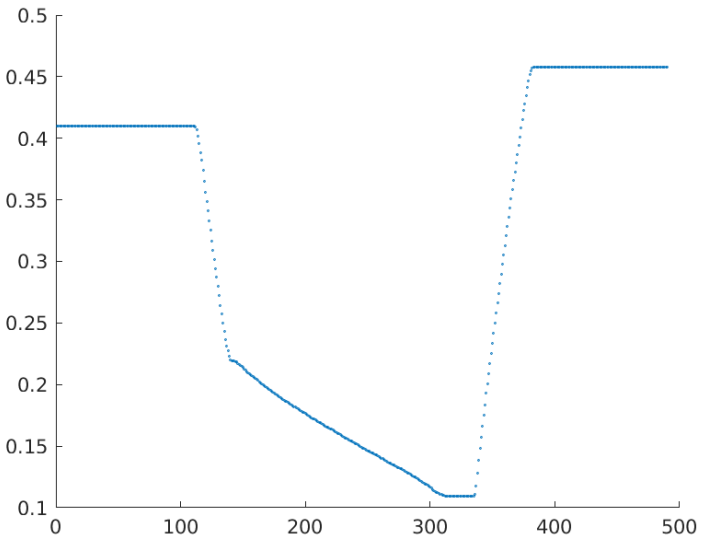


Figure 4.8: *Evolution of the z coordinate of the end effector, during visual servo driven grasping of the dispenser from the feeder.*

5 THE ROBÌ PROJECT

A branch of this thesis work involved the customization of the prototype mobile manipulator Robì (Bascetta, Baur, and Gruosso, 2017) to fit the requirements of the mobile base requested by the GRAPE project.

ROBÌ MOBILE MANIPULATOR

Robì (see Figure 5.1) is a prototype small-sized mobile manipulator for agricultural applications, whose aim is to support the development and testing of innovative perception and control algorithms. Both the mechanical structure and the motion control system are designed to be simple, flexible, low-cost and low-weight, to create a system that can act as an open source base for project in agricultural robotics, contrary to the large amount of task-specific platforms in agricultural robotics (*e.g.* for asparagus (Chatzimichali, Georgilas, and Tourassis, 2009) or tomato (Yaguchi et al., 2016) harvesting).

Since the platform is thought as a versatile mobile base to be adapted to several fields (*e.g.* vineyards, herbaceous plants) its chassis is designed to have flexible geometrical characteristics (ground clearance, track, wheelbase).

More generally, the design principles of Robì are:

- **low cost**, to make it easier to use it for example applications and, possibly, fleet-based applications
- **low weight**, to increase the battery life and, consequently, allow for more long and versatile missions

Figure 5.1: *The Robì mobile manipulator chassis, in one of its*

Ground clearance	[0.25, 0.35] m
Track	[0.75, 1.2] m
Wheelbase	[0.6, 1] m

Table 5.1: *Ranges of Robì geometrical characteristics.*

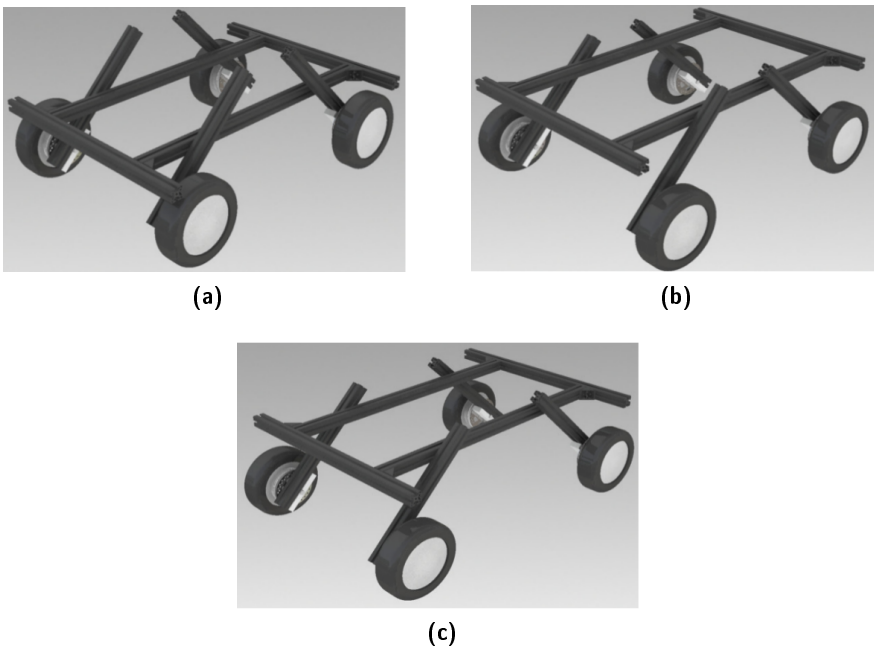


Figure 5.2: *3D rendering of different configuration of Robì base, obtained thanks to the chassis mechanical design.*

- **simple mechanical design**, to make it easy to build it out of a mounting kit

The aforementioned principles are implemented through the following design choices:

CHASSIS

The chassis is made out of ITEM® aluminium bars, that presents lightweight but strong section; the slide rails embedded in the ITEM® bars, together with four rotational joints applied to the bars supporting the wheels, allow for multiple configuration of the robot geometrical characteristics to vary in the ranges listed in Table 5.1. In Figure 5.2 you can see 3D rendering of some of the available Robì configurations. The bars that support the wheel, however, can't rotate during the robot movement, thus Robì has a skid steering kinematic model.

IN-WHEEL MOTORS

Robì is powered by four in-wheel electric DC brushless motors; the model is HUB10GL (see Figure 5.3). In-wheel motors have

Figure 5.3: *HUB10GL, the in-wheel motor model adopted in Robì.*

been chosen mostly because they allow for much simpler mechanical design, with respect to classical electrical powertrain, because:

- no transmission is required
- the weight is reduced, and the available space on the chassis increases
- high level of maneuverability, without the need to introduce Ackermann kinematics

The motors are independent, so a suitable control board is required to control each of them; the board used in Robì are the ones described in VESC project[1], an open source brushless motor control project. These boards offers several interfaces to control the motors: PPM signal, analog, UART, I^2C, USB or CAN-bus.

ROBÌ AS GRAPE ROBOTIC BASE

The features mentioned in the previous section made Robì a convincing candidate as the mobile platform for the GRAPE project, because of its flexibility as a mobile manipulator for agricultural robotics field. Unfortunately, for mere lack of time, the late phases the project focused on the development of the whole system on a commercial robot, the Husky that was already described as the final UGV of the GRAPE project.

However, significant steps in the adaptation of Robì as a base for a real agricultural robotic project were made, thus we are explaining them in this section.

1 http://vedder.se/2015/01/vesc-open-source-esc/

(a) (b) (c)

Figure 5.4: *Some of the sensors Robì is equipped with: LMS291 (5.4a), and LD-MRS400001 (5.4b), both laser scanners from SICK; Trimble 5700 GPS transceiver (5.4c), from Trimble*

Hardware configuration

Since our goal was to create the system that was later developed on the Husky robot, the sensors mounted on it were similar; on the other side, we had to dedicate significant efforts to the creation of an hardware interface between the on-board computer (a commercial HP laptop) and the motors control boards. Given the flexibility of Robì physical structure, we decided to add a simple structure in the frontal part of Robì as a support for the bulkier sensors (see Figure 5.6), built out of ITEM® bars for compatibility. Note that the Jaco^2 was still the designated arm in this configuration (see Figure 5.6b).

LIDARS

In section 5.1 we pointed out that Robì, as the Husky robot, comes up with a skid steering kinematic model. Thus, we chose to use two frontal LIDARs for obstacles avoidance during navigation tasks. The LIDAR models were:

- **Sick LMS291**: this sensor[2] only has a single measuring plane, maintained parallel to the ground, in order to detect long-range obstacles during navigation
- **Sick LD-MRS400001**: this sensor[3] is a multi-range laser scanner, with 4 measuring planes with narrow aperture angles. Unlike *LMS291* scanner, its arrangement in the sensors support bar is slightly tilted forward, to be able to scan closer obstacles.

2 https://www.sick.com/ag/en/detection-and-ranging-solutions/2d-lidar-sensors/lms2xx/lms291-s05/p/p109849

3 https://www.sick.com/de/en/detection-and-ranging-solutions/3d-lidar-sensors/ld-mrs/ld-mrs400001/p/p112355

	LMS291	LD-MRS400001
Number of Channels	1	4
Scan Angle	180°	85°
Rotation rate	75Hz	12.5 Hz
Angular Resolution	0.25°	0.125°
Range	80m	300m
Power Consumption	30W	8W
Weight	4.5kg	1kg

Table 5.2: Comparison of Robì on-board LIDARs

GPS

We opted for a very reliable solution, the **Trimbe 5700** GPS receiver (see Figure 5.4c) from Trimble, together with its dedicated antenna.

RADIO

In order to easily control and monitor the activity of the UGV especially during outdoor tasks, we also installed on Robì the required hardware to create a *point-to-point* radio bridge with another base station (*e.g.* another laptop in a more comfortable place). Two base stations were chosen, which performance were measured in urban environment (because, as already stated, the development of Robì as GRAPE robotic base was aborted before *on-the-field* validation):

- **Rocket M5** station[4] from Ubiquiti, together with an omnidirectional antenna, mounted directly on Robì
- **NanoStation M5** station[5] from Ubiquiti, to be connected to the external device.

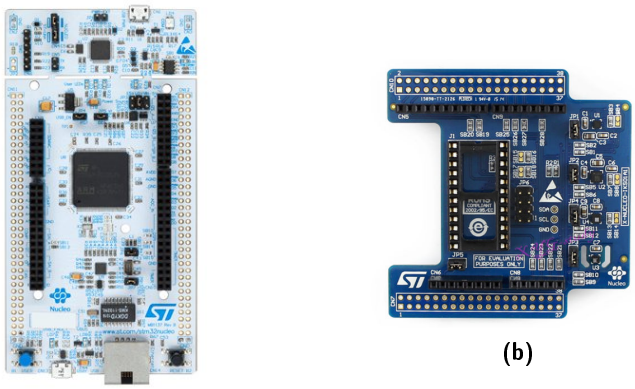
This couple of devices were able to durably create a radio link of ~340 meters.

MOTORS INTERFACE

In order to communicate with the motors control boards, we had to introduce a single hardware interface among them and the on-board PC. We decided to use **NUCLEO-F746ZG** development board[6] from ST (see Figure 5.5a). It provides:

- RJ-45 Ethernet interface, very useful to communicate with the on-board PC

4 https://dl.ubnt.com/datasheets/rocketm/RocketM_DS.pdf
5 https://dl.ubnt.com/datasheets/nanostationm/nsm_ds_web.pdf
6 http://www.st.com/en/evaluation-tools/nucleo-f746zg.html



(a) (b)

Figure 5.5: *The Nucleo board we used to communicate with the motors control board (Figure 5.5a), and the shield mounted on it to integrate IMU sensor (see Figure 5.5b).*

- CAN protocol interface, to communicate with the motors control boards
- ST Zio connector, which extends the Arduino Uno V3 connectivity and provides an easy means of expanding the functionality of the Nucleo platform with a wide choice of specialized shields

In particular, the interface with the on-board PC was implemented as a bidirectional UDP connection. This will be cleared out in Section 5.2.2.

IMU

For the choice of IMU, we decided to take advantage of the Zio interface of the Nucleo board, so we inserted in the system the **X-NUCLEO-IKS01A1** shield[7] (see Figure 5.5b). This shield includes, among others:

- LSM6DSo MEMS 3D accelerometer and 3D gyroscope
- LIS3MDL MEMS 3D magnetometer

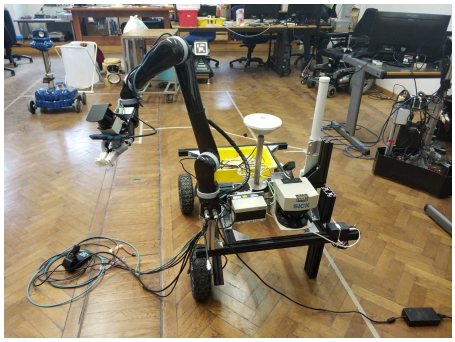
Software architecture

We couldn't test any business logic on Robì platform, again because it didn't reach the main phases of GRAPE project as the chosen platform. However, a possible platform switch would take great advantage from ROS intrinsic modularity. In the ROS computational graph, the only components influenced by the underlying hardware are:

- the topic where to publish the target linear and angular velocities for the UGV (typically, *cmd_vel* topic)

7 http://www.st.com/en/ecosystems/x-nucleo-iks01a1.html

(a)



(b)

Figure 5.6: *The Robì platform, equipped with the sensors required by GRAPE project.*

- the topic where to read the odometry estimated only from the wheel movement (in the Husky, */husky_velocity_controller/odom* topic)
- the topics where to read the disparate sensors measurements
- *tf* topic, where the complete *tf* tree of the UGV is published

For these reason, we are quite confident that, excluding the unavoidable changes at least in Move Base configuration, the platform switch *should* not be too challenging.

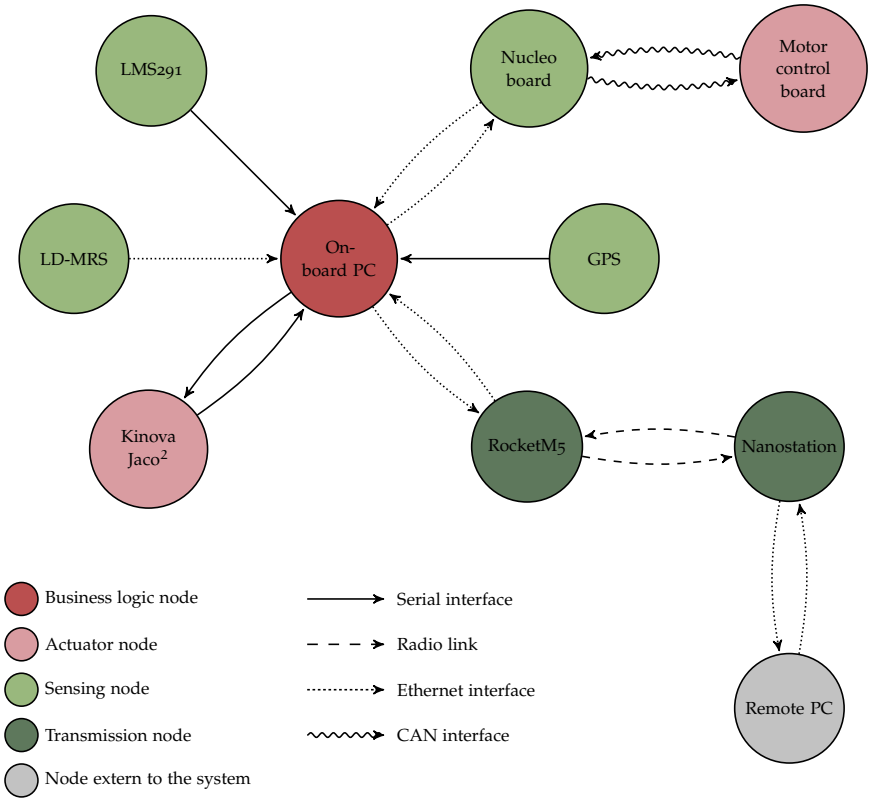
Anyway, we are going to show the low-level software architecture of the system, by means of a graph where:

- the nodes are hardware components, that generate or process data
- the (oriented) edges represent the data flow, from the node that generates data (tail) to the node that process them (head); different arrow types represent different communication strategies

Note that:

- actuator nodes both receive and generate data, because they both accept commands and generate feedbacks

- Nucleo board both provides data from IMU shield and wheels feedback (outgoing arrow), and accepts velocity commands for the motors (ingoing arrow). Both the communication are implemented via UDP server.



BIBLIOGRAPHY

Bascetta, Luca, Marco Baur, and Giambattista Gruosso (2017). "ROBI': A Prototype Mobile Manipulator for Agricultural Applications." In: *Electronics* 6.2, p. 39 (cit. on p. 53).

Calabrese, Luca (2014). "Robust odometry, localization and autonomous navigation on a robotic wheelchair." In: (cit. on p. 15).

Carde Ring T Minks, Albert K (1995). "Control of moth pests by mating disruption: successes and constraints." In: *Annual review of entomology* 40.1, pp. 559–585 (cit. on p. 27).

Chatzimichali, Anna P, Ioannis P Georgilas, and Vassilios D Tourassis (2009). "Design of an advanced prototype robot for white asparagus harvesting." In: *Advanced Intelligent Mechatronics, 2009. AIM 2009. IEEE/ASME International Conference on*. IEEE, pp. 887–892 (cit. on p. 53).

Colledanchise, Michele and Petter Ögren (2017). "Behavior Trees in Robotics and AI, an Introduction." In: *arXiv preprint arXiv:1709.00084*.

Cucci, Davide A and Matteo Matteucci (2014). "Position tracking and sensors self-calibration in autonomous mobile robots by gauss-newton optimization." In: *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE, pp. 1269–1275 (cit. on p. 15).

Cucci, Davide Antonio and Matteo Matteucci (2013). "A Flexible Framework for Mobile Robot Pose Estimation and Multi-Sensor Self-Calibration." In: *ICINCO (2)*, pp. 361–368 (cit. on p. 15).

Diago, Maria P, Javier Tardaguila, et al. (2015). "A new robot for vineyard monitoring." In: *Wine & Viticulture Journal* 30.3, p. 38 (cit. on p. 30).

Dudek, Wojciech, Wojciech Szynkiewicz, and Tomasz Winiarski (2016). "Nao robot navigation system structure development in an agent-based architecture of the RAPP platform." In: *Challenges in Automation, Robotics and Measurement Techniques*. Springer, pp. 623–633 (cit. on p. 15).

Elmenreich, Wilfried (2002). "Sensor fusion in time-triggered systems." In: (cit. on p. 13).

Foote, Tully (2013). "tf: The transform library." In: *Technologies for Practical Robot Applications (TePRA), 2013 IEEE International Conference on*. Open-Source Software workshop, pp. 1–6 (cit. on p. 5).

Fox, Dieter, Wolfram Burgard, Frank Dellaert, and Sebastian Thrun (1999). "Monte carlo localization: Efficient position estimation for mobile robots." In: *AAAI/IAAI* 1999.343-349, pp. 2–2 (cit. on p. 18).

Gage, Douglas W (1995). *UGV history 101: A brief history of Unmanned Ground Vehicle (UGV) development efforts*. Tech. rep. NAVAL COM-

MAND CONTROL, OCEAN SURVEILLANCE CENTER RDT, and E DIV SAN DIEGO CA.

Hart, Peter E, Nils J Nilsson, and Bertram Raphael (1968). "A formal basis for the heuristic determination of minimum cost paths." In: *IEEE transactions on Systems Science and Cybernetics* 4.2, pp. 100–107 (cit. on p. 20).

Herring, David (2001). "Precision farming: Feature articles." In: (cit. on p. 26).

Hinkel, Georg, Henning Groenda, Lorenzo Vannucci, Oliver Denninger, Nino Cauli, and Stefan Ulbrich (2015). "A domain-specific language (DSL) for integrating neuronal networks in robot control." In: *Proceedings of the 2015 Joint MORSE/VAO Workshop on Model-Driven Robot Software Engineering and View-based Software-Engineering*. ACM, pp. 9–15.

Hutchinson, Seth, Gregory D Hager, and Peter I Corke (1996). "A tutorial on visual servo control." In: *IEEE transactions on robotics and automation* 12.5, pp. 651–670.

Latombe, Jean-Claude (2012). *Robot motion planning*. Vol. 124. Springer Science & Business Media (cit. on p. 18).

LaValle, Steven M (1998). "Rapidly-exploring random trees: A new tool for path planning." In: (cit. on p. 22).

Lopes, Carlos M, Albert Torres, Robert Guzman, João Graça, Miguel Reyes, Gonçalo Vitorino, Ricardo Braga, Ana Monteiro, and André Barriguinha (2017). "Using an unmanned ground vehicle to scout vineyards for non-intrusive estimation of canopy features and grape yield." In: *GiESCO International Meeting, 20th, Sustainable viticulture and wine making in climate chenge scenarios, 5-10 November 2017*. GiESCO (cit. on p. 30).

Lozano-Perez, Tomas (1983). "Spatial planning: A configuration space approach." In: *IEEE transactions on computers* 2, pp. 108–120 (cit. on p. 19).

Madhavan, Raj, Lino Marques, Edson Prestes, Renan Maffei, Vitor Jorge, Baptiste Gil, Sedat Dogru, Gonçalo Cabrita, Renata Neuland, and Prithviraj Dasgupta (2015). *2015 humanitarian robotics and automation technology challenge*.

Marder-Eppstein, Eitan, Eric Berger, Tully Foote, Brian Gerkey, and Kurt Konolige (2010). "The Office Marathon: Robust Navigation in an Indoor Office Environment." In: *International Conference on Robotics and Automation* (cit. on p. 17).

Mohanty, Vikram, Shubh Agrawal, Shaswat Datta, Arna Ghosh, Vishnu Dutt Sharma, and Debashish Chakravarty (2016). "DeepVO: a deep learning approach for monocular visual odometry." In: *arXiv preprint arXiv:1611.06069* (cit. on p. 15).

Moore, T. and D. Stouch (2014). "A Generalized Extended Kalman Filter Implementation for the Robot Operating System." In: *Proceed-*

ings of the 13th International Conference on Intelligent Autonomous Systems (IAS-13). Springer (cit. on p. 15).

Oberti, Roberto, Massimo Marchi, Paolo Tirelli, Aldo Calcante, Marcello Iriti, M Hocevar, and H Ulbrich (2014). "Crops agricultural robot: application to selective spraying of grapevine's diseases." In: *Proceedings of the Second RHEA International Conference on Robotics and associated High-technologies and Equipment for Agriculture* (cit. on p. 31).

Ostafew, Chris J, Angela P Schoellig, and Timothy D Barfoot (2013). "Visual teach and repeat, repeat, repeat: Iterative learning control to improve mobile robot path tracking in challenging outdoor environments." In: *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*. IEEE, pp. 176–181.

Remagnino, Paolo, Dorothy N Monekosso, and Lakhmi C Jain (2011). *Innovations in Defence Support Systems-3: Intelligent Paradigms in Security*. Vol. 336. Springer Science & Business Media (cit. on p. 13).

Ringdahl, Ola, Polina Kurtser, Ruud Barth, and Yael Edan (2016). "Operational flow of an autonomous sweetpepper harvesting robot." In: *The 5th Israeli Conference on Robotics 2016, 13-14 April 2016. Air Force Conference Center Hertzilya, Israel* (cit. on p. 31).

Santos, Pablo Gonzalez-de, Angela Ribeiro, and C Fernandez-Quitanilla (2012). "The RHEA Project: using a robot fleet for a highly effective crop protection." In: *Proceedings of the International Conference of Agricultural Engineering (CIGR-Ageng'12), Valencia, Spain* (cit. on p. 31).

Serrano, Daniel, Pietro Astolfi, Gianluca Bardaro, Alessandro Gabrielli, Luca Bascetta, and Matteo Matteucci (2017). "GRAPE: Ground Robot for vineyArd Monitoring and ProtEction." In: *ROBOT 2017: Third Iberian Robotics Conference*. Vol. 1. Springer, p. 249 (cit. on pp. 28, 37).

Shamah, Benjamin (1999). "Experimental Comparison of Skid Steering vs. Explicit Steering for Wheeled Mobile Robot," M. Sc." In: (cit. on p. 12).

Thrun, Sebastian (2002). "Probabilistic robotics." In: *Communications of the ACM* 45.3, pp. 52–57 (cit. on p. 14).

Wang, Tianmiao, Yao Wu, Jianhong Liang, Chenhao Han, Jiao Chen, and Qiteng Zhao (2015). "Analysis and experimental kinematics of a skid-steering wheeled robot based on a laser scanner sensor." In: *Sensors* 15.5, pp. 9681–9702 (cit. on pp. 12, 14).

Yaguchi, Hiroaki, Kotaro Nagahama, Takaomi Hasegawa, and Masayuki Inaba (2016). "Development of an autonomous tomato harvesting robot with rotational plucking gripper." In: *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*. IEEE, pp. 652–657 (cit. on p. 53).

Yi, Jingang, Junjie Zhang, Dezhen Song, and Suhada Jayasuriya (2007). "IMU-based localization and slip estimation for skid-steered mo-

bile robots." In: *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*. IEEE, pp. 2845–2850 (cit. on p. 12).