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 |

# 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 sofware 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 APIs 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 offical 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 provides tools and libraries for obtaining, building, writing, and running code across multiple computers.*

1 http://www.fao.org/fileadmin/templates/rap/files/uploads/E-agriculture_Solutions_Forum.pdf
2 http://wiki.ros.org/ROS/Introduction

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 tecniques that nodes can use to communicate among them. Keep in mind that, even if these tecniques differs a lot, they all are strongly typed *i.e.* in order to define a channel (with *channel* we now mean one of the tecnique 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 throught 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:

- this technique leads to a strong decoupling between publishers and subscribers to a topic, because a publisher node

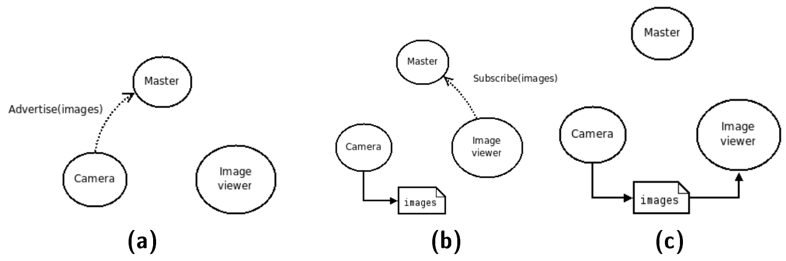


**Figure 1.1:** *Three phases of the setup of communication of nodes throught topics.*

is, from an high-level perspective[3], not even aware of the presence of subscribers, and viceversa.

- 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 Laser Imaging Detection and Ranging (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 of execution time (*e.g.*, navigation of a robot from one point to

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.

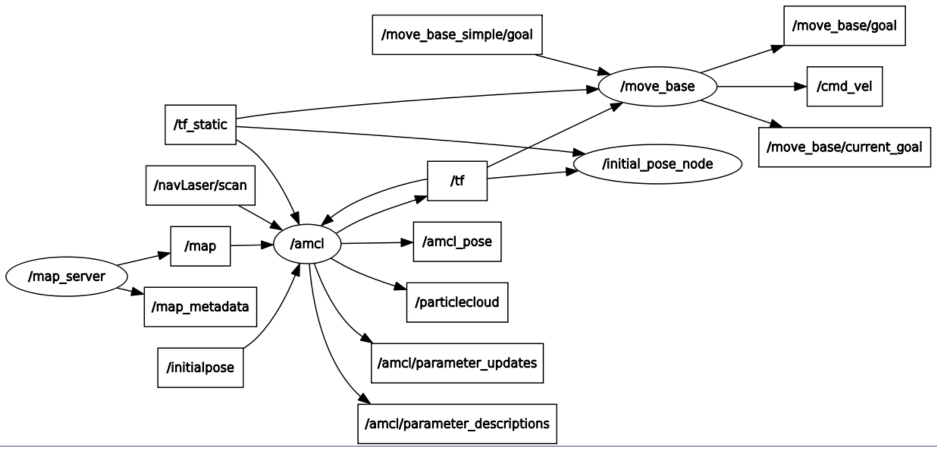


**Figure 1.2:** *An example of the* ROS *Computation Graph, visualized with* rqt*: nodes are represented with circles, rectangles represent topics, and arrows go from a node to a topic it publishes on, or from a topic to a node subscribed to it. It's easy to recognize the* many-to-many *relationship.*

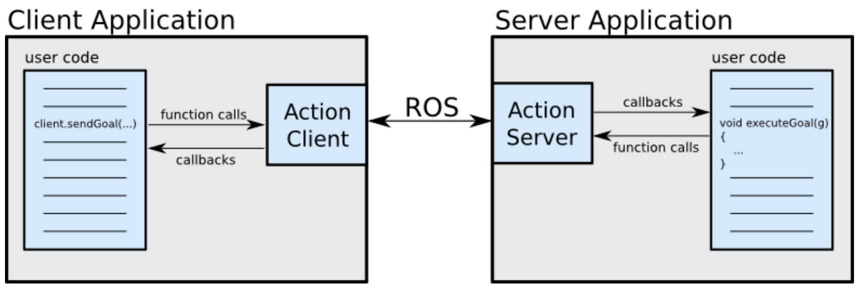


**Figure 1.3:** *Sketch of the implementation of actionlib, through* ROS *topics and a callback system.*

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. 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 1.3). Asynchronicity is provided by the use of callbacks.

## 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:

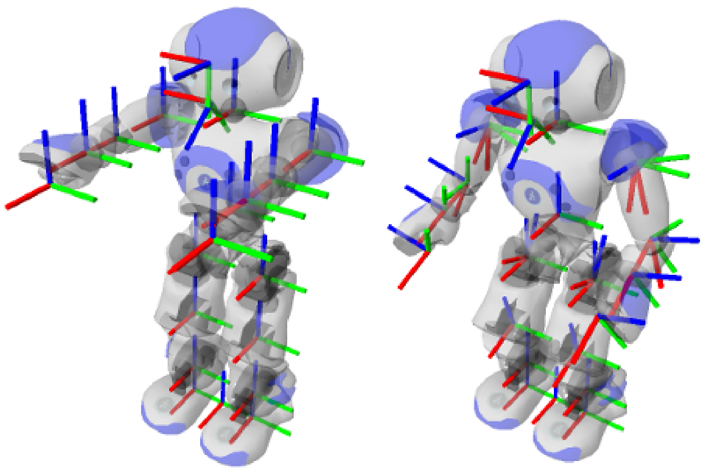


**Figure 1.4:** *A robot in 2 different positions, with* tf *frames in evidence: x-axis is red, y-axis is green, z-axis is blue. The frames are the same in both configuration, but the transformations (i.e. rototranslations) between them are different.*

> *" [...] 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)*

The utility of such a component is straightforward, even in quite simple robotic systems. We'll describe here a situation we stumpled 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 6. 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.

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 interpulate old values using SLERP (Spherical Linear intERPolation) to return a transformation for which there is no measured value at the requested timestamp.

In figure 1.4 you can see a graphical representation of the reference frames tracked in a Nao Robot, while figure 1.5 shows an example of *tf* graph visualized with visualization framework *rqt*.

## 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

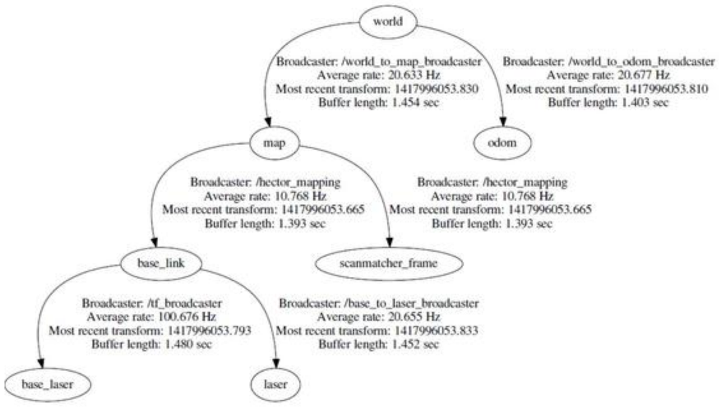

world
Broadcaster: /world_to_map_broadcaster
Average rate: 20.633 Hz
Most recent transform: 1417996053.830
Buffer length: 1.454 sec
Broadcaster: /world_to_odom_broadcaster
Average rate: 20.677 Hz
Most recent transform: 1417996053.810
Buffer length: 1.403 sec
map
odom
Broadcaster: /hector_mapping
Average rate: 10.768 Hz
Most recent transform: 1417996053.665
Buffer length: 1.393 sec
Broadcaster: /hector_mapping
Average rate: 10.768 Hz
Most recent transform: 1417996053.665
Buffer length: 1.393 sec
base_link
scanmatcher_frame
Broadcaster: /tf_broadcaster
Average rate: 100.676 Hz
Most recent transform: 1417996053.793
Buffer length: 1.480 sec
Broadcaster: /base_to_laser_broadcaster
Average rate: 20.655 Hz
Most recent transform: 1417996053.833
Buffer length: 1.452 sec
base_laser
laser

**Figure 1.5:** *An example of* tf *tree*

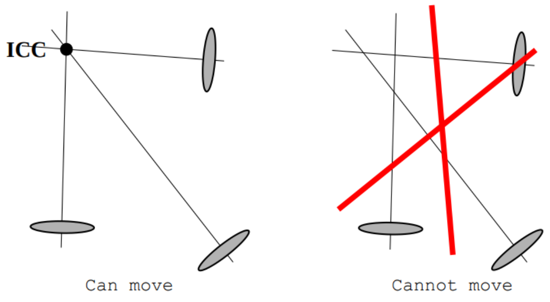


**Figure 1.6:** *If the axis of all the wheels intersect in a single point, it's called ICC and the robot can move without slipping*

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 travelled 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 perendicular 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

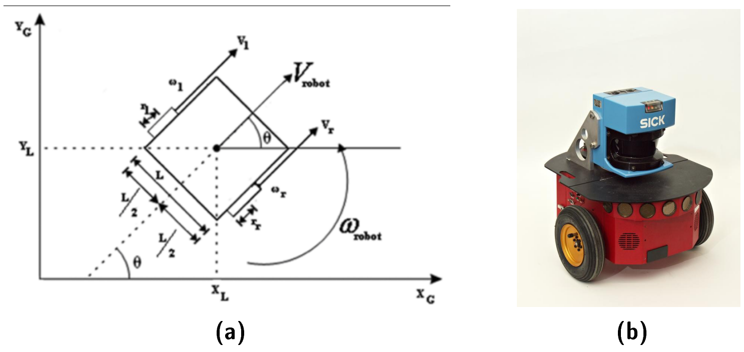


**Figure 1.7:** *On figure 1.7a, the scheme of a differential drive motion model; in figure 1.7b, an example of a differential drive robot (Pioneer 3DX).*

Instantaneous Center of Curvature (ICC) (see Figure 1.6), 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. 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 descript 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 3.1, the robot we used is the Husky platform (see Figure 1.11) 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 1.7), 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 $\omega$, 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:

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

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 $(x, y, \theta)$. We assume that in the time interval $t_0 \to (t_0 + \delta t)$ the values $v_r$ and $v_l$ are constant; if we observe figure 1.8 under these condition, we have:

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

Write now the expressions for $(x', y', \theta')$:

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

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 $(x_0, y_0, \theta_0)$, knowing that:

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

where $V(t)$ represent the complexive 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 $(x, y, \theta)$ by integrating the speed of the robot over time, that is:

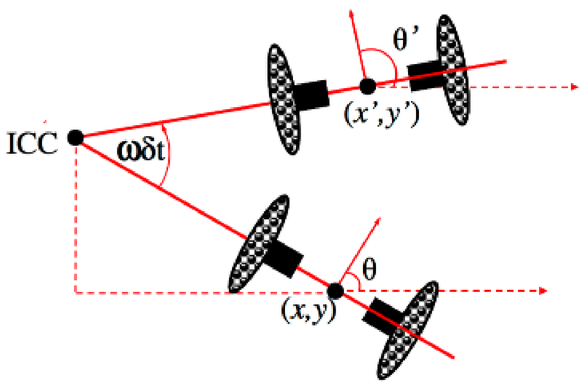


**Figure 1.8:** *Differential drive:* $(x, y, \theta) \rightarrow (x', y', \theta')$

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

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

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

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 1.11, 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

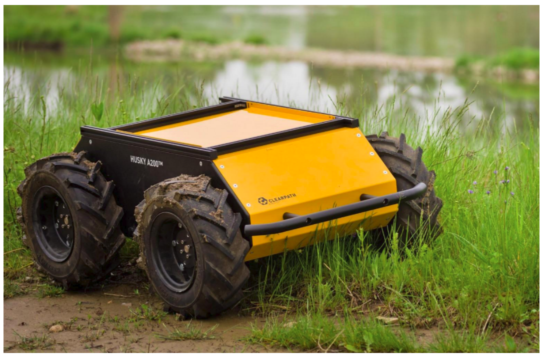


**Figure 1.9:** *Husky platform from Clearpath Robotics is the platform used for the development of GRAPE project.*

- 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:

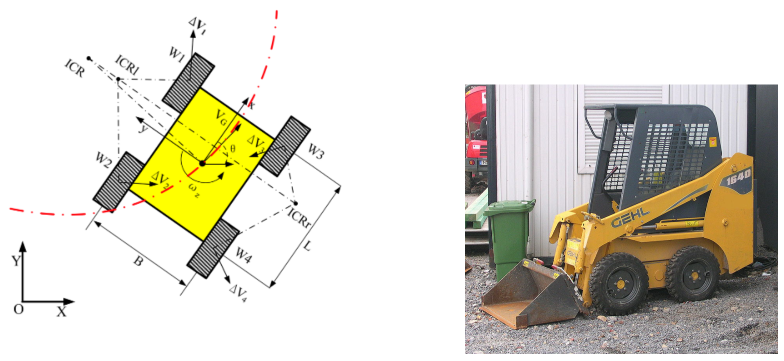
$$v_{fr} = v_{rr} \tag{1.13}$$
$$v_{fl} = v_{rl} \tag{1.14}$$

This type of motion model is called *skid steering*, and is often use in real-world applications (see figure 1.10b) 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 consuption 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 accettable approximation the skid steering as an equivalent differential drive system, where the distance between the wheels is



(a) (b)

**Figure 1.10:** *On figure 1.10a, the scheme of a skid steering motion model; in figure 1.10b, an example of a skid steering vehicle.*

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

- the mass center of the robotis 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:

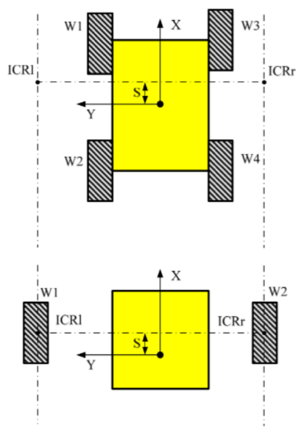


**Figure 1.11:** *The equivalence of differential drive motion model and skid steering model according to Wang et al., 2015*

- *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 tecniques make a step further, and aim to the construction of a single common representation, using all the available sensor sources. The difference betweeen multi-sensor integration and sensor fusion is described graphically in image 1.12, 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-

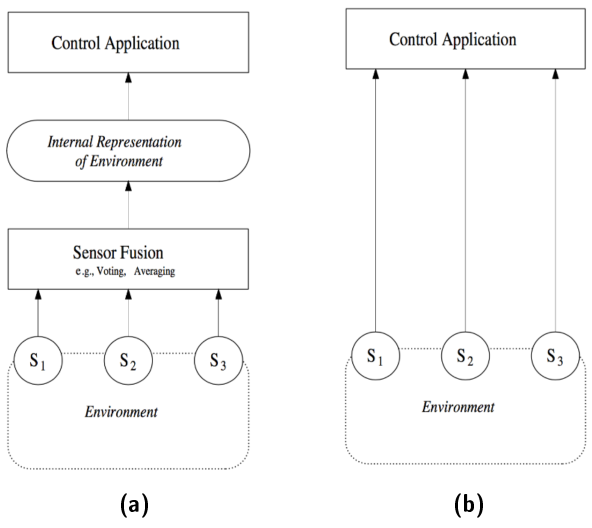


**Figure 1.12:** *Sensor fusion 1.12a vs multi-sensor integration 1.12b*

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[4], so we opted for this last one. Further details about our usage or Robot Localization will be given in Chapter 5, while we are now giving a few hints about its general functioning.

### *Robot Localization*

Citing Robot Localization documentation[5]:

> *"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."*

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

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

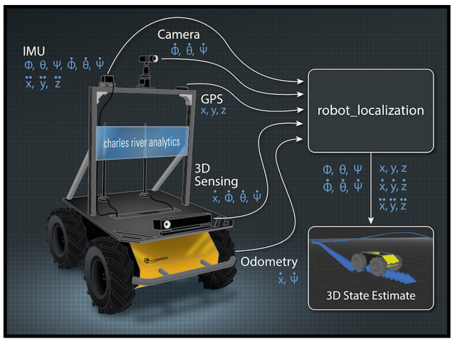

Camera
$\dot{\phi}, \dot{\theta}, \dot{\psi}$
IMU
$\phi, \theta, \psi, \dot{\phi}, \dot{\theta}, \dot{\psi}$
$\ddot{x}, \ddot{y}, \ddot{z}$
GPS
x, y, z
charles river analytics
3D Sensing
$\dot{x}, \dot{\phi}, \dot{\theta}, \dot{\psi}$
Odometry
$\dot{x}, \dot{\psi}$
robot_localization
$\phi, \theta, \psi$
$\dot{\phi}, \dot{\theta}, \dot{\psi}$
x, y, z
$\dot{x}, \dot{y}, \dot{z}$
$\ddot{x}, \ddot{y}, \ddot{z}$
3D State Estimate

**Figure 1.13:** *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 $\psi, \theta, \phi$ 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 5.
- *sensor_msgs/Imu*: the output of an Inertial Measurement Unit (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

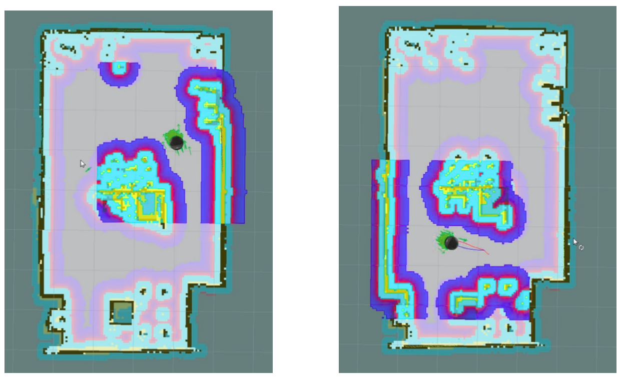


**Figure 1.14:** *Local and global costmap visualized with* RViZ. *As you can see, the local costmap (in brighter colors) is built around the robot, and moves together with it.*

from the various sensors for a large enough amount of time, the filter keeps producing odometry estimation exploiting an internal motion model. Remeber 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 1.15):

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.

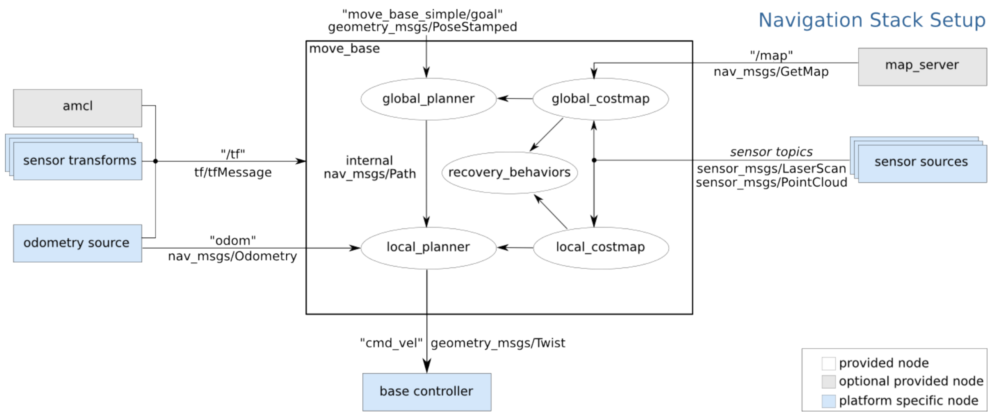


**Figure 1.15:** *A scheme representation of the Navigation Stack; see the color legend for classification in provided, optional provided, and platform specific nodes.*

AMCL Implementation of a probabilistic localization system, based on the Monte Carlo localization, as described in Fox et al., 1999. (TODO se alla fine lo usiamo lo descrivo meglio magari).

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.

# 2 THE GRAPE PROJECT

The GRAPE project chapter SKETCH:

- descrizione del progetto e i suoi macro goals (navigazione autonoma, monitoraggio della salute della vigna, dispenser dei ferormoni, interfaccia user friendly)
- enti collaborativi (polimi, eurecat, vitirover) con relativi compiti
- difficoltà previste: outdoor, terreno accidentato
- parti su cui ho lavorato io direttamente (design dell'architettura sw, scan motion, parti del deployment)
- qualche foto di vigna/dispensers

# 3 GRAPE HARDWARE AND SOFTWARE ARCHITECURE

Grape architecture SKETCH:

## GRAPE HW ARCHITECTURE

## GRAPE SW ARCHITECTURE

- Hardware del robot:
  - scelta dell'husky (adatto per rugged terrain, supporto di pacchetti ROS), foto
  - scelta del braccio, con pro (compatto, 6 dof, power consumption non elevata) e contro (movimenti non precisi, fragile, scarso controllo di collisione)
  - lidar sul braccio per scan
  - lidar davanti per navigazione e eventualmente localizzazione
  - camera montata sul braccio
  - velodyne davanti (a cosa serve?)
  - camera fissa per eventualmente scattare foto alla vigna
  - imu
- foto del robot
- Software del robot
  - descrizione più precisa di quale è nel complesso la procedura: viene dato a mano (interfaccia grafica di Eurecat) una posizione finale, e serie di waypoints intermedi. Procedura di scan che usando PCL trova punti adatti al deployment. Procedura di deploymnet. Waypoint successivo.
  - struttura col coordinatore, che chiama le 3 action (move_base, scan, deploy)
  - di ogni azione, il file che descrive i tipi di goal/result/feedback, con spiegazioni del significato dei parametri

# 4 THE ROBÌ PROJECT

The Robì project chapter:

- obiettivo: realizzare un'altra base mobile alternativa all'husky su cui portare il sistema GRAPE
- architettura hw: sensori che ci sono montati a bordo, spiegazione sistema controllo motori tramite CAN usando la board STM
- architettura sw: compatibile con quella di Grape, a meno dei nodi driver dei sensori e attuatori
- spiegazioni e immagini del vario hardware

# 5 SENSOR FUSION AND LOCALIZATION IN GRAPE

Localization Chapter SKETCH:

- problemi avuti: mappa molto ripetitiva, il robot tenta di passare attraverso i filari, odometria delle ruote poco affidabile dato il terreno, problemi con amcl
- virtual obstacles: cenni (perché non l'ho fatto io) e immagini
- configurazione di robot localization, con i sensori usati (ruote+imu+gps)
- non sappiamo ancora quale sarà la soluzione utilizzata alla fine tra la versione mapless e amcl integrato con robot_localization (o eventuali altre soluzioni che salteranno fuori). Una volta che è stata presa una decisione, presentare quella come soluzione definitiva e spiegare perché l'altra è stata giudicata meno efficace

# 6 KINOVA ARM

Kinova Arm chapter: SKETCH

- moveit: cos'è, concetti di planning e execution
- scopo del braccio in GRAPE: scan della pianta, deployment, automa a stati che rappresenta la sequenza di azioni del braccio
- problema 1 dovuto alla scarsa precisione: accenni alle parti di visual servo con marker per compensare questo problema in fase di grasping e deployment (cenni, fatta da Polito e Bascetta).
- problema 2 dovuto ai limiti di execution di moveit: impossibilità di fare un movimento di scansione fluido e completo usando moveit, metodo della jacobiana e la pubblicazione delle velocità
- utilizzo della camera per validazione del grasping e del deployment
- foto del braccio, immagini della nuvola di punti processata dal codice di ferran, alcune immagini dell'image processing nelle fasi di visual servo e di validazione di grasping/deployment

# 7 EXPERIMENTAL RESULTS

The experimental results chapter:

- TODO: decidere prima della integration week che cosa vogliamo metterci, così sappiamo che dati procurarci

# BIBLIOGRAPHY

Calabrese, Luca (2014). "Robust odometry, localization and autonomous navigation on a robotic wheelchair." In: (cit. on p. 14).

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. 14).

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. 14).

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. 14).

Elmenreich, Wilfried (2002). "Sensor fusion in time-triggered systems." In: (cit. on p. 12).

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. 17).

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. 16).

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. 14).

Moore, T. and D. Stouch (2014). "A Generalized Extended Kalman Filter Implementation for the Robot Operating System." In: *Proceedings of the 13th International Conference on Intelligent Autonomous Systems (IAS-13)*. Springer (cit. on p. 14).

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. 12).

Shamah, Benjamin (1999). "Experimental Comparison of Skid Steering vs. Explicit Steering for Wheeled Mobile Robot," M. Sc." In: (cit. on p. 11).

Thrun, Sebastian (2002). “Probabilistic robotics.” In: *Communications of the ACM* 45.3, pp. 52–57 (cit. on p. 13).

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. 11, 13).

Yi, Jingang, Junjie Zhang, Dezhen Song, and Suhada Jayasuriya (2007). “IMU-based localization and slip estimation for skid-steered mobile robots.” In: *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*. IEEE, pp. 2845–2850 (cit. on p. 11).