POLITECNICO DI TORINO

Master's Degree in
Mechatronic Engineering - Technologies for eMobility

Master's Degree Thesis

Design and development of vineyard row following algorithms for agricultural robotic vehicles

Supervisors

Prof. Alessandro RIZZO

Ing. Antonio PETITTI

Candidate

Luca TERZO

26 Ottobre 2021

Summary

The whole thesis project was realised in collaboration with STIIMA-CNR (Sistemi e Tecnologie Industriali Intelligenti per il Manifatturiero Avanzato - Consiglio Nazionale delle Ricerche), Bari, Italy.

The need to improve the relationship between phenotyping and automation is increasing due to the world's current sub-optimal and worsening situation regarding food production. It is therefore important to research and implement new methods in order to increase sustainability and food security worldwide. The first step we used to automate phenotyping is to enable the robot to move autonomously within the rows of vines. This is made possible by data acquisition through several Intel Real Sense D345 from which PointClouds are exploited. The points are used to construct a suitable plan that best fits the row. By extrapolating the data from the plane normal, the robot can recognise and adjust its angle to the row and also the distance so that it is always parallel. In addition, with the ultimate aim of improving the torque distribution in the four drive wheels, a system was developed to calculate the odometry of the robot, obtaining the x and y distances from the starting position and the rotation angle. The entire system was tested and verified through several indoor and outdoor tests, which yielded good results, thus validating the methods used. The collected data was further analysed and, through an offline study, a Kalman Filter was designed and tested to smooth the online data collected and thus avoid decision inaccuracies of the robot.

Acknowledgements

"All the world's a stage"
As You Like It, William Shakespeare

Table of Contents

List of Figures

Glossary

Acronyms

AWD all-wheel-drive

DEM discrete element method

ETS error-tolerant switch

FEM finite element method

HTP high-throughput phenotyping

LiDAR light detection and ranging

REKF robust extended Kalman filter

SfM structure from motion

UAV unmanned aerial vehicle

Symbols

b wheel width

c soil cohesion

c_1 contact angle coefficient

c_2 contact angle coefficient

C_σ pressure sinkage modulus

F_B bulldozing resistance

F_{DP} drawbar pull force

F_N normal force

F_S lateral force

h sinkage

I coordinate rotational matrix

I_B bulldozing distribution

j shearing deformation

k_c cohesive modulus

k_ϕ frictional modulus

M_D driving torque

M_S steering torque

n sinkage exponent

r wheel radius

s slip ratio

X stress vector

z_1 maximum sinkage

z_2 residual sinkage

Greek symbols

β slip angle

η viscous damping coefficient

θ wheel-soil contact angle

θ_1 entrance angle

θ_2 exit angle

θ_m maximal normal stress angle

μ tangential friction coefficient

ρ_d soil density

σ normal stress

Σ_T terrain reference frame

Σ_W wheel reference frame

τ shear stress

τ_x longitudinal shear stress

τ_y lateral shear stress

ϕ internal friction angle

ψ_c destructive angle

ω_D heading rate

ω_S angular rate

Chapter 1

Introduction

1.1 Precision Agriculture

Several definitions of precision agriculture exist, but one of the most quoted is an approach to managing the agricultural production process in order to "do the right thing, at the right time, at the right place" [1]. This definition aptly sums up the principles and objectives of precision agriculture, namely to improve the efficiency of the inputs of dynamic process management, but taking into account the variability in time and space of the factors affecting the agricultural production process and compensating for this variability.

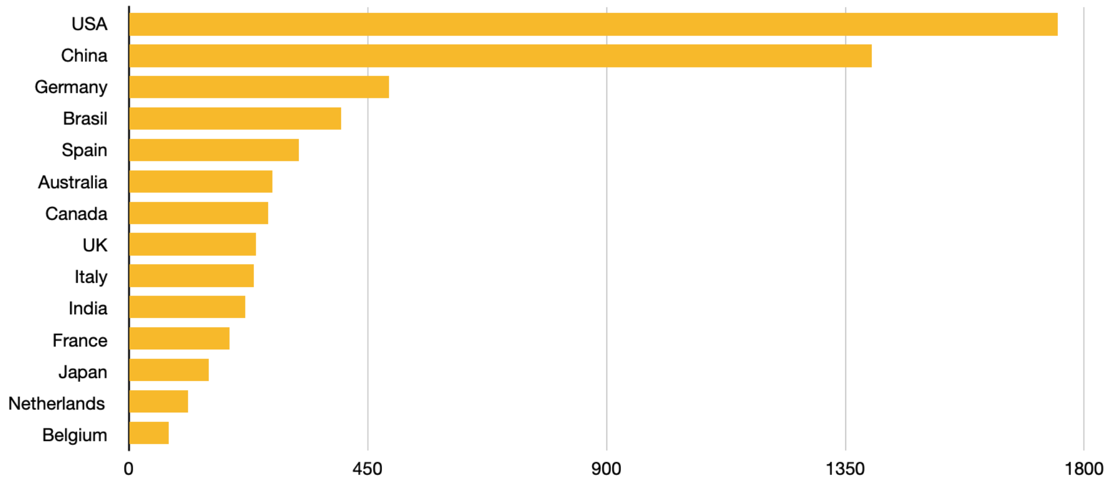


Figure 1.1: Total number of scientific publications on precision agriculture from the Scopus bibliographic database, using the search terms "precision agriculture" and "precision farming" in the period 1990-2015.

At present, precision agriculture is becoming increasingly popular mainly in countries where the technology related to agriculture is more advanced [2]. Research on precision agriculture in Italy has had a good scientific productivity despite the fact that it has not been able to count on funding comparable to that of other countries, and in fact it ranks ninth in the world in terms of the number of scientific publications , as depicted in Fig. 1.1).

In Italy, specifically, as shown in Fig. 1.2, the sectors in which most research is concentrated concern wine and cereals [2]: in viticulture, for example, income maximisation is achieved mainly by increasing the value of the product (i.e. its quality) thanks to process optimisation through precision farming.

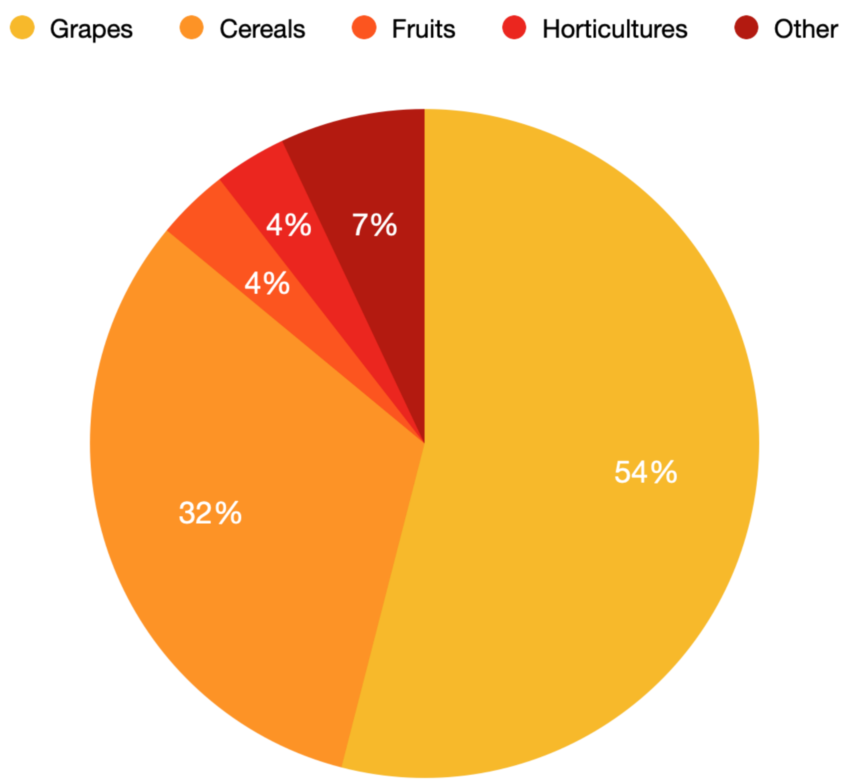


Figure 1.2: Scientific publications on precision agriculture made in Italy in the period 1990-2015 sorted by crop type.

A further cross-cutting objective for the production chains and technologies involved, which supports the introduction of precision agriculture techniques, is the need to quantify the hours of work, fertilisers, seeds, weed-killers, fuels, machinery and lubricants that are now used in unnecessary quantities and at unnecessary times. In fact, the adoption of the different techniques of precision agriculture, allows to optimise the management activity and to reduce, up to almost zero, the waste, so as to find a large economic saving, energy and a net reduction of the environmental impact.

1.2 Intelligent self-driving tractors

The current world situation with regard to food production is not optimal and is getting worse [3]. As the world's population is currently growing, the world's farmers are faced with the very difficult challenge of maximising crop yields on ever smaller plots of land. In addition, there are major problems such as drought and flooding, plant disease and significant economic costs [4]. For these reasons, it is therefore important to research and implement new methods in order to increase sustainability and food security worldwide. One avant-garde innovation useful for increasing crop yields, limiting waste of resources and the use of pesticides and plant protection products targeted to actual needs, is the High-Throughput Phenotyping process (HTP) [5]. It involves observing and analysing plants and their fruits in a specific way, so that decisions and predictions can be made based on the actual characteristics that can be identified plant by plant. Currently, the

Figure 1.3: Example of self-driving tractors.

phenotyping of plants is a largely manual process involving many workers carrying out checks and analyses. This process is particularly laborious and time-consuming, requires defined skills and can be subject to subjective operator estimates. For this reason, it is necessary to optimise the process by introducing the use of autonomous, airborne or ground, vehicles. One of the technology sectors with the highest impact on agriculture is certainly the robotics field, in particular autonomous robotics for specific work on plants, as well as robot-guided sensor platforms. Robots are in fact used to reduce the human labour component in the various stages of tillage, from soil preparation and sowing to harvesting, by means of driverless tractors (Fig. 1.3) [6].

Significant resources have been dedicated to research and development of intelligent tractors to meet the needs of the agricultural sector, save energy, protect the environment and improve productivity [7].

In any case, unfortunately, although the external environment is well structured, the actual structure and layout changes from field to field and it is therefore of fundamental importance to achieve a high level of application dynamism in the robot that makes it capable of adapting to any eventuality. It follows then that the primary skill that needs a strong level improvement is visual perception.

1.3 Purpose and approach

The main topic of this thesis is to automate the inspection process of intelligent tractors between the vineyard rows and to increase their efficiency. Therefore, the main objective is the development of specific controls and commands in order to follow the row. For this purpose, the robot must be able to maintain the right distance and the right angle to the vineyard row.

a plane identification algorithm which requires as input the acquired point cloud. The aim is to work on-line by analyzing the point cloud during the execution of the row following task. Then, the plane normal is used as a reference in order to perform the calculations for the robot manoeuvres by adjusting the angle and distance. The data obtained is processed through a Kalman Filter to exclude noise and thus improve its quality. A model used to calculate the robot odometry is also proposed, so that the x and y distance of the robot from the initial position can be determined. This will be useful for the next steps in calculating the slip in order to improve the torque distributed in the drive wheels.

1.4 Outline

This thesis consists of five chapters. This chapter introduces purpose and approach of this research.

In Chapter 2, an in-depth analysis of the literature study is carried out. Starting with the less recent cornerstones and ending with the latest innovations and applications. It then shows the most interesting methods of implementation.

In Chapter 3, the problems to be addressed are identified as the final objectives. Proposed solutions for the purpose are then explained in detail and their development and implementation is shown.

In Chapter 4, the tests carried out are shown and the results analysed. In Chapter 5, conclusions are drawn with a focus on future work.

Chapter 2

Automation systems for phenotyping in the agricultural domain

2.1 Vision systems for automation

Precision agriculture is an agricultural management strategy that uses modern tools. It focuses on the implementation of agronomic measures and takes into account the real needs of the crops and the biochemical and physical properties of the soil. Thanks to today's technology, it is finally possible to monitor the different stages of agricultural production. Time-of-flight, stereo and RGB cameras can certainly be used to obtain maps of the state of vegetation. These maps then help farmers to maximise agricultural yields. Thus, it is possible to take a series of photographs in a few minutes without any effort. These will then be very useful in understanding the health of the crops.

Remote sensing is a possible solution to the problem of obtaining spatial information on the vegetative state of crops without being invasive. This is made possible by unmanned aerial vehicle (UAVs) equipped with RGB cameras. By acquiring numerous images of the surface, a dense point cloud can be obtained using Structure from Motion (SfM, [8, 9]) algorithms. In this way, after being filtered and meshed, the digital model of the surface is obtained. This methodology has been applied both for the recognition of vineyards [10] and for other types of crops [11]. In [12] it is shown how the SfM can be of great help in the reconstruction of the point clouds. In fact, through the SfM algorithm it is possible to obtain information such as orientation, height, width and spacing of the rows and also it allows to optimally separate the background from the vineyard (2.1).

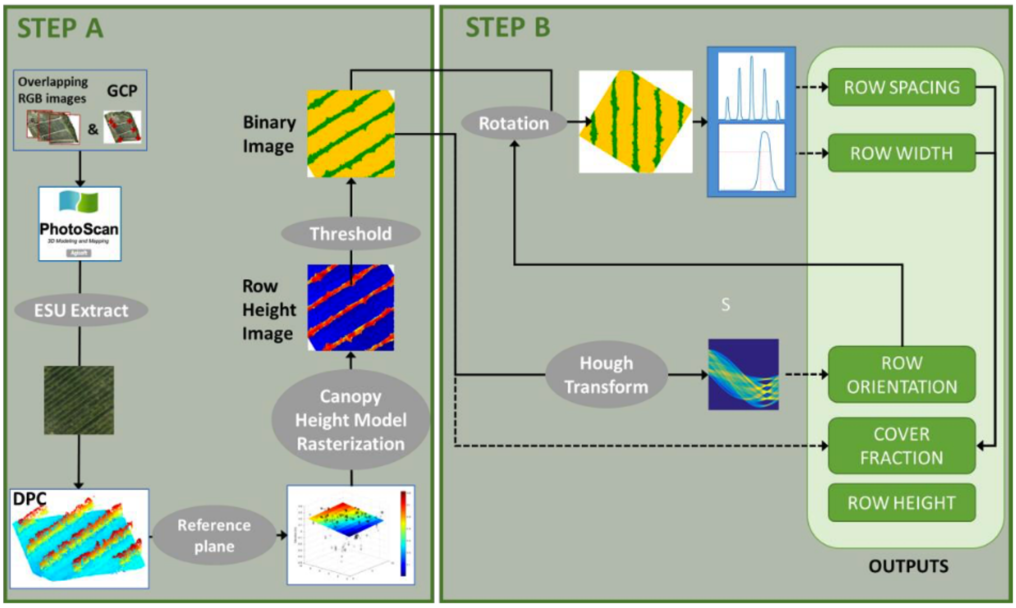
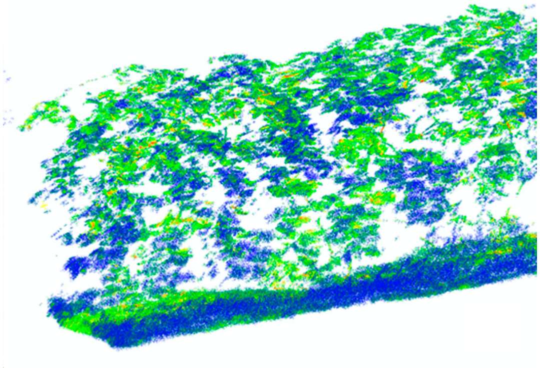


Figure 2.1: The figure shows how the point cloud calculation algorithm works. By superimposing the acquired RGB images, a binary image is obtained (step A). The characteristics of the vineyard are then extrapolated: orientation, height, width (step B).

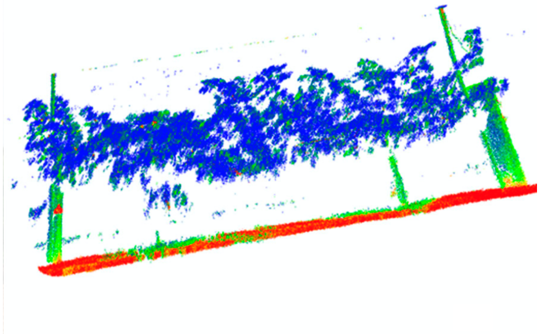
Light detection and ranging (LiDAR) may also be particularly useful for phenotyping in vineyards [13]. It works by sending pulses of laser light over a surface very quickly and, by measuring the time it takes for the reflected light to return and its intensity using sensors, three-dimensional coordinates are obtained. It has been used, with positive results, to monitor the growth status of plants of different crops [13, 14, 15, 16, 17]. In [18] for example, it is shown how LiDAR technology can be used to scan vineyards and capture vine growth characteristics. In particular, it is shown that depending on the intensity values of the acquired subject, the colour changes (Fig. 2.2).

2.2 Wheel-soil interaction models

Working on agricultural land, in many cases, requires high traction forces developed by the tractor's wheels. A tyre interacts with the soil through a system of stresses along the contact surface between the tyre and the soil, and this interaction generates deformations in both the soil and the tyre. The soil is subject to normal and tangential stresses on the tyre's contact surface, and tangential stresses increase rapidly as tractive force increases, and can lead to the breakdown of compressed soil



(a) Old and yellow foliage.



(b) Novel and green foliage.

Figure 2.2: The figure shows that depending on the colour of the leaf, a different colour can be obtained. This can be achieved by higher or lower reflectance properties. In (a), the intensity values are higher and are reflected more as the leaf tissue absorbs less infrared light and the dots appear green. In contrast, in (b) the light is absorbed more and is shown in blue. The ground is displayed in red.

between the tread grooves (soil shear effect). This leads to the formation of a surface layer of soil lacking mechanical resistance and, therefore, highly exposed to erosion phenomena, and an underlying layer in which the effect of shear deformations contributes to altering the functionality of the soil structure.

In the literature, soil is usually modelled as an elastic or plastic material. Elasticity theory allows the soil to be modelled as an elastic medium. This method has found applications in the study of soil compaction and soil damage due to vehicular traffic. On the other hand, modelling the soil as a rigid and perfectly plastic material has found applications in predicting the maximum traction developed by off-road vehicles. Both of these physical models have limitations, for example, the elasticity theory is only valid for a limited vehicle load, so the ground can be considered elastic. Whereas, the plastic equilibrium theory can only be used to estimate the maximum vehicle load that the ground can bear, but cannot be used in the calculation of wheel sinkage.

Following the theoretical assumptions mentioned before, several approaches have been developed in modelling wheel-ground interaction. The four main ones are:

- Finite Element Method (FEM);
- Discrete Element Method (DEM);
- Empirical model;
- Semi-empirical model.

2.2.1 Finite Element Method (FEM)

Advances in computational techniques in recent years make it possible to model terrain using the finite element method (FEM) or the discrete element method (DEM). These methods have the potential ability to investigate the dynamic aspects of the physical nature of vehicle-soil interaction in detail. The finite element method is a numerical technique that finds approximate solutions by subdividing complex problems described by partial derivative equations (PDE) into a finite number of small segments. In recent years, studies on the application of FEM to the analysis of wheel-ground interaction have progressed significantly. In order to accommodate different types of soil behaviour, a number of constitutive models have been introduced. Due to the inelastic deformation of the soil when subjected to normal pressure and/or shear stress at the wheel-soil interface, the behaviour of soil materials is performed by means of pressure-dependent elasto-plastic models. However, the high computational cost still hinders its application in real-time operations.

Discrete Element Method (DEM)

The discrete element method is another numerical approach that represents the soil as a set of many discrete elements, where each is described by its size, shape, position, velocity and orientation. In its basic form it assumes that each element has a stiffness characterised by a spring constant k and has damping denoted by a viscous damping coefficient η. It also assumes that along the wheel-ground contact there is friction in the tangential direction denoted by the coefficient μ. However, the computational cost is of paramount importance for real-time applications, therefore, this approach is also costly to adopt due to the high computational requirements.

Empirical model

The empirical model is generally obtained by interpolating a large amount of experimental data. Cone penetrometer and bevameter are typical instruments that can measure and derive soil parameters. However, in most cases the mathematical relations obtained by means of interpolation data have no physical meaning and are strictly specific to the studied environment.

Semi-empirical model

Semi-empirical model theory deals with physical dynamics under a few assumptions. Bekker pioneered the formulation of terramechanical models [19], [20]. Later, Wong [21] and Reece [22] developed another model, which is widely used in straight and constant motion. These models consider the wheel as a non-deformable rigid ground travelling on a soft deformable ground, combining both elastic and plastic theories. However, over the years various research has been carried out to improve these models. Semi-empirical models are derived from theoretical analysis and experimental data. They are most commonly used because of their high fidelity and physical significance and are also suitable for real-time applications [23].

2.2.2 Reference model

A free body diagram of a rigid drive wheel on a soft soil is shown in Fig. 2.3. Two different reference frames for wheel and terrain are defined, respectively $\Sigma_W\{x_w, y_w, z_w\}$ and $\Sigma_T\{x_T, y_T, z_T\}$. The rigid drive wheel has radius r and width b, with θ_1 and θ_2 indicating the entry and exit angles. Between them the contact angle $\theta \in [\theta_1, \theta_2]$ is considered. Furthermore, z_1 and z_2 represent the maximum and residual sinkage. In the case of a steerable wheel, the steering angular rate ω_S must be considered in addition to the heading rate ω_D. The slip angle β is to be considered between the direction of the speed and the longitudinal axis of the wheel, and the shear stress $\tau\{\tau_x, \tau_y\}$ is therefore generated. The normal stress is denoted by σ. Finally, assuming that the wheel is resting perpendicularly on flat ground, three forces can be highlighted, the normal force F_N, the drawbar pull force F_{DP} and lateral force F_S, and two resistant torques, driving torque M_D and steering torque M_S, from the soil to the wheel.

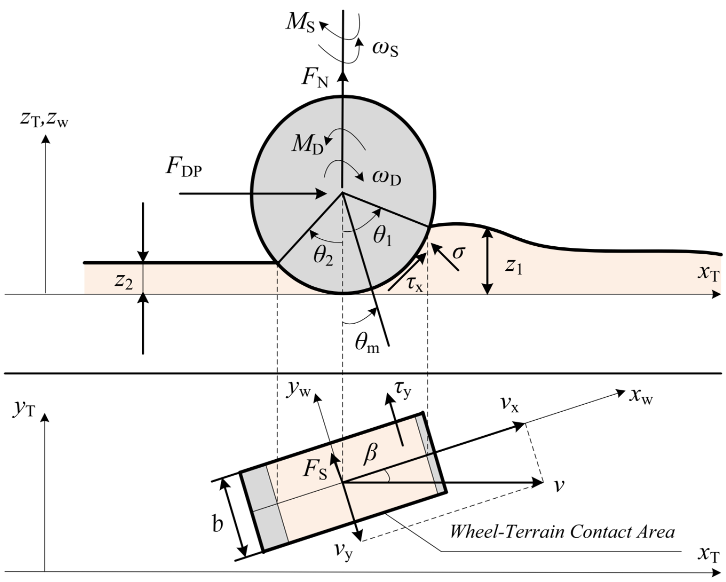


Figure 2.3: Free body diagram of a rigid drive wheel on a soft soil.

As expressed through Bekker's terramechanical theory in [24], the wheel-soil interaction generates shear and normal stresses. The latter can be calculated through [22]

$$\sigma(\theta) = C_\sigma h^n(\theta) \tag{2.1}$$

where $h(\theta)$ is the wheel sinkage in function of θ, n the sinkage exponent and C_σ the pressure sinkage modulus that is usually expressed as

$$C_\sigma = \frac{k_c}{b} + k_\phi \tag{2.2}$$

where k_c and k_ϕ are respectively the cohesive and frictional moduli of sinkage. The wheel sinkage $h(\theta)$ of (2.1) is geometrically given by

$$h(\theta) = \begin{cases} r(\cos\theta - \cos\theta_1) & \text{if} \quad \theta \in [\theta_m, \theta_1] \\ r\Big\{cos\Big[\theta_1 - \frac{\theta-\theta_2}{\theta_m-\theta_2}(\theta_1 - \theta_m)\Big] - \cos\theta_1\Big\} & \text{if} \quad \theta \in [\theta_2, \theta_m) \end{cases} \tag{2.3}$$

in which θ_m is the angle where the maximal normal stress occurs and it is express as

$$\theta_m = (c_1 + c_2 s)\theta_1 \tag{2.4}$$

and s is the slip ratio defined as

$$s = 1 - \frac{v_x}{r\omega_D} \tag{2.5}$$

where is then possible to notice the relationship between the angular rate ω_D and the forward velocity v_x.

The contact angle coefficients c_1 and c_2 given in (2.4), depend on soil type and, as reported in [22], typical values are $c_1 = 0.4$ and $c_2 \in [0, 0.3]$. It is therefore possible to approximate (2.4), for many slip rate values, as

$$\theta_m = \frac{1}{2}(\theta_1 + \theta_2) \tag{2.6}$$

The entrance and exit angles θ_1 and θ_2 in (2.3) can be calculated as follow

$$\theta_i = (-1)^{i-1}\cos^{-1}\frac{r - z_i}{r} \qquad i = 1{,}2 \tag{2.7}$$

The shear stress is expressed as [25]

$$\tau_x(\theta) = \tau(\theta)\Big(1 - e^{\frac{-j_x(\theta)}{K_x}}\Big) \tag{2.8a}$$

$$\tau_y(\theta) = \tau(\theta)\Big(1 - e^{\frac{-j_y(\theta)}{K_y}}\Big) \tag{2.8b}$$

where $\tau(\theta)$ is the shear stress that corresponds to the normal stress computed by means the Coulomb's equation

$$\tau(\theta) = c + \sigma(\theta) \tan \phi \tag{2.9}$$

in which c the soil cohesion factor and ϕ is the internal friction angle. K_x and K_y in (4.1) are the longitudinal and lateral shearing deformation modulus. As reported in [22] and in [26], the corresponding shearing deformation j_x and j_y can be formulated as

$$j_x(\theta) = r[\theta_1 - \theta - (1 - s)(\sin \theta_1 - \sin \theta)] \tag{2.10a}$$
$$j_y(\theta) = r(1 - s)(\theta_1 - \theta) \tan \beta. \tag{2.10b}$$

Assuming that the speed of the wheel v is constant, the wheel-soil forces can be calculated as follows

$$F_N = rb \int_{\theta_2}^{\theta_1} [\sigma(\theta) \cos \theta + \tau_x(\theta) \sin \theta] d\theta \tag{2.11a}$$
$$F_{DP} = rb \int_{\theta_2}^{\theta_1} [\tau_x(\theta) \cos \theta - \sigma(\theta) \sin \theta] d\theta \tag{2.11b}$$
$$F_S = rb \int_{\theta_2}^{\theta_1} \tau_y(\theta) d\theta + F_B \tag{2.11c}$$

where F_B is the bulldozing resistance [27] shown in Fig. 2.4. Note that only the quantity F_N is known a priori and the rest of the quantities need to be estimated. The bulldozing area is also shown by a ground swell phase with internal friction angle ϕ and a destructive phase with a destructive angle ψ_c. Thus,

$$F_B = rb \int_{\theta_2}^{\theta_1} f_B(\theta) d\theta \tag{2.12}$$

$$f_B(\theta) = D_1 h(\theta) \Big[c + \frac{1}{2} D_2 \rho_d h(\theta) \Big] [r - h(\theta) \cos \theta] \tag{2.13}$$

where

$$D_1 = \cot \psi_c + \tan(\phi + \psi_c), \tag{2.14a}$$
$$D_2 = \cot \psi_c + \frac{\cot^2 \psi_c}{\cot \psi} \tag{2.14b}$$

and ρ_d is the soil density. The destructive angle, can be approximated as

$$\psi_c = \frac{\pi - 2\phi}{4}. \tag{2.15}$$

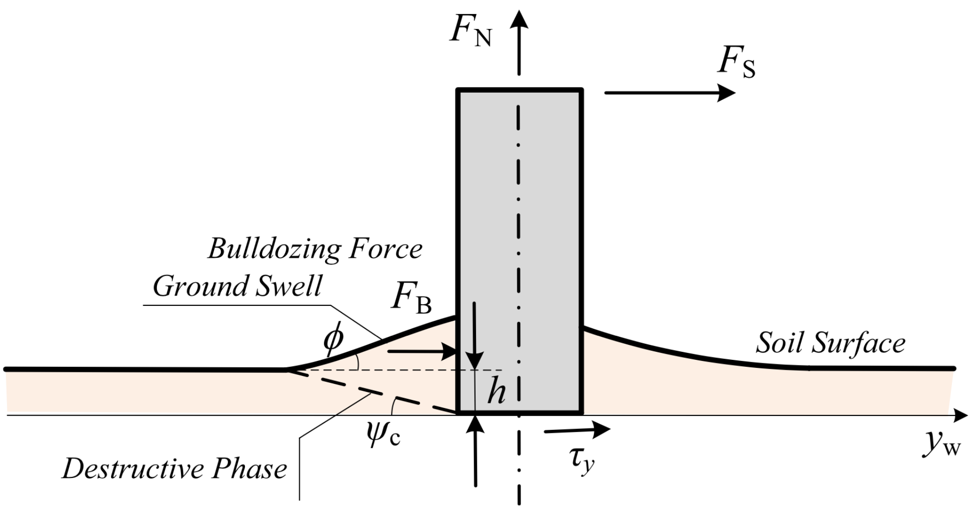


Figure 2.4: Bulldozing resistance on wheel side.

It is possible to rewrite (2.11) in a compact form by defining the force vector $\mathbf{F} = [F_N, F_{DP}, F_S]^T$ and the stress vector $\mathbf{X} = [\sigma, \tau_x, \tau_y]^T$ and obtaining

$$\mathbf{F} = rb\int_{\theta_2}^{\theta_1} \mathbf{I}(\theta)\mathbf{X}(\theta)d\theta + \mathbf{I}_B F_B \tag{2.16}$$

where the coordinate rotational matrix

$$\mathbf{I}(\theta) = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

and the vector $\mathbf{I}_B = [0,0,1]^T$ represent the bulldozing distribution. The driving and steering resistance torque are finally defined as

$$M_D = r^2 b\int_{\theta_2}^{\theta_1} \tau_x(\theta)d\theta \tag{2.17}$$

$$M_S = \int_{\theta_2}^{\theta_1} \sin\theta dF_S - \int_{-\frac{b}{2}}^{\frac{b}{2}} ydF_{DP}. \tag{2.18}$$

Assuming F_{DP} is uniformly distributed along the normal of the wheel plane, the right integral of (2.18) is null, so combining with (2.11b), (2.18) can be rewritten as

$$M_S = r^2 b\int_{\theta_2}^{\theta_1} \sin\theta)[\tau_y(\theta) + f_B(\theta)]d\theta \tag{2.19}$$

At the end of the exposition of the current model, it is possible to state that equations (2.16), (2.17) and (2.19) constitute the main references of the wheel-soil interaction for a rigid wheel on soft soil.

2.2.3 Parameter estimation

The measurement and interpretation of the spatio-temporal variability of soils and vegetation is fundamental to the application of precision agriculture. The monitoring of such variations can be done through the acquisition of environmental data by remote and/or proximal sensors placed on the machines. In order to keep the reliability of vehicles in the terrain high, it would be ideal to be able to predict the characteristics of the surrounding environment with a high degree of accuracy so that the entire system can be planned and controlled quickly.

In lasts years many methods have been studied to determine soil parameters. Ding et al. [28] classified the soil parameters into three separate sets and then decomposed the wheel-soil model to solve them sequentially. This type of approach has also been experimentally validated by Xia et al. [29]. Hutangkabodee et al. [30], [31] adopted the composite Simpson's rule to fit the integrals in the model and then Newton-Raphson method to find solutions of approximated nonlinear equations set numerically. As these methods are very complex in terms of calculations, Iagnemma et al. [32] used an online estimation approach. However, the success of the method used depends mainly on the quality of the assumptions. With the aim of estimating N and C_σ, J.Y. Wong [33] proposed the plate sinkage test, and for c, ϕ and K the shear test. Thus, starting from measurable quantities such as F_{DP}, M_D and z_1, it is possible to estimate the terramechanical parameters.

Real-time dominant parameters estimation method

In order to safeguard energy expenditure and calculation complexity, a real-time estimation strategy is proposed by [34] concerning only a few focal parameters, the so-called dominant parameters: the sinkage exponent n and the internal friction angle ϕ. The former dominates normal stress, while the latter determines the relationship between shear stress and normal stress, which are the two main forces in the soil-wheel interaction. These two parameters were chosen because they are the most sensitive and dominate the other parameters, which can be set with empirical values. In this way, it is possible to have a good estimate, greatly easing the complexity of on-board calculations. The algorithm used to estimate the two parameters in real-time must have the ability to follow external disturbances very quickly and, as the wheel-ground reference is empirically defined, must take account of modelling errors. Hence, it has to be able to tolerate errors and disturbances in order to keep the parameter estimates stable. This can be achieved by means of an error-tolerant switched robust extended Kalman filter (ETS-REKF). The REKF itself has the inherent ability to tolerate errors due to external disturbances, but with the addition of the ETS the ability to tolerate modelling errors is added, allowing the filtering mode to be switched between robust and optimal. The whole structure is shown in Fig. 2.5 [34]. However, this method precludes the direct measurement of certain parameters such as force and torque directly from the wheel.

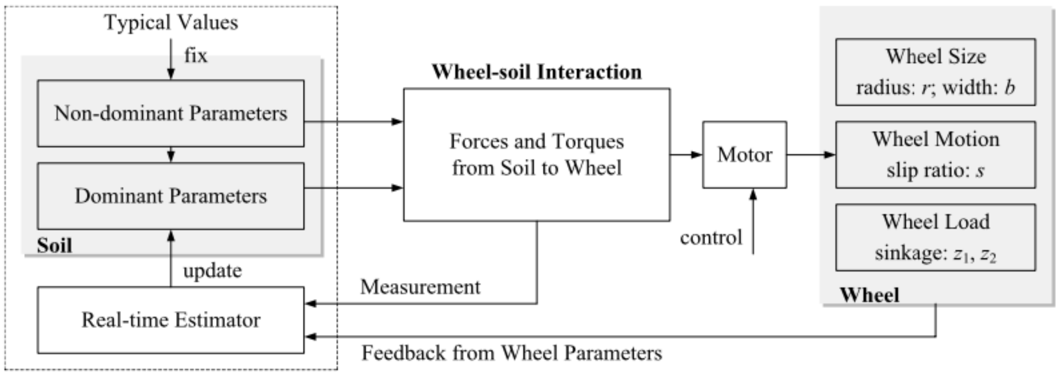
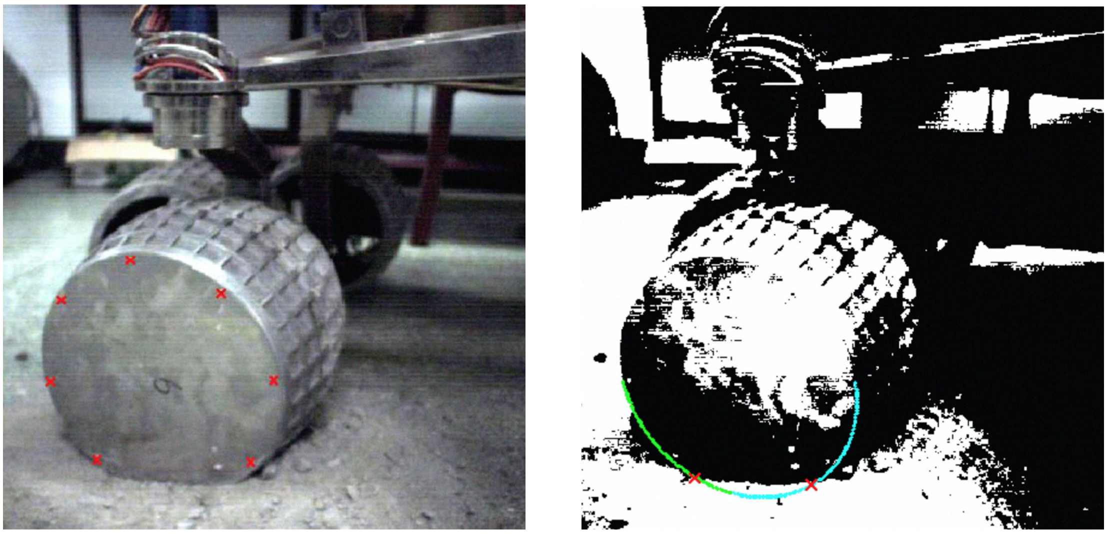


Figure 2.5: Structure of the real-time dominant parameters estimation method.

Sinkage visual measurement based on homography

An important parameter to estimate accurately is wheel sinkage, as several important capabilities depend on it, such as being able to adjust torque to improve traction or identify terrain. Based on the method proposed by Reina er al. [35], which included a camera to estimate wheel sinkage, but which, due to some image transformations led to deformations of the images, thus producing errors in the system, L. Wang et al. [36] show how it is possible to exploit homography for this purpose. The homography is obtained by means of an image of the wheel to which a number of reference points have been attached, thus finally obtaining the sinking of the wheel in 3D space Fig. 2.6. This method is more accurate and cost-effective than those used in the past, as the sinking of the wheel can be calculated directly from the camera, without additional sensors.

The method in question involves, in order to obtain the sinking of the wheel, firstly determining the homography between the plane of the wheel and that of the camera image and then analysing the variations in intensity of the image using a 1D spatial filter. This kind of filter transforms the original image into a binary one where all points corresponding to the wheel will be equivalent to 1 and all others to 0. Finding then the left p_l and right p_r contact points when the filter output reaches its minimum by sliding along the lower quadrants as shown in Fig. 2.7. In



(a) Captured image for measurement. (b) Contact point detection.

Figure 2.6: Data acquisition method.

order to make the process faster, in fact, the circle of the wheel has been divided into two portions, upper and lower, and only the latter will be analysed.

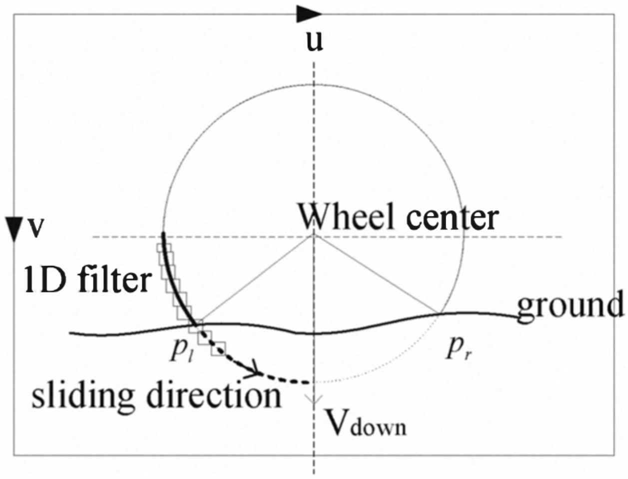


Figure 2.7: 1D filter sliding.

In the case analysed, however, the wheels are rigid and of a single colour well defined, so inaccuracies may arise if these features are missing. In addition, further problems with the display of the wheel rim could occur if the camera was held fixed and the wheel steered.

All-wheel-drive vehicles speed estimation

Obtaining an accurate estimate of vehicle's speed is an important achievement since, in this way, it is possible to improve the slip rate and, consequently, provide adequate torque to the drive wheels. In two-wheel drive vehicles, this is done by relatively simple methods based on the free-rolling wheels. This is clearly not possible in all-wheel-drive (AWD) vehicles as all wheels are subject to constant torque application. In order to overcome this problem, other solutions have been adopted such as using sensors already on board or using other systems such as GPS.

Velocity estimation using GPS

In [37], the global positioning system (GPS) is used to calculate vehicle speed, as well as wheel slip and vehicle slip angle. Wheel slip is measured, as usual, by making the difference between the speed measured by the GPS and the one measured in the wheel. In computing the lateral slip angle, the direction of travel can be determined from the GPS speed, while the vehicle heading can be obtained by integrating the gyroscope yaw. If the steering angle is not available, the yaw rate measured by the gyroscope can be used as input to a Kalman filter to estimate the vehicle heading and gyro bias.

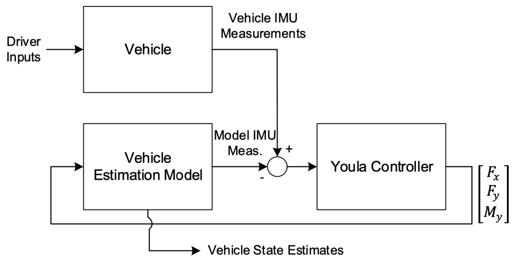
Using two GPS antennas instead of one makes calculations easier. The slip angle in fact becomes the difference between the GPS velocity vector of the first antenna and the GPS measurement of the second antenna's heading. In this way, errors in direction estimation caused by the integration of data received from the gyroscope are also eliminated.

Problems from detecting speed and other components using the GPS signal could arise if a high amount of weather-related noise occurs. In addition, it may happen that the signal is too low and therefore the speed estimate is unreliable or completely unrealistic.

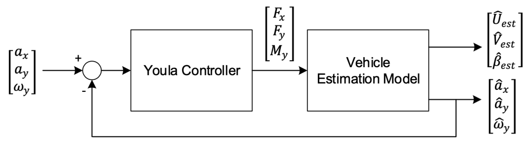
Youla controller output observer

Alcantar et al. in [38] show how, by exploiting the inertial measurement unit (IMU), it is possible to take advantage of the yaw rate measurements and accelerometer. The latter, in particular, provides longitudinal and lateral acceleration from which the lateral slip angle could be calculated and, in turn, can be used to contribute to the stability control system. In 2.8a, the entire proposed system.

In Fig. 2.8b, instead, is shown the equivalent feedback control problem where the input are the IMU measurements, such as longitudinal acceleration a_x, lateral acceleration a_y and yaw rate ω_y and the correspondent outputs are the longitudinal speed $\hat{U}_{est}$, the lateral speed $\hat{V}_{est}$ and the lateral slip angle $\hat{\beta}_{est}$ estimates. The



(a) Vehicle state estimation concept.



(b) Equivalent feedback control problem.

Figure 2.8: Structure including Youla controller design.

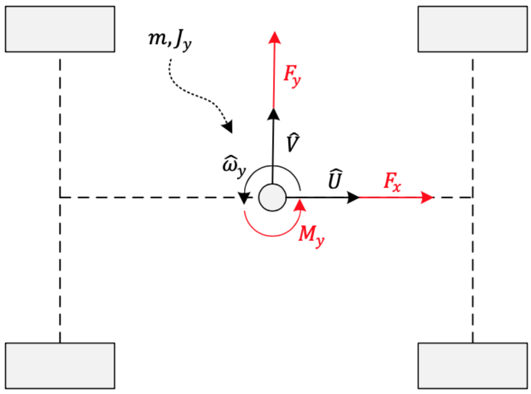


Figure 2.9: Vehicle model.

outputs of the Youla controller are the forces corresponding to the centre of gravity, longitudinal and lateral, and the yaw moment, F_x, F_y and M_y respectively. These are the inputs for the vehicle estimation model, shown in Fig. 2.9 and, given the vehicle mass m and inertia yaw moment j_y, the desired data can be extrapolated.

Chapter 3

Vineyard row path following

3.1 Row following problem statement

The yield of a grape plant depends on various physical, chemical and biological factors such as climate, soil properties, geographical location, pest and disease infestation, etc. In order to monitor the condition of the vines in relation to the events occurring during the growing and harvesting season, it is of fundamental importance to have an appropriate analysis. Over the years, this role has been dispensed with by man manually and this process is highly labor-intensive and wasteful in terms of time, requires well-defined competency and can be subject to the operator's personal judgment. For this reason, it is necessary to improve the process by implementing the use of autonomous vehicles. One of the most impactful technology areas in the agricultural sector is undoubtedly robotics, especially autonomous robotics for targeted work on plants, as well as robot-driven sensor platforms. Robots are in fact used to reduce the human workforce element in the various stages of working the soil, from preparation and sowing to harvesting, using driverless tractors. For these reasons, the development of an autonomous robot capable of inspecting vineyards, able to recognise the surrounding environment and, in particular, follow the row of vines without hindrance in order to carry out the analysis, may be useful.

Let us assume that we have a vineyard consisting of several rows and that each side of each row needs to be analysed. In this scenario, we formulate the following problem: developing a control algorithm for a 4WD mobile robot that allows the latter to visit each row at a predetermined distance.

3.2 Proposed solution

The solution proposed in this paper is based on two main ROS nodes and an Intel RealSense D435. Several useful data are acquired from the latter, including point clouds which are extrapolated and filtered by the first script. Using the RANdom SAmple Consensus (RANSAC) robust regression algorithm, the most popular technique to date for extracting single planar patches from noisy datasets containing multiple surfaces, the best-fitting plane is calculated. Both scripts are either subscriber and publisher, so in this case a vector containing data referring to the normal of the recognised plane is published. The second script, subscribed to the topic of the first, the data received is processed and, depending on the position of the robot with respect to the plane, the action to be performed by the robot (move away, approach or rotate) is selected and sent.

3.3 Development of the proposed solution

3.3.1 Solution design by means of ROS/Gazebo simulation framework

A small model of the robot was made in order to emulate the movements within the simulation environment. The components of the robot are described in a .xacro

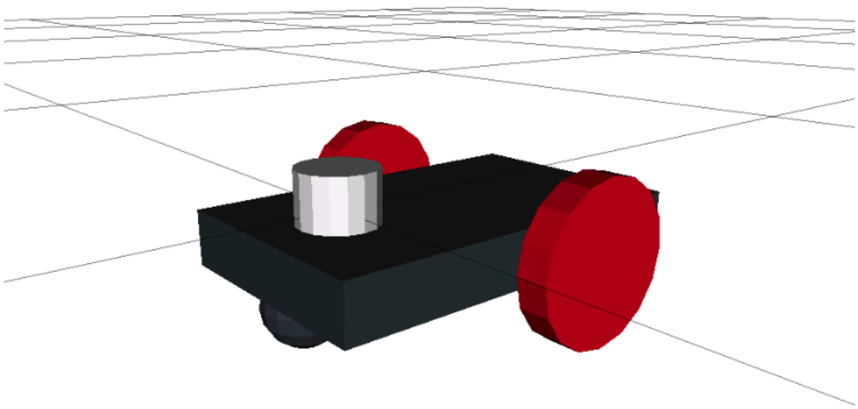
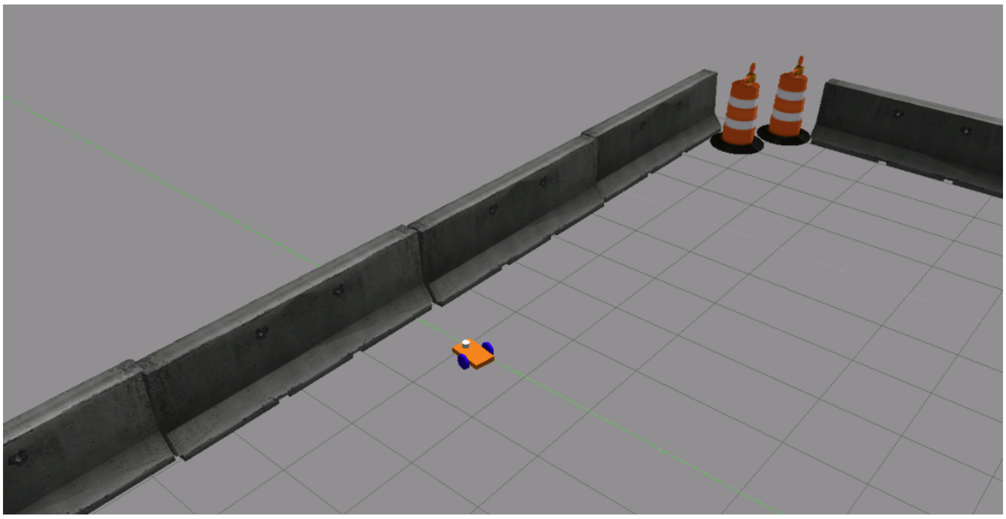
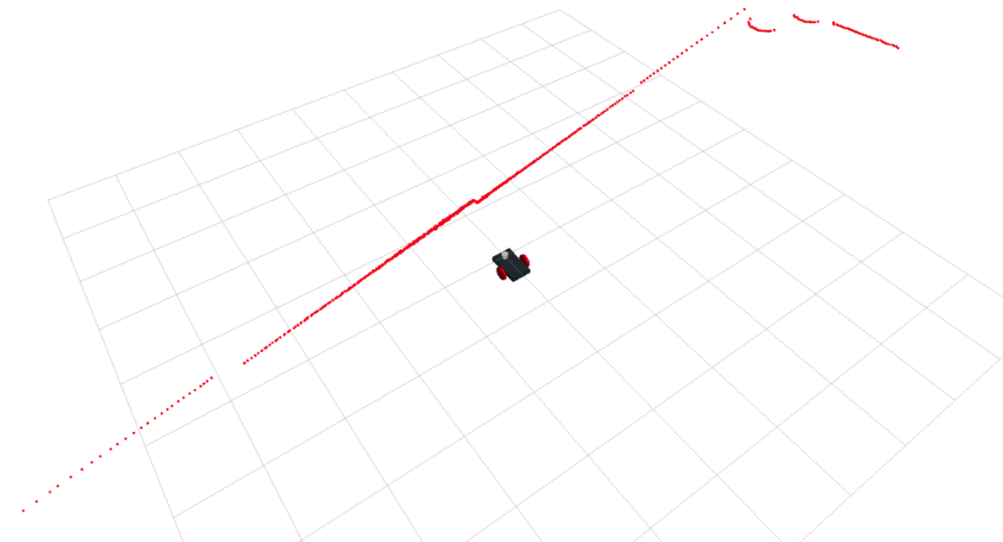


Figure 3.1: Two-wheel-drive robot model with laser scan.

design file (see Appendix A). As shown in Fig. 3.1, it is a two-wheel-drive robot with a free-rolling central sphere. Protruding at the front centre, it is possible to observe the laser scan. The latter is made real and functional by the Gazebo plug-in called "*gazebo_ros_head_hokuyo_controller*", at an angle of 180°. In Fig.



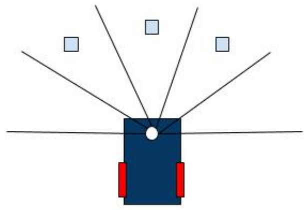
(a) Robot pose in Gazebo.



(b) Point cloud visualization in Rviz.

Figure 3.2: The figure in **a** shows the robot inside the Gazebo simulator, while **b** shows the view obtained on Rviz of the laser scan point cloud.

3.2b is shown, with reference to Fig. 3.2a, how the robot perceives the point cloud by means of laser scanning. After setting up the model, motion controls were developed using a python script (see Appendix B.1) in order to maintain a roughly stable distance from the wall. The scanning area is divided into five smaller areas and, for simplicity's sake, three of these are used to define the various detection



	Front Left	Front	Front Right
Case 1			
Case 2		x	
Case 3			x
Case 4	x		
Case 5		x	x
Case 6	x	x	
Case 7	x	x	x
Case 8	x		x

Figure 3.3: Laser detection cases.

cases: *Front-Left, Front* and *Front-Right* (Fig. 3.3). There are then eight possible combinations and, depending on the case, a decision is made. In this way, the robot will follow the wall as long as it is present, while if it finds itself in a situation where no wall is present nearby, it will start looking for one to follow.

3.3.2 Implementation of the proposed solution

Matlab data processing

A first setup, shown in Fig. 3.4, consisting of an Intel® RealSense™ D435 depth camera, used to scan a real wall in order to obtain the point cloud, an Intel® RealSense™ T265 tracking camera, used to track the robot odometry, and other sensors was set up in order to perform short acquisitions to collect essential data for algorithm development.

After acquiring the bag using ROS, the point cloud was then reconstructed using MATLAB to calculate the distance of the robot from the wall. The reconstructed point cloud is shown in Fig. 3.5a and the best fitting plane reconstructed on the point cloud using the *pcfitplane* (see Appendix B.2) function is shown in Fig. 3.5b.

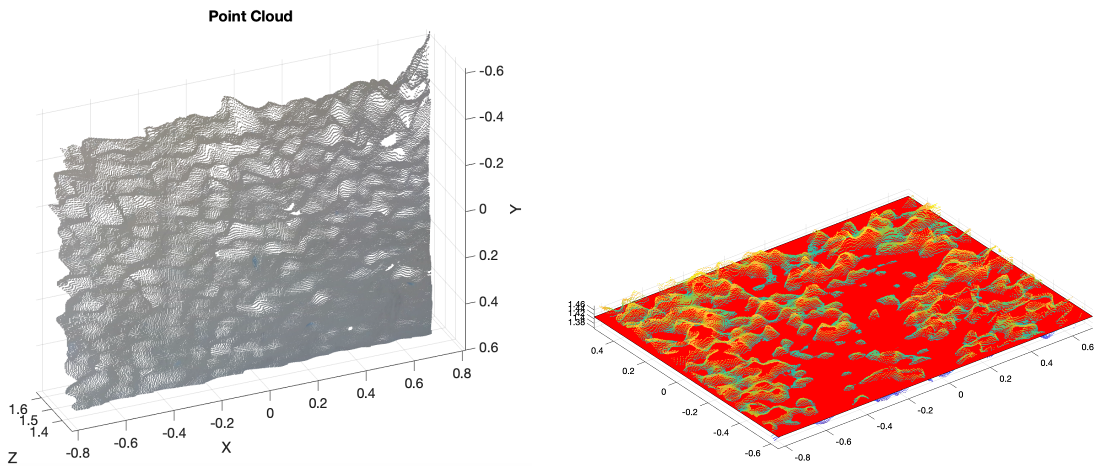
In addition, during the calculation of the best fitting plane, instant by instant, the distance of the plane from the robot was calculated, shown in Fig. 3.6 where it is also possible to notice some poorly defined peaks due to some glass windows

Figure 3.4: The figure shows the initial setup consisting of a mobile cart with sensors: 1) Intel® RealSense™ D435 depth camera, 2) Intel® RealSense™ T265 tracking camera, 3) GPS, 4) IMU.

along the wall. Fig. 3.6 also shows the odometry of the x-axis (oriented from the robot towards the wall) of the robot.

Online plane detection

Obtaining a fitted plane to the point cloud very quickly is a crucial step for the robot's performance as the used camera sends data at 30fps. A python script was implemented for this purpose, but the computation speed was low and as a consequence only a maximum of 10fps could be achieved for the plane determination. Because of this, the whole algorithm was rewritten in C++, increasing performance considerably by a factor of three. In Appendix B.3 the entire algorithm is given. The first operation performed is the conversion from PointCloud2 to PointCloud, which is required to perform the next steps, including the use of the plane recognition function. The point cloud is then filtered, limiting its height and depth to realistic



(a) Flat wall. **(b)** Point cloud fitted plane.

Figure 3.5: Point cloud reconstruction.

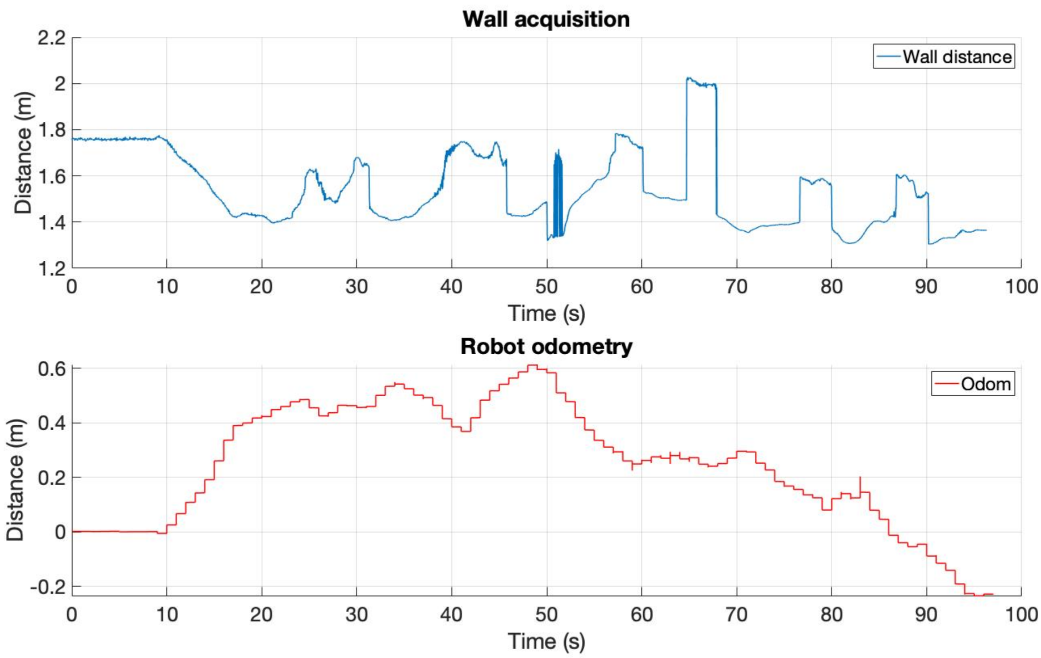


Figure 3.6: Wall distance and robot odometry over the acquisition time.

estimated values: $[0.1, 1]$ for the z axis and $[-0.6, 0.3]$ for the y axis (considering the camera in a horizontal position) [39]. This procedure allows us to obtain all the information about the plane normal with respect to the robot, and thus distance and inclination. The data obtained is then sent, via a suitable ROS topic, to the motion control algorithm.

Motion control

The purpose of the *Motion Control* algorithm, shown in Appendix B.4, is to receive the vector containing the data about the plane normal values and to analyse them. The model is based on predefined distances and angles from the plane, respectively *d_des* (distance of the robot from the plane) and *nx_des* (x component of the normal). While the former is expressed in metres and can be set to a preferred value with a certain tolerance *d_in*, the latter has a chosen value of 0, with tolerance *nx_in*, since if the component on the x-axis is zero, the robot is perpendicular to the estimated plane. In both cases, a hysteresis allowance has been provided, so that the robot does not remain in the boundary edge. Thus *d_out* for the distance and *nx_out* for the angle indicate the maximum amounts, added to the desired value, that can be tolerated beyond which the robot's realignment logic comes into play. After analysing the data, the Motion Controller can then determine to perform various actions, namely: approaching or moving away from the wall if it is too far or too close, or clockwise or anticlockwise rotation if the x-component of the normal is negative or positive. Once the action has been determined, the command is sent to the next node, the *Control Joy*.

Control joy

The *Control Joy* algorithm was developed in an initial phase so that the robot could be controlled exclusively by the Controller shown in the Fig. 3.7, and then the automatic function was implemented, managed by the *Motion Control* algorithm.

Figure 3.7: Robot remote control.

The first action to take, after starting all the scripts necessary for operation, to take control of the robot is to press the left and right triggers simultaneously and then the A button. This is a safety check, designed to prevent unintentional commands being given inadvertently. The robot is then ready to be controlled, and depending on the various key combinations, a different mode can be chosen. In each mode the left and right triggers are used to move forward and backward respectively. The default mode is *two-wheel steering*, in which it is possible to go forwards and backwards by steering with the front wheels via the right Thumbstick. To switch to *side steering* mode (Fig. 3.8a), use the Directional Pad, which rotates all four wheels ninety degrees, making lateral movement possible. Holding down the right Thumbstick switches to *circle steering* mode (Fig. 3.8b) and the left and right triggers are then used to rotate clockwise and counterclockwise respectively. The last mode is *four-wheel steering*, for which the left Thumbstick is used. In this mode, the front and rear wheels rotate discordantly to reduce the steering angle. This first part of the code had already been developed and tested by my colleague, PhD student Fabio Vulpi.

(a) Side steering mode. **(b)** Circle steering mode.

Figure 3.8: This image shows the two main modes used in the proposed solution.

The automatic mode is only activated when the 'Right Bumper' is pressed. In this way the commands of the remote controller are inhibited, with the exception of the button to switch the automatic mode on/off, and only the instructions received via the topic *cmd_vel* from the Motion Control node are taken into account. Various safety protocols are applied to preserve the status of electric motors. In fact, it has been made impossible to drive the robot when it is in the process of changing modes and the wheels have not completed the transition, or the SecureVelShutdown() function is used so that the motors do not stop instantly when the acceleration command stops, but gradually.

Odometry

Odometry is a localisation method which uses information from sensors such as encoders to derive an estimated position relative to the point of origin. It is particularly useful in autonomous vehicles as it facilitates certain tasks in the field thanks to the intrinsic knowledge of the current position. The pose, body position, is composed of two entities, i.e. position and direction of the robot. It can therefore be represented by x, y and θ coordinates. In order to determine the current robot position and thus update its pose, the variation must be computed using the sensor readings. In our case, 1024-bit encoders were used to control the rotation of the wheel. As a 1:4 gearbox is placed between the motor and the wheel, the final calculation to obtain the encoder resolution is $Enc_{res} = \frac{2\pi}{4096} = 0{,}0015\ rad/bit$. Knowing also the steering angle, fixed at 30 degrees, it is possible to calculate the centre of instantaneous rotation. The steering angle is fixed at 30 degrees, which allows the centre of instantaneous rotation to be calculated. In addition, since the circumference of the wheel is a known dimension, this makes it possible to calculate the distances travelled by using the encoder ticks. The calculations and procedures are shown in detail in Appendix B.6.

However, incremental encoders on wheels measure their rotation, but not the robot's attitude in a fixed absolute reference system. If a wheel slips, the encoder detects a displacement that is not consistent with the change in attitude of the robot. For this reason it is important to combine the odometry measurement with a second measurement, such as GPS or visual odometry, in order to calculate the amount of slippage and refine the positioning of the robot in space.

Chapter 4

Tests and experiments

This chapter describes the experimental tests carried out through which data necessary for the production of improvements was collected. In detail, the first section describes the indoor experiments, in which paths and obstacles were simulated, while the second section describes the outdoor experimentation campaign.

Figure 4.1: Robot raised on stands.

4.1 Indoor tests

Indoor tests took place during the entire node design period in order to find and fix any bugs or problems. At first, the robot was kept elevated on tripods (Fig. 4.1) to test the model and exclude any possibility of damage to the robot.

In order to test the implemented functionalities and the decision-making capacity of the Motion Control algorithm, a first test was carried out by simulating the distance and inclination variations of the wall as shown in Fig. 4.2.

Figure 4.2: The picture depicts the robot, raised, with the Intel® RealSense™ D435 camera mounted on top to identify the orientation of the moving panel.

When good results were obtained, we moved on to testing the robot in controlled environments specially composed to provide for the activation of every possible mode that could be selected by the control algorithm. Fig. 4.3 shows the robot that independently follows the wall and then, by rotating, adapts to the different inclinations of the pathway.

4.2 Outdoor tests

The outdoor tests were carried out in San Cassiano (LE) and they lasted 3 days. The first day was taken up with the final configuration of the robot and the assembly of the final setup, depicted in Fig. 4.4 on the robot body composed of five cameras, GPS, IMU and two additional Intel® NUCs connected to each other and to the central unit of the robot via LAN. The remaining two days were spent collecting data and conducting experiments among the vineyard rows. Unfortunately, as can

Figure 4.3: Robot following the path.

Figure 4.4: Final robot setup for outdoor acquisitions.

be seen in Fig. 4.5 it was not possible to activate the robot's automatic control algorithm because the soil was too soft and, since the wheels were too small and thin for the type of soil, the robot was not able to move to align itself with a sufficiently high accuracy. However, it was possible to test the plane recognition software (Fig. 4.6) and a lot of useful data was acquired for offline analysis and development of future implementations.

Figure 4.5: Final robot setup for outdoor acquisitions.

Moreover, the outdoor tests have shown the need to filter the data for online decision analysis.

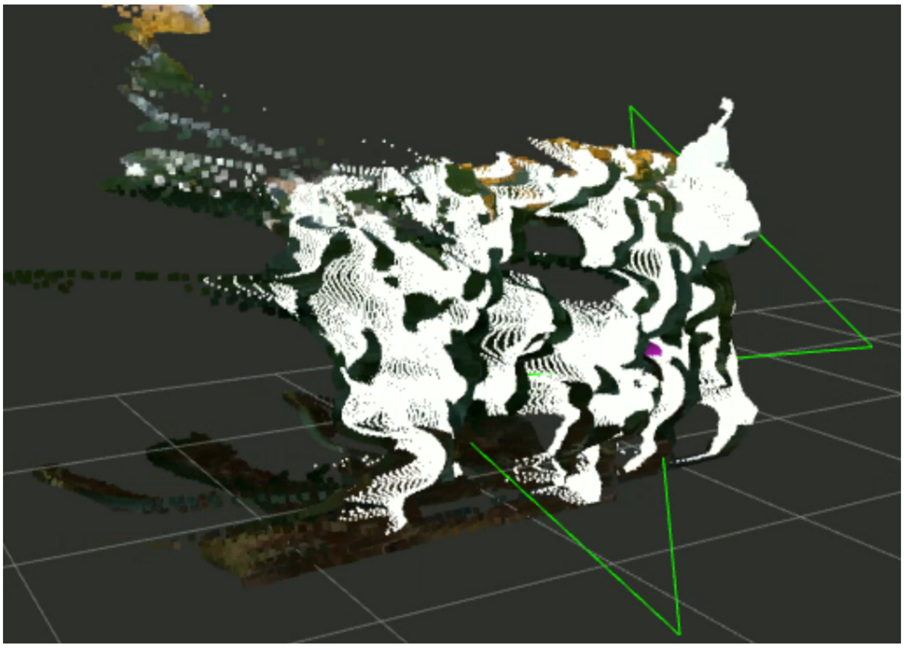


Figure 4.6: Pointcloud visualisation of the vineyard row on Rviz. In white the filtered points, taken into account, while the green bow tie shows the estimated plan, the purple ball is the centre of it.

4.3 Data analysis

From the data acquired during the experimental tests, especially the outdoor tests, it emerged that the high reactivity and therefore high oscillation of these, can result in the selection of incorrect robot motion modes and poor accuracy and reliability. For this reason, a Kalman Filter was implemented in order to smooth the data.

Starting with the state equations,

$$\dot{\theta}_{k+1} = \omega_{\dot{\theta}} \tag{4.1a}$$

$$\theta_{k+1} = \theta_k + \dot{\theta}\Delta t + \omega_\theta \tag{4.1b}$$

$$\dot{d}_{k+1} = \omega_{\dot{d}} \tag{4.1c}$$

$$d_{k+1} = d_k + \dot{d}\Delta t + \omega d \tag{4.1d}$$

where θ is the inclination of the robot with respect to the row and d the distance, the following matrices were obtained:

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 \\ \Delta t & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \Delta t & 1 \end{bmatrix} \tag{4.2a}$$

$$B = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{4.2b}$$

$$C = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.2c}$$

$$D = 0 \tag{4.2d}$$

The matrices A B C D were obtained using a linear time invariant model (LTI). Thus, using the Kalman equations the Kalman Filter model was developed using Matlab (see Appendix B.7).

$$x_{k+1|k} = A * x_{k|k} \tag{4.3a}$$

$$P_{k+1|k} = A * P_{k|k} * A' + Q \tag{4.3b}$$

$$K = P_{k+1|k} * C' * inv(C * P_{k+1|k} * C' + R) \tag{4.3c}$$

$$x_{k+1|k+1} = x_{k+1|k} + K * (y - C * x_{k+1|k}) \tag{4.3d}$$

$$P_{k+1|k+1} = I - K * C) * P_{k+1|k} \tag{4.3e}$$

$$y_{k+1|k+1} = (C * x_{k+1|k+1} \tag{4.3f}$$

$$x_{k|k} = x_{k+1|k+1} \tag{4.3g}$$

$$P_{k|k} = P_{k+1|k+1} \tag{4.3h}$$

In the above equations, x and y are the input and output respectively, P is the covariance matrix (a measure of the estimated accuracy of the estimated state) and K is the gain that minimises the residual error.

The implementation of a Kalman filter may often be challenging due to the complexity of obtaining a good estimate of the noise covariance matrices Q and R. The first depends on the sensitivity of the sensor while the second is the covariance of the process noise and after numerous refinement tests, good results were obtained by setting R $= \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}$ and Q $= \begin{bmatrix} 0.1 & 0 & 0 & 0 \\ 0 & 0.1 & 0 & 0 \\ 0 & 0 & 0.1 & 0 \\ 0 & 0 & 0 & 0.1 \end{bmatrix}$ we observe the result in Fig. 4.7 which shows a smoother and less angular curve. In this way it is possible to avoid high peak surges and instead maintain a more linear trajectory. This allows the robot to be more accurate and travel strategies more efficient.

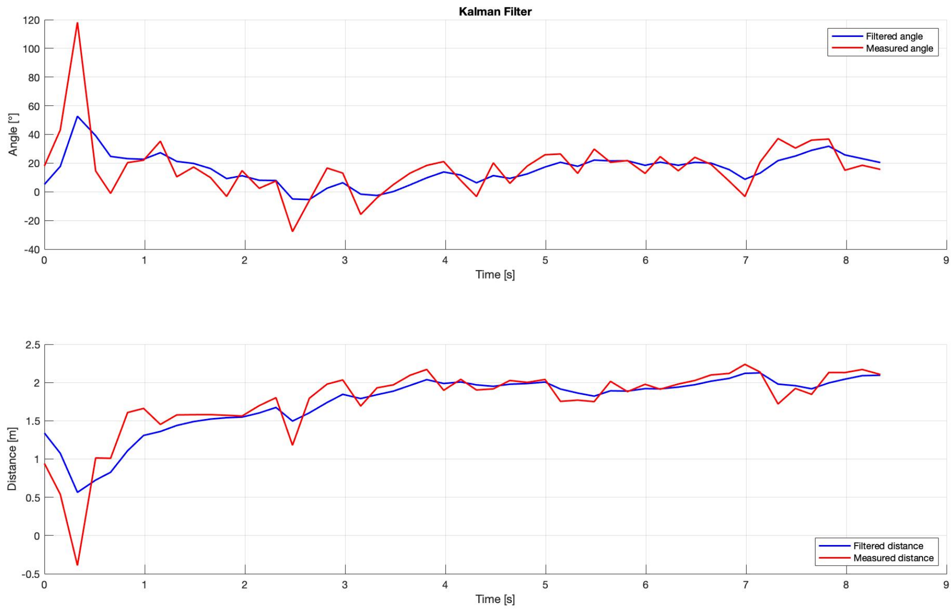


Figure 4.7: Comparison of measured and Kalman Filtered angle and distance.

Chapter 5

Conclusions

5.1 Achievements

In conclusion, it is therefore possible to say that depth cameras can be of great help in precision farming and can be an inexpensive but valuable means of developing effective products. The elaborated model, based on the detection of the plane with respect to the received point cloud, proved to be valid and can represent an excellent starting point for future implementations. Thanks to both indoor and outdoor tests, the objectives set were validated, in particular the addition of the Kalman Filter for data processing proved to be of great impact and necessary to improve the decision-making process and thus increase the accuracy, reliability and repeatability of the robot.

5.2 Future work

Future work could be divided into three main tasks:

- Perception improvement;
- Autonomous phenotyping;
- Cooperative analysis.

One of the goals of perception that should be aimed at is therefore to improve the control part that concerns the interaction of the wheel on the ground. Autonomous phenotyping would make the process more efficient, the use of autonomous robots would then automate the data collection and increase the temporal resolution of the growth status by precisely highlighting any phytosanitary problems. Depending on the type of crop, it may be useful or necessary to approach field inspections with different methodologies. In some circumstances an area-based inspection would be indispensable, while in other environments a co-operative analysis (e.g. robot on wheels and drone) might be useful, resulting in a multi-layer map with a more defined amount of information. In order to achieve the set objectives, the research activity proposes to start from the study of the state of the art to examine in detail the aspects related to the application of complex systems in the context of agriculture.

A methodology that can be implemented in the future could involve the use of methods based on neural networks with the aim of recognising the type of plant, the state of ripeness of the fruit, any problems related to its state of health and any additional information for a complete analysis. As reported in [40] and [41], the deep neural network has proven to be a particularly effective and fruitful technique, it gives the possibility to manage and classify data with a high adaptability. Further studies also show that the use of neural networks is applicable to the recognition of specific plants and their possible diseases.

The use of neural networks in agriculture, specifically in trait control, phenotyping and contiguous operation, certainly has enormous advantages. In particular, one would have a large amount of detailed and accurate data in a short period of time, thus regulating the use of fertilisers, drugs, pesticides and water dosage in a specific way thus avoiding waste and drastically reducing the pollution produced. Traction control directed according to the type of terrain or obstacle to be tackled would allow greater reliability of agricultural vehicles and greater autonomy. The latter could find applications in aerospace rovers, where the ability not to get stuck is crucial.

Appendix A

Gazebo model

A.1 Main part

```
<?xml version="1.0" ?>
<robot name="m2wr" xmlns:xacro="http://www.ros.org/wiki/xacro">

  <xacro:include filename="$(find m2wr_description)/urdf/materials.
    xacro" />
  <xacro:include filename="$(find m2wr_description)/urdf/m2wr.gazebo"
     />
  <xacro:include filename="$(find m2wr_description)/urdf/macros.xacro
    " />

  <link name="link_chassis">
    <!-- pose and inertial -->
    <pose>0 0 0.1 0 0 0</pose>
    <inertial>
      <mass value="5"/>
      <origin rpy="0 0 0" xyz="0 0 0.1"/>
      <inertia ixx="0.0395416666667" ixy="0" ixz="0" iyy="
    0.106208333333" iyz="0" izz="0.106208333333"/>
    </inertial>
    <!-- body -->
    <collision name="collision_chassis">
      <geometry>
        <box size="0.5 0.3 0.07"/>
      </geometry>
    </collision>
    <visual>
      <origin rpy="0 0 0" xyz="0 0 0"/>
      <geometry>
        <box size="0.5 0.3 0.07"/>
```

```
      </geometry>
      <material name="blue"/>
    </visual>
    <!-- caster front -->
    <collision name="caster_front_collision">
      <origin rpy=" 0 0 0" xyz="0.35 0 -0.05"/>
      <geometry>
        <sphere radius="0.05"/>
      </geometry>
      <surface>
        <friction>
          <ode>
            <mu>0</mu>
            <mu2>0</mu2>
            <slip1>1.0</slip1>
            <slip2>1.0</slip2>
          </ode>
        </friction>
      </surface>
    </collision>
    <visual name="caster_front_visual">
      <origin rpy=" 0 0 0" xyz="0.2 0 -0.05"/>
      <geometry>
        <sphere radius="0.05"/>
      </geometry>
    </visual>
  </link>

  <link name="sensor_laser">
    <inertial>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <mass value="1" />
      <xacro:cylinder_inertia mass="1" r="0.05" l="0.1" />
    </inertial>

    <visual>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <cylinder radius="0.05" length="0.1"/>
      </geometry>
      <material name="white" />
    </visual>

    <collision>
      <origin xyz="0 0 0" rpy="0 0 0"/>
      <geometry>
        <cylinder radius="0.05" length="0.1"/>
      </geometry>
    </collision>
```

```
  </link>

  <joint name="joint_sensor_laser" type="fixed">
    <origin xyz="0.15 0 0.05" rpy="0 0 0"/>
    <parent link="link_chassis"/>
    <child link="sensor_laser"/>
  </joint>

  <xacro:link_wheel name="link_right_wheel" />
  <xacro:joint_wheel name="joint_right_wheel" child="link_right_wheel
    " origin_xyz="-0.05 0.20 0" />

  <xacro:link_wheel name="link_left_wheel" />
  <xacro:joint_wheel name="joint_left_wheel" child="link_left_wheel"
    origin_xyz="-0.05 -0.20 0" />
</robot>
```

A.2 Macros

```
<?xml version="1.0"?>
<robot xmlns:xacro="http://www.ros.org/wiki/xacro">
    <xacro:macro name="link_wheel" params="name">
        <link name="${name}">
            <inertial>
              <mass value="0.2"/>
              <origin rpy="0 1.5707 1.5707" xyz="0 0 0"/>
              <inertia ixx="0.000526666666667" ixy="0" ixz="0" iyy="
    0.000526666666667" iyz="0" izz="0.001"/>
            </inertial>
            <collision name="link_right_wheel_collision">
              <origin rpy="0 1.5707 1.5707" xyz="0 0 0"/>
              <geometry>
                <cylinder length="0.04" radius="0.1"/>
              </geometry>
            </collision>
            <visual name="${name}_visual">
              <origin rpy="0 1.5707 1.5707" xyz="0 0 0"/>
              <geometry>
                <cylinder length="0.04" radius="0.1"/>
              </geometry>
            </visual>
        </link>
    </xacro:macro>

    <xacro:macro name="joint_wheel" params="name child origin_xyz">
```

```
    <joint name="${name}" type="continuous">
      <origin rpy="0 0 0" xyz="${origin_xyz}"/>
      <child link="${child}"/>
      <parent link="link_chassis"/>
      <axis rpy="0 0 0" xyz="0 1 0"/>
      <limit effort="10000" velocity="1000"/>
      <joint_properties damping="1.0" friction="1.0"/>
    </joint>
  </xacro:macro>

  <xacro:macro name="cylinder_inertia" params="mass r l">
    <inertia  ixx="${mass*(3*r*r+l*l)/12}" ixy = "0" ixz = "0"
              iyy="${mass*(3*r*r+l*l)/12}" iyz = "0"
              izz="${mass*(r*r)/2}" />
  </xacro:macro>
</robot>
```

A.3 Materials

```
<?xml version="1.0"?>
<robot xmlns:xacro="http://www.ros.org/wiki/xacro">
  <material name="black">
    <color rgba="0.0 0.0 0.0 1.0"/>
  </material>
  <material name="blue">
    <color rgba="0.203125 0.23828125 0.28515625 1.0"/>
  </material>
  <material name="green">
    <color rgba="0.0 0.8 0.0 1.0"/>
  </material>
  <material name="grey">
    <color rgba="0.2 0.2 0.2 1.0"/>
  </material>
  <material name="orange">
    <color rgba="1.0 0.423529411765 0.0392156862745 1.0"/>
  </material>
  <material name="brown">
    <color rgba="0.870588235294 0.811764705882 0.764705882353 1.0"/>
  </material>
  <material name="red">
    <color rgba="0.80078125 0.12890625 0.1328125 1.0"/>
  </material>
  <material name="white">
    <color rgba="1.0 1.0 1.0 1.0"/>
  </material>
```

```
</robot>
```

Appendix B

Algorithms

B.1 Wall following

```
#! /usr/bin/env python

import rospy
from sensor_msgs.msg import LaserScan
from geometry_msgs.msg import Twist
from nav_msgs.msg import Odometry
from tf import transformations

import math

pub_ = None
regions_ = {
    'right': 0,
    'fright': 0,
    'front': 0,
    'fleft': 0,
    'left': 0,
}
state_ = 0
state_dict_ = {
    0: 'find the wall',
    1: 'turn left',
    2: 'follow the wall',
}

def clbk_laser(msg):
    global regions_
    regions_ = {
        'right':  min(min(msg.ranges[0:143]), 10),
```

```
        'fright': min(min(msg.ranges[144:287]), 10),
        'front':  min(min(msg.ranges[288:431]), 10),
        'fleft':  min(min(msg.ranges[432:575]), 10),
        'left':   min(min(msg.ranges[576:713]), 10),
    }

    take_action()


def change_state(state):
    global state_, state_dict_
    if state is not state_:
        print 'Wall follower - [%s] - %s' % (state, state_dict_[state])
        state_ = state

def take_action():
    global regions_
    regions = regions_
    msg = Twist()
    linear_x = 0
    angular_z = 0

    state_description = ''

    d = 1.5

    if regions['front'] > d and regions['fleft'] > d and regions['fright'] > d:
        state_description = 'case 1 - nothing'
        change_state(0)
    elif regions['front'] < d and regions['fleft'] > d and regions['fright'] > d:
        state_description = 'case 2 - front'
        change_state(1)
    elif regions['front'] > d and regions['fleft'] > d and regions['fright'] < d:
        state_description = 'case 3 - fright'
        change_state(2)
    elif regions['front'] > d and regions['fleft'] < d and regions['fright'] > d:
        state_description = 'case 4 - fleft'
        change_state(0)
    elif regions['front'] < d and regions['fleft'] > d and regions['fright'] < d:
        state_description = 'case 5 - front and fright'
        change_state(1)
    elif regions['front'] < d and regions['fleft'] < d and regions['fright'] > d:
```

```
        state_description = 'case 6 - front and fleft'
        change_state(1)
    elif regions['front'] < d and regions['fleft'] < d and regions['fright'] < d:
        state_description = 'case 7 - front and fleft and fright'
        change_state(1)
    elif regions['front'] > d and regions['fleft'] < d and regions['fright'] < d:
        state_description = 'case 8 - fleft and fright'
        change_state(0)
    else:
        state_description = 'unknown case'
        rospy.loginfo(regions)

def find_wall():
    msg = Twist()
    msg.linear.x = 0.2
    msg.angular.z = 0.3
    return msg

def turn_left():
    msg = Twist()
    msg.angular.z = -0.3
    return msg

def follow_the_wall():
    global regions_
    msg = Twist()
    msg.linear.x = 0.3
    return msg

def main():
    global pub_
    rospy.init_node('reading_laser')
    pub_ = rospy.Publisher('/cmd_vel', Twist, queue_size=1)
    sub = rospy.Subscriber('/m2wr/laser/scan', LaserScan, clbk_laser)
    rate = rospy.Rate(20)
    while not rospy.is_shutdown():
        msg = Twist()
        if state_ == 0:
            msg = find_wall()
        elif state_ == 1:
            msg = turn_left()
        elif state_ == 2:
            msg = follow_the_wall()
            pass
        else:
            rospy.logerr('Unknown state!')

```

```
        pub_.publish(msg)

        rate.sleep()

if __name__ == '__main__':
    main()
```

B.2 MATLAB plane detection

```
clear all, close all, clc
load(strcat(folder,"/MATS/","bag.mat"))

%%

maxDistance = 0.10;
referenceVector = [0,0,1];
maxAngularDistance = 5;

wallDistStruct = cell(length(data.pointcloud),1);
for i = 1:length(data.pointcloud)
    ptCloud = pointCloud(data.pointcloud{i, 1}.XYZ);

    % Detect the first plane, the table, and extract it from the point cloud.
    [model1,inlierIndices,outlierIndices] = pcfitplane(ptCloud,...
                maxDistance,referenceVector,maxAngularDistance);
    plane1 = select(ptCloud,inlierIndices);
    remainPtCloud = select(ptCloud,outlierIndices);

    VOID_QUOTE =  double(mean(plane1.Location(:,3)));
            AVG_PlaneModel = planeModel([0 0 -1 VOID_QUOTE]);

    wallDistStruct{i,1}.distance = AVG_PlaneModel.Parameters(1,4);
    wallDistStruct{i,1}.stamp.sec = data.pointcloud{i, 1}.Header.Stamp.Sec;
    wallDistStruct{i,1}.stamp.Nsec = data.pointcloud{i, 1}.Header.Stamp.Nsec;
    i
end
%%

d = zeros(length(wallDistStruct),1);
t1 = zeros(length(wallDistStruct),1);

for i = 1:length(d)
```

```
    d(i,1) = wallDistStruct{i, 1}.distance ;
    t1(i,1) = wallDistStruct{i, 1}.stamp.sec + wallDistStruct{i, 1}.stamp.Nsec*10^(-9)...
        - wallDistStruct{1, 1}.stamp.sec - wallDistStruct{1, 1}.stamp.Nsec*10^(-9);
end

%%

bag = rosbag("bag.bag")

bSel = select(bag,'Topic','/camera/depth/color/points');
bSel1 = select(bag,'Topic','/t265/odom/sample');

msgStructs1 = readMessages(bSel1,'DataFormat','struct');

start = msgStructs1{1}.Header.Stamp.Sec;

for i = 1:length(msgStructs1)
    x(i) = msgStructs1{i}.Pose.Pose.Position.X;
    y(i) = msgStructs1{i}.Pose.Pose.Position.Y;
    z(i) = msgStructs1{i}.Pose.Pose.Position.Z;

    t(i) = msgStructs1{i}.Header.Stamp.Sec - start;
end
%%
close all

figure(1)
nexttile
set(gcf,'color','w');
set(gca,'fontsize',25)
grid on, hold on
plot(t1, d, 'LineWidth', 1.5)
title('Wall acquisition')
legend('Wall distance'), xlabel('Time (s)'), ylabel('Distance (m)')

figure(1)
nexttile
set(gcf,'color','w');
set(gca,'fontsize',25)
grid on, hold on
plot(t,x,'r','linewidth',1.5)
title('Robot odometry')
legend('Odom'), xlabel('Time (s)'), ylabel('Distance (m)')
```

B.3 Plane detection

```
#include "ros/ros.h"
#include "std_msgs/String.h"
#include <std_msgs/Float32MultiArray.h>

#include <motion_controller/wall.h>

#include "sensor_msgs/PointCloud2.h"
#include <iostream>
#include <fstream>
#include <sstream>

#include <pcl_conversions/pcl_conversions.h>
#include <pcl/point_types.h>
#include <pcl/point_cloud.h>
#include <pcl/PCLPointCloud2.h>
#include <pcl/conversions.h>
#include <pcl_ros/transforms.h>

#include <pcl/point_cloud.h>
#include <pcl/sample_consensus/ransac.h>
#include <pcl/sample_consensus/sac_model_plane.h>
#include <Eigen/Core>

#include <pcl/ModelCoefficients.h>
#include <pcl/io/pcd_io.h>
#include <pcl/sample_consensus/model_types.h>
#include <pcl/segmentation/sac_segmentation.h>
#include <pcl/filters/conditional_removal.h>

#include <math.h>

#include <chrono>
#include <ros/console.h>

#define threshold 0.01


class distSubPub
{
  public:
    distSubPub()
    {
      //publisher
      wall_pub_ = n_.advertise<motion_controller::wall>("/wallinfo/
  LatCam_frame", 1000);
```

```
    pc_pub_= n_.advertise<sensor_msgs::PointCloud2> ("/wallinfo/pc", 1000);
    //subscriber
    wall_sub_ = n_.subscribe<sensor_msgs::PointCloud2>("/LatCam/depth/color/points", 1, &distSubPub::pointcloudCallback, this);

 }
 void pointcloudCallback(const boost::shared_ptr<const sensor_msgs::PointCloud2>& input){

    pcl::PCLPointCloud2 pcl_pc2;
    pcl_conversions::toPCL(*input,pcl_pc2);
    pcl::PointCloud<pcl::PointXYZ>::Ptr temp_cloud(new pcl::PointCloud<pcl::PointXYZ>);
    pcl::fromPCLPointCloud2(pcl_pc2,*temp_cloud);

    // build the condition
    pcl::ConditionAnd<pcl::PointXYZ>::Ptr range_cond (new pcl::ConditionAnd<pcl::PointXYZ> ());
    range_cond->addComparison (pcl::FieldComparison<pcl::PointXYZ>::ConstPtr (new pcl::FieldComparison<pcl::PointXYZ> ("z", pcl::ComparisonOps::GT, 0.1)));
    range_cond->addComparison (pcl::FieldComparison<pcl::PointXYZ>::ConstPtr (new pcl::FieldComparison<pcl::PointXYZ> ("z", pcl::ComparisonOps::LT, 1.0)));
    range_cond->addComparison (pcl::FieldComparison<pcl::PointXYZ>::ConstPtr (new pcl::FieldComparison<pcl::PointXYZ> ("y", pcl::ComparisonOps::LT, 0.3)));
    range_cond->addComparison (pcl::FieldComparison<pcl::PointXYZ>::ConstPtr (new pcl::FieldComparison<pcl::PointXYZ> ("y", pcl::ComparisonOps::GT, -0.6)));

    // build the filter
    pcl::ConditionalRemoval<pcl::PointXYZ> condrem;
    condrem.setCondition (range_cond);
    condrem.setInputCloud (temp_cloud);
    condrem.setKeepOrganized (true);
    // apply filter
    condrem.filter (*temp_cloud);


    // Convert to ROS data type
    pcl::PCLPointCloud2 temp_cloud2;
    pcl::toPCLPointCloud2(*temp_cloud, temp_cloud2);
    sensor_msgs::PointCloud2 temp_cloud_filt;
    pcl_conversions::moveFromPCL(temp_cloud2, temp_cloud_filt);
    pc_pub_.publish (temp_cloud_filt);


```

```

      pcl::ModelCoefficients::Ptr coefficients (new pcl::ModelCoefficients);
      pcl::PointIndices::Ptr inliers (new pcl::PointIndices);
      // Create the segmentation object
      pcl::SACSegmentation<pcl::PointXYZ> seg;
      // Optional
      seg.setOptimizeCoefficients (true);
      // Mandatory
      seg.setModelType (pcl::SACMODEL_PLANE);
      seg.setMethodType (pcl::SAC_RANSAC);
      seg.setDistanceThreshold (threshold);
      seg.setInputCloud (temp_cloud);

      //auto start_time = std::chrono::high_resolution_clock::now();
      try {
        seg.segment (*inliers, *coefficients);
    motion_controller::wall wallmsg;
    wallmsg.header.stamp = ros::Time::now();
    wallmsg.header.frame_id = "/LatCam_depth_optical_frame";
    wallmsg.distance = -coefficients->values[3];
    wallmsg.normal_vector.push_back(coefficients->values[0]);
    wallmsg.normal_vector.push_back(coefficients->values[1]);
    wallmsg.normal_vector.push_back(coefficients->values[2]);
    wall_pub_.publish(wallmsg);
        //ROS_INFO_STREAM("try");
      }
      catch (...){
    ROS_INFO_STREAM("catch");
        if (inliers->indices.size () == 0)
        {
          //PCL_ERROR ("Could not estimate a planar model for the given dataset.");
      ROS_INFO_STREAM("no inl");
        }
      }

      //auto end_time = std::chrono::high_resolution_clock::now();
      //auto time = end_time - start_time;

  }

  private:
    ros::NodeHandle n_;
    ros::Publisher wall_pub_;
    ros::Publisher pc_pub_;
    ros::Subscriber wall_sub_;
};

```

```
int main(int argc, char **argv)
{
  ros::init(argc, argv, "distance_node");
  distSubPub dspo;
  ros::spin();

  return 0;
}
```

B.4 Motion control

```
#!/usr/bin/env python

import rospy
from motion_controller.msg import wall
import tf2_ros
import tf2_geometry_msgs
from geometry_msgs.msg import Point, PointStamped, Twist,
    PolygonStamped, Point32
from tf.transformations import quaternion_multiply
import numpy as np

d_flg = False
nx_flg = False

d_des = 0.6
d_in = 0.05
d_out = 0.1

nx_des = 0
nx_in = 0.05
nx_out = 0.15


class motionCommands(object):

    def __init__(self):
        self.vel_msg = Twist()
        self.vel_msg.linear.x = 0
        self.vel_msg.linear.y = 0
        self.vel_msg.linear.z = 0
        self.vel_msg.angular.x = 0
        self.vel_msg.angular.y = 0
        self.vel_msg.angular.z = 0

```

```
    def goStraight(self):
        self.vel_msg.linear.x = 0.2
        return self.vel_msg

    def rightRot(self):
        self.vel_msg.angular.z = -0.2
        return self.vel_msg

    def leftRot(self):
        self.vel_msg.angular.z = 0.2
        return self.vel_msg

    def goFurther(self):
        self.vel_msg.linear.y = 0.2
        return self.vel_msg

    def goCloser(self):
        self.vel_msg.linear.y = -0.2
        return self.vel_msg

def motionCallback(data):
    global vel_pub
    global d_flg
    global nx_flg
    global transf
    global point_pub
    global poly_pub
    global d_des
    global d_in
    global d_out

    global nx_des
    global nx_in
    global nx_out
    mC = motionCommands()

    p_cam = [data.distance*data.normal_vector[0],data.distance*data.normal_vector[1],data.distance*data.normal_vector[2]]

    p_base = PointStamped(point=(tf2_geometry_msgs.do_transform_point(PointStamped(point=Point(p_cam[0], p_cam[1], p_cam[2])), transf).point))
    p_base.header.stamp = rospy.Time.now()
    p_base.header.frame_id = "base_link"
    point_pub.publish(p_base)

    p = np.array([p_base.point.x,p_base.point.y,p_base.point.z])
    d = np.linalg.norm(p)

```

```
    nq0 = [data.normal_vector[0], data.normal_vector[1], data.normal_vector[2], 0]
    transq = [transf.transform.rotation.x,transf.transform.rotation.y,transf.transform.rotation.z,transf.transform.rotation.w]
    nq1 = quaternion_multiply(quaternion_multiply(transq, nq0),[-transq[0],-transq[1],-transq[2],transq[3]])
    rospy.loginfo(nq1)
    nx = nq1[0]

    n_base = np.array([nq1[0],nq1[1],nq1[2]])
    u = np.array([-n_base[1],n_base[0],0])/np.linalg.norm(np.array([-n_base[1],n_base[0],0]))
    v = np.cross(n_base,u)
    v = v/np.linalg.norm(v)
    poly = PolygonStamped()
    poly.header.stamp = rospy.Time.now()
    poly.header.frame_id = "base_link"
    poly.polygon.points = [Point32(x=p[0]-u[0], y=p[1]-u[1], z=p[2]-u[2]),
                           Point32(x=p[0]+u[0], y=p[1]+u[1], z=p[2]+u[2]),
                           Point32(x=p[0]-v[0], y=p[1]-v[1], z=p[2]-v[2]),
                           Point32(x=p[0]+v[0], y=p[1]+v[1], z=p[2]+v[2])]
    poly_pub.publish(poly)

    if abs(nx-nx_des)<nx_out:
        if abs(nx-nx_des)<nx_in:
            if not nx_flg:
                nx_flg = True
            if abs(d-d_des)<d_out:
                if abs(d-d_des)<d_in:
                    if d_flg:
                        vel_msg = mC.goStraight()
                    else:
                        d_flg = True
                        vel_msg = mC.goStraight()

                else:
                    if d_flg:
                        vel_msg = mC.goStraight()
                    else:
                        if d < d_des-d_in:
                            vel_msg = mC.goFurther()
                        else:
                            vel_msg = mC.goCloser()
            else:
                d_flg = False
                if d < d_des-d_out:
                    vel_msg = mC.goFurther()
                else:
```

```
                    vel_msg = mC.goCloser()
        else:
            if nx_flg:
                if abs(d-d_des)<d_out:
                    if abs(d-d_des)<d_in:
                        if d_flg:
                            vel_msg = mC.goStraight()
                        else:
                            d_flg = True
                            vel_msg = mC.goStraight()

                    else:
                        if d_flg:
                            vel_msg = mC.goStraight()
                        else:
                            if d < d_des-d_in:
                                vel_msg = mC.goFurther()
                            else:
                                vel_msg = mC.goCloser()
                else:
                    d_flg = False
                    if d < d_des-d_out:
                        vel_msg = mC.goFurther()
                    else:
                        vel_msg = mC.goCloser()
            else:
                if nx < 0:
                    vel_msg = mC.rightRot()
                else:
                    vel_msg = mC.leftRot()

    else:
        nx_flg = False
        if nx < 0:
            vel_msg = mC.rightRot()
        else:
            vel_msg = mC.leftRot()

    try:
        vel_pub.publish(vel_msg)
        #rospy.loginfo(vel_msg)
    except:
        pass

def motionControl():
    global vel_pub
    global transf
    global point_pub
    global poly_pub
```

```
    rospy.init_node("cmd_vel")
    vel_pub = rospy.Publisher("cmd_vel", Twist, queue_size=1)
    point_pub = rospy.Publisher("wallinfo/base_frame/point",
   PointStamped, queue_size=1)
    poly_pub = rospy.Publisher("wallinfo/base_frame/polygon",
   PolygonStamped, queue_size=1)

    tfBuffer = tf2_ros.Buffer()
    listener = tf2_ros.TransformListener(tfBuffer)
    transf = []
    while transf == []:
        try:
            transf = tfBuffer.lookup_transform('base_link', '
   LatCam_depth_optical_frame', rospy.Time(0))
        except:
            pass
    rospy.Subscriber("/wallinfo/LatCam_frame", wall, motionCallback,
   queue_size=None)
    rospy.spin()

if __name__ == "__main__":
    try:
        motionControl()
    except rospy.ROSInterruptException:
        pass
```

B.5 Control joy

```
#!/usr/bin/env python

import time
import rospy
import numpy as np
import math
import tf
from geometry_msgs.msg import Twist
from std_msgs.msg import Bool, String
from sensor_msgs.msg import Joy
from robo_explorer.msg import robo_io
import os

# JOY CMD VECTOR JCV
# l2  r2  lao lav rao rav l3  fo  fv
# AX5 AX4 AX0 AX1 AX2 AX3 B14 AX6 AX7
```

```

stat = 0
ready = 0

V = 0
W = 0
velcom = Twist()
Vmax = 300 #pwm
DecelRate = 0.05
AccelRate = 0.05
secure_bound = 0.07
securetime = 5

circflg = False
wd4flg = False
sstrflg = False
motcontflg = False
recflg = False

l2 = 1
r2 = 1
lao = 0
lav = 0
rao = 0
rav = 0
r3 = 0
fo = 0
fv = 0
L = False

def motion_callback(cmds):
    global velcom
    velcom = cmds

def callback(jcmd):
    global stat
    global ready
    global l2
    global r2
    global lao
    global lav
    global rao
    global rav
    global r3
    global fo
    global fv
    global L
    global motcontflg
    global motsub
```

```
    global recflg
    if stat == 0:
        if ready == 0:
            if jcmd.axes[5]==-1 and jcmd.axes[4]==-1:
                ready = 1
                rospy.loginfo("PRESS A TO START TRANSMISSION")
            else:
                rospy.loginfo("PRESS L2 AND R2 TO INITIALIZE TRANSMISSION")
        else:
            if jcmd.buttons[0]==1:
                stat = 1
                rospy.loginfo("READY TO TRANSMIT")
            else:
                rospy.loginfo("PRESS A TO START TRANSMISSION")
    else:
        l2  = jcmd.axes[5]
        r2  = jcmd.axes[4]
        lao = -jcmd.axes[0]
        lav = jcmd.axes[1]
        rao = -jcmd.axes[2]
        rav = jcmd.axes[3]
        r3  = jcmd.buttons[14]
        fo  = -jcmd.axes[6]
        fv  = jcmd.axes[7]
        if jcmd.buttons[1]==1:
            recflg = ~recflg
            light = robo_io()
            if recflg:
                #os.system('sh /home/oem/Documents/AutoBags/record.sh')
                os.system('sh /media/oem/Samsung_T5/AutoBags/record.sh')
                light.out_0 = True
                modpub.publish(light)
                time.sleep(1)
                light.out_0 = False
                modpub.publish(light)
            else:
                nodes = os.popen("rosnode list").readlines()
                for i in range(len(nodes)):
                    nodes[i] = nodes[i].replace("\n","")
                for node in nodes:
                    if "record" in node:
                            os.system("rosnode kill "+ node)
                light.out_0 = True
                modpub.publish(light)
                time.sleep(0.5)
                light.out_0 = False
```

```
                    modpub.publish(light)
                    time.sleep(0.5)
                    light.out_0 = True
                    modpub.publish(light)
                    time.sleep(0.5)
                    light.out_0 = False
                    modpub.publish(light)
          if jcmd.buttons[4]==1:
               light = robo_io()
               if L == False:
                    light.out_0 = True
                    modpub.publish(light)
                    rospy.loginfo("LIGHT ON")
               else:
                    light.out_0 = False
                    modpub.publish(light)
                    rospy.loginfo("LIGHT OFF")
               L = ~L
          if jcmd.buttons[7]==1:
               motcontflg = not(motcontflg)
               if motcontflg:
                    motsub = rospy.Subscriber('/cmd_vel', Twist,
    motion_callback)
               else:
                    motsub.unregister()



def control_joy():
     global securetime
     rospy.init_node('control_joy')
     global pub
     global modpub
     global motsub

     global circflg
     global wd4flg
     global sstrflg
     global motcontflg

     global l2
     global r2
     global lao
     global lav
     global rao
     global rav
     global r3
     global fo
     global fv
```

```
    global V
    global W
    global rate
    global velcom

    v = 0

    pub = rospy.Publisher('/robo_explorer/cmd_vel', Twist, queue_size=10)
    modpub = rospy.Publisher('/robo_explorer/io_status', robo_io, queue_size=10)
    statpub = rospy.Publisher('/robo_explorer/state', String, queue_size=10)
    rospy.Subscriber('/joy', Joy, callback)
    rate = rospy.Rate(10)
    motsub = rospy.Subscriber('/cmd_vel', Twist, motion_callback)
    time.sleep(1)
    motsub.unregister()
    statpub.publish("2wd")

    while not rospy.is_shutdown():
        vel = Twist()
        if motcontflg:
            if velcom.angular.z!=0 and velcom.linear.x==0 and velcom.linear.y==0:
                if circflg==False and wd4flg==False and sstrflg==False:
                    SecureVelShutdown()
                    dw4ON = robo_io()
                    dw4ON.out_3 = True
                    modpub.publish(dw4ON) # 4WD ON
                    rospy.loginfo("4WD ON")
                    time.sleep(1)
                    cstON = robo_io()
                    cstON.out_2 = True
                    modpub.publish(cstON) # CIRC ON
                    rospy.loginfo("CIRCLE STEERING ON")
                    time.sleep(1)
                    chvel = Twist()
                    chvel.linear.x = 0
                    chvel.linear.y = 0
                    chvel.angular.z = -1
                    pub.publish(chvel)
                    time.sleep(securetime)
                    circflg = True
                    statpub.publish("circ")
                elif circflg==True and wd4flg==False and sstrflg==False:
```

```
                pass
            elif circflg==False and wd4flg==True and sstrflg==False:
                SecureVelShutdown()
                cstON = robo_io()
                cstON.out_2 = True
                modpub.publish(cstON) # CIRC ON
                rospy.loginfo("CIRCLE STEERING ON")
                time.sleep(1)
                SecureVelShutdown()
                chvel = Twist()
                chvel.linear.x =  0
                chvel.linear.y =  0
                chvel.angular.z = -1
                pub.publish(chvel)
                time.sleep(securetime)
                wd4flg = False
                circflg = True
                statpub.publish("circ")
            elif circflg==False and wd4flg==False and sstrflg==True:
                SecureVelShutdown()
                if W != 0:
                    SecureVelShutdown()
                    chvel = Twist()
                    chvel.linear.x =  0
                    chvel.linear.y =  0
                    chvel.angular.z = 0
                    pub.publish(chvel)
                    time.sleep(securetime)
                sstOFF = robo_io()
                sstOFF.out_1 = True
                modpub.publish(sstOFF) #SIDESTEER OFF
                rospy.loginfo("SIDE STEERING OFF")
                time.sleep(1)
                cstON = robo_io()
                cstON.out_2 = True
                modpub.publish(cstON) # CIRC ON
                rospy.loginfo("CIRCLE STEERING ON")
                time.sleep(1)
                SecureVelShutdown()
                chvel = Twist()
                chvel.linear.x =  0
                chvel.linear.y =  0
                chvel.angular.z = -1
                pub.publish(chvel)
                time.sleep(securetime)
                sstrflg = False
                circflg = True
```

```
                statpub.publish("circ")
            W = -1
            if np.abs(V-velcom.angular.z) <= secure_bound:
                V = velcom.angular.z
            else:
                V = V + np.sign(velcom.angular.z)*AccelRate

        elif velcom.angular.z==0 and velcom.linear.x!=0 and velcom.linear.y==0:
            if circflg==False and wd4flg==False and sstrflg==False:
                SecureVelShutdown()
                if W != 0:
                    chvel = Twist()
                    chvel.linear.x = 0
                    chvel.linear.y = 0
                    chvel.angular.z = 0
                    pub.publish(chvel)
                    time.sleep(securetime)
                dw4ON = robo_io()
                dw4ON.out_3 = True
                modpub.publish(dw4ON) # 4WD ON
                rospy.loginfo("4WD ON")
                time.sleep(1)
                sstON = robo_io()
                sstON.out_1 = True
                modpub.publish(sstON) #SIDESTEER ON
                rospy.loginfo("SIDE STEERING ON")
                time.sleep(1)
                sstrflg = True
                statpub.publish("side")
            elif circflg==True and wd4flg==False and sstrflg==False:
                SecureVelShutdown()
                chvel = Twist()
                chvel.linear.x = 0
                chvel.linear.y = 0
                chvel.angular.z = 0
                pub.publish(chvel)
                time.sleep(securetime)
                cstOFF = robo_io()
                cstOFF.out_2 = True
                modpub.publish(cstOFF) #CIRC OFF
                rospy.loginfo("CIRCLE STEERING OFF")
                time.sleep(1)
                sstON = robo_io()
                sstON.out_1 = True
                modpub.publish(sstON) #SIDESTEER ON
                rospy.loginfo("SIDE STEERING ON")
```

```
                    time.sleep(1)
                    circflg = False
                    sstrflg = True
                    statpub.publish("side")
                    W = 0
                elif circflg==False and wd4flg==True and sstrflg==False:
                    SecureVelShutdown()
                    if W != 0:
                        SecureVelShutdown()
                        chvel = Twist()
                        chvel.linear.x =  0
                        chvel.linear.y =  0
                        chvel.angular.z = 0
                        pub.publish(chvel)
                        time.sleep(securetime)
                    sstON = robo_io()
                    sstON.out_1 = True
                    modpub.publish(sstON) #SIDESTEER ON
                    rospy.loginfo("SIDE STEERING ON")
                    time.sleep(1)
                    wd4flg = False
                    sstrflg = True
                    statpub.publish("side")
                elif circflg==False and wd4flg==False and sstrflg==True:
                    pass
                if W != 0:
                    SecureVelShutdown()
                    chvel = Twist()
                    chvel.linear.x =  0
                    chvel.linear.y =  0
                    chvel.angular.z = 0
                    pub.publish(chvel)
                    time.sleep(securetime*2)
                W = 0
                if np.abs(V-velcom.linear.x) <= secure_bound:
                    V = velcom.linear.x
                else:
                    V = V + np.sign(velcom.linear.x)*AccelRate

            elif velcom.angular.z==0 and velcom.linear.x==0 and velcom.linear.y!=0:
                if circflg==False and wd4flg==False and sstrflg==False:
                    SecureVelShutdown()
                    if W != 0:
                        chvel = Twist()
                        chvel.linear.x =  0
```

```
                    chvel.linear.y =  0
                    chvel.angular.z = 0
                    pub.publish(chvel)
                    time.sleep(securetime)
                dw4ON = robo_io()
                dw4ON.out_3 = True
                modpub.publish(dw4ON) # 4WD ON
                rospy.loginfo("4WD ON")
                time.sleep(1)
                sstON = robo_io()
                sstON.out_1 = True
                modpub.publish(sstON) #SIDESTEER ON
                rospy.loginfo("SIDE STEERING ON")
                time.sleep(1)
                sstrflg = True
                statpub.publish("side")
            elif circflg==True and wd4flg==False and sstrflg==
    False:
                SecureVelShutdown()
                chvel = Twist()
                chvel.linear.x =  0
                chvel.linear.y =  0
                chvel.angular.z = 0
                pub.publish(chvel)
                time.sleep(securetime)
                cstOFF = robo_io()
                cstOFF.out_2 = True
                modpub.publish(cstOFF) #CIRC OFF
                rospy.loginfo("CIRCLE STEERING OFF")
                time.sleep(1)
                sstON = robo_io()
                sstON.out_1 = True
                modpub.publish(sstON) #SIDESTEER ON
                rospy.loginfo("SIDE STEERING ON")
                time.sleep(1)
                circflg = False
                sstrflg = True
                statpub.publish("side")
                W = 0
            elif circflg==False and wd4flg==True and sstrflg==
    False:
                SecureVelShutdown()
                if W != 0:
                    SecureVelShutdown()
                    chvel = Twist()
                    chvel.linear.x =  0
                    chvel.linear.y =  0
                    chvel.angular.z = 0
                    pub.publish(chvel)
```

```
                        time.sleep(securetime)
                        W = 0
                    sstON = robo_io()
                    sstON.out_1 = True
                    modpub.publish(sstON) #SIDESTEER ON
                    rospy.loginfo("SIDE STEERING ON")
                    time.sleep(1)
                    wd4flg = False
                    sstrflg = True
                    statpub.publish("side")
                elif circflg==False and wd4flg==False and sstrflg==True:
                    pass
                if W != -1:
                    SecureVelShutdown()
                    chvel = Twist()
                    chvel.linear.x =  0
                    chvel.linear.y =  0
                    chvel.angular.z = -1
                    pub.publish(chvel)
                    time.sleep(securetime*2)
                W = -1
                if np.abs(V-velcom.linear.y) <= secure_bound:
                    V = velcom.linear.y
                else:
                    V = V + np.sign(velcom.linear.y)*AccelRate
            else:
                SecureVelShutdown()
        else:
            if bool(r2!=1) != bool(l2!=1):
                if np.abs(V-((1-r2)/2+(l2-1)/2)) <= secure_bound:
                    V = (1-r2)/2+(l2-1)/2
                else:
                    V = V + np.sign((1-r2)/2+(l2-1)/2)*AccelRate
            else:
                if np.abs(V)>secure_bound:
                    V = np.sign(V)*(np.abs(V)-DecelRate)
                else:
                    V=0

            if rao!=0 and r3==0 and lao==0 and fo==0 and fv==0:
                if circflg==False and wd4flg==False and sstrflg==False:
                    pass
                elif circflg==True and wd4flg==False and sstrflg==False:
                    SecureVelShutdown()
                    chvel = Twist()
                    chvel.linear.x =  0
```

```
                chvel.linear.y =  0
                chvel.angular.z = 0
                pub.publish(chvel)
                time.sleep(securetime)
                cstOFF = robo_io()
                cstOFF.out_2 = True
                modpub.publish(cstOFF) #CIRC OFF
                rospy.loginfo("CIRCLE STEERING OFF")
                time.sleep(1)
                dw4OFF = robo_io()
                dw4OFF.out_3 = True
                modpub.publish(dw4OFF) #4WD OFF
                rospy.loginfo("4WD OFF")
                time.sleep(1)
                circflg = False
                statpub.publish("2wd")
            elif circflg==False and wd4flg==True and sstrflg==
    False:
                if W!=0:
                    SecureVelShutdown()
                    chvel = Twist()
                    chvel.linear.x =  0
                    chvel.linear.y =  0
                    chvel.angular.z = 0
                    pub.publish(chvel)
                    time.sleep(securetime)
                dw4OFF = robo_io()
                dw4OFF.out_3 = True
                modpub.publish(dw4OFF) #4WD OFF
                rospy.loginfo("4WD OFF")
                time.sleep(1)
                wd4flg = False
                statpub.publish("2wd")
            elif circflg==False and wd4flg==False and sstrflg==
    True:
                SecureVelShutdown()
                if W!=0:
                    chvel = Twist()
                    chvel.linear.x =  0
                    chvel.linear.y =  0
                    chvel.angular.z = 0
                    pub.publish(chvel)
                    time.sleep(securetime)
                sstOFF = robo_io()
                sstOFF.out_1 = True
                modpub.publish(sstOFF) #SIDESTEER OFF
                rospy.loginfo("SIDE STEERING OFF")
                time.sleep(1)
                dw4OFF = robo_io()
```

```
                dw4OFF.out_3 = True
                modpub.publish(dw4OFF) #4WD OFF
                rospy.loginfo("4WD OFF")
                time.sleep(1)
                sstrflg = False
                statpub.publish("2wd")
            W = -1*np.sign(rao)

        elif r3==1 and lao==0 and fo==0 and fv==0:

            if circflg==False and wd4flg==False and sstrflg==False:
                SecureVelShutdown()
                dw4ON = robo_io()
                dw4ON.out_3 = True
                modpub.publish(dw4ON) # 4WD ON
                rospy.loginfo("4WD ON")
                time.sleep(1)
                cstON = robo_io()
                cstON.out_2 = True
                modpub.publish(cstON) # CIRC ON
                rospy.loginfo("CIRCLE STEERING ON")
                time.sleep(1)
                chvel = Twist()
                chvel.linear.x = 0
                chvel.linear.y = 0
                chvel.angular.z = -1
                pub.publish(chvel)
                time.sleep(securetime)
                circflg = True
                statpub.publish("circ")
            elif circflg==True and wd4flg==False and sstrflg==False:
                pass
            elif circflg==False and wd4flg==True and sstrflg==False:
                cstON = robo_io()
                cstON.out_2 = True
                modpub.publish(cstON) # CIRC ON
                rospy.loginfo("CIRCLE STEERING ON")
                time.sleep(1)
                SecureVelShutdown()
                chvel = Twist()
                chvel.linear.x = 0
                chvel.linear.y = 0
                chvel.angular.z = -1
                pub.publish(chvel)
                time.sleep(securetime)
                wd4flg = False
```

```
                    circflg = True
                    statpub.publish("circ")
                elif circflg==False and wd4flg==False and sstrflg==True:
                    if W != 0:
                        SecureVelShutdown()
                        chvel = Twist()
                        chvel.linear.x =  0
                        chvel.linear.y =  0
                        chvel.angular.z = 0
                        pub.publish(chvel)
                        time.sleep(securetime)
                    sstOFF = robo_io()
                    sstOFF.out_1 = True
                    modpub.publish(sstOFF) #SIDESTEER OFF
                    rospy.loginfo("SIDE STEERING OFF")
                    time.sleep(1)
                    cstON = robo_io()
                    cstON.out_2 = True
                    modpub.publish(cstON) # CIRC ON
                    rospy.loginfo("CIRCLE STEERING ON")
                    time.sleep(1)
                    SecureVelShutdown()
                    chvel = Twist()
                    chvel.linear.x =  0
                    chvel.linear.y =  0
                    chvel.angular.z = -1
                    pub.publish(chvel)
                    time.sleep(securetime)
                    sstrflg = False
                    circflg = True
                    statpub.publish("circ")
                W = -1

            elif rao==0 and r3==0 and lao!=0 and fo==0 and fv==0:
                if circflg==False and wd4flg==False and sstrflg==False:
                    if W != 0:
                        SecureVelShutdown()
                        chvel = Twist()
                        chvel.linear.x =  0
                        chvel.linear.y =  0
                        chvel.angular.z = 0
                        pub.publish(chvel)
                        time.sleep(securetime)
                    dw4ON = robo_io()
                    dw4ON.out_3 = True
                    modpub.publish(dw4ON) # 4WD ON
                    rospy.loginfo("4WD ON")
```

```
                    time.sleep(1)
                    wd4flg = True
                    statpub.publish("4wd")
                elif circflg==True and wd4flg==False and sstrflg==False:
                    SecureVelShutdown()
                    chvel = Twist()
                    chvel.linear.x =  0
                    chvel.linear.y =  0
                    chvel.angular.z = 0
                    pub.publish(chvel)
                    time.sleep(securetime)
                    cstOFF = robo_io()
                    cstOFF.out_2 = True
                    modpub.publish(cstOFF) #CIRC OFF
                    rospy.loginfo("CIRCLE STEERING OFF")
                    time.sleep(1)
                    circflg = False
                    wd4flg = True
                    statpub.publish("4wd")
                elif circflg==False and wd4flg==True and sstrflg==False:
                    pass
                elif circflg==False and wd4flg==False and sstrflg==True:
                    SecureVelShutdown()
                    chvel = Twist()
                    chvel.linear.x =  0
                    chvel.linear.y =  0
                    chvel.angular.z = 0
                    pub.publish(chvel)
                    time.sleep(securetime)
                    sstOFF = robo_io()
                    sstOFF.out_1 = True
                    modpub.publish(sstOFF) #SIDESTEER OFF
                    rospy.loginfo("SIDE STEERING OFF")
                    time.sleep(1)
                    sstrflg = False
                    wd4flg = True
                    statpub.publish("4wd")
                W = -1*np.sign(lao)

            elif rao==0 and r3==0 and lao==0 and (fo!=0 or fv!=0):
                if circflg==False and wd4flg==False and sstrflg==False:
                    SecureVelShutdown()
                    if W != 0:
                        chvel = Twist()
                        chvel.linear.x =  0
```

```
                    chvel.linear.y =  0
                    chvel.angular.z = 0
                    pub.publish(chvel)
                    time.sleep(securetime)
                dw4ON = robo_io()
                dw4ON.out_3 = True
                modpub.publish(dw4ON) # 4WD ON
                rospy.loginfo("4WD ON")
                time.sleep(1)
                sstON = robo_io()
                sstON.out_1 = True
                modpub.publish(sstON) #SIDESTEER ON
                rospy.loginfo("SIDE STEERING ON")
                time.sleep(1)
                sstrflg = True
                statpub.publish("side")
            elif circflg==True and wd4flg==False and sstrflg==False:
                SecureVelShutdown()
                chvel = Twist()
                chvel.linear.x =  0
                chvel.linear.y =  0
                chvel.angular.z = 0
                pub.publish(chvel)
                time.sleep(securetime)
                cstOFF = robo_io()
                cstOFF.out_2 = True
                modpub.publish(cstOFF) #CIRC OFF
                rospy.loginfo("CIRCLE STEERING OFF")
                time.sleep(1)
                sstON = robo_io()
                sstON.out_1 = True
                modpub.publish(sstON) #SIDESTEER ON
                rospy.loginfo("SIDE STEERING ON")
                time.sleep(1)
                circflg = False
                sstrflg = True
                statpub.publish("side")
            elif circflg==False and wd4flg==True and sstrflg==False:
                if W != 0:
                    SecureVelShutdown()
                    chvel = Twist()
                    chvel.linear.x =  0
                    chvel.linear.y =  0
                    chvel.angular.z = 0
                    pub.publish(chvel)
                    time.sleep(securetime)
                sstON = robo_io()
```

```
                    sstON.out_1 = True
                    modpub.publish(sstON) #SIDESTEER ON
                    rospy.loginfo("SIDE STEERING ON")
                    time.sleep(1)
                    wd4flg = False
                    sstrflg = True
                    statpub.publish("side")
                elif circflg==False and wd4flg==False and sstrflg==
    True:
                    pass

                if fo!=0 and fv!=0:
                    pass
                elif fo==0 and fv!=0:
                    W = 0
                elif fo!=0 and fv==0:
                    W = -1
            elif rao==0 and r3==0 and lao==0 and fo==0 and fv==0:
                if circflg==True:
                    SecureVelShutdown()
                    chvel = Twist()
                    chvel.linear.x =  0
                    chvel.linear.y =  0
                    chvel.angular.z = 0
                    pub.publish(chvel)
                    time.sleep(securetime)
                    cstOFF = robo_io()
                    cstOFF.out_2 = True
                    modpub.publish(cstOFF) #CIRC OFF
                    rospy.loginfo("CIRCLE STEERING OFF")
                    time.sleep(1)
                    dw4OFF = robo_io()
                    dw4OFF.out_3 = True
                    modpub.publish(dw4OFF) #4WD OFF
                    rospy.loginfo("4WD OFF")
                    time.sleep(1)
                    circflg = False
                    statpub.publish("2wd")
                if sstrflg==False:
                    W = 0
            else:
                pass

        vel.linear.x =  -V*Vmax
        vel.linear.y =  0
        vel.angular.z = W
        pub.publish(vel)
        rate.sleep()
```

```
def SecureVelShutdown():
    global pub
    global V
    global W
    chvel = Twist()
    while np.abs(V)>secure_bound:
        V = np.sign(V)*(np.abs(V)-DecelRate)
        chvel.linear.x = -V*Vmax
        chvel.linear.y = 0
        chvel.angular.z = W
        pub.publish(chvel)
        rate.sleep()

if __name__ == '__main__':
    control_joy()
```

B.6 Odometry

```
#!/usr/bin/env python

import time
import rospy
import numpy as np
import math
import tf
from geometry_msgs.msg import Point, Pose, Quaternion, Twist, Vector3
from std_msgs.msg import String
from nav_msgs.msg import Odometry
from tf.transformations import quaternion_multiply

state = "2wd"
R = 0.13
EncRes = (np.pi*2)/4096
diag = np.sqrt(0.58**2+0.75**2)/2
steerang = np.pi/6
width = 0.58
depth = 0.75
cir2wd = np.array([depth/2,(width/2)+depth*np.tan(np.pi/2-steerang)
    ,0])
cir4wd = np.array([0,(width/2)+(depth/2)*np.tan(np.pi/2-steerang),0])
# enc topic [time stamp, RL, RR, FR, FL, count]

x = 0
y = 0
th = 0
```

```

#lt = 0
#le = np.array([0,0,0,0])

def enc_callback(data):
    global odom_pub
    global odom_broadcaster
    global state
    global W
    global lt
    global le
    global x
    global y
    global th

    global R
    global EncRes
    global diag
    global steerang
    global cir2wd
    global cir4wd
    global depth
    global width

    enc = data.data.split(',')
    ct = float(enc[0])
    ce = np.array([float(enc[1]),float(enc[2]),float(enc[3]),float(enc[4])])
    try:
        dt = ct-lt
        dg = le-ce

        odom = Odometry()
        if state == "circ":
            vx = 0
            vy = 0
            dth = R*EncRes*(np.abs(dg)).mean()/diag
            vth = np.sign(dg[1])*dth/dt
        if state == "2wd":
            if W== -1:
                dst = R*EncRes*(dg)
                dst = (1/dt)*np.array([dst[0]/(width/2+cir2wd[1]),
                                dst[1]/(cir2wd[1]-width/2),
                                dst[2]/np.sqrt(depth**2+(cir2wd[1]-width/2)**2),
                                dst[3]/np.sqrt(depth**2+(cir2wd[1]+width/2)**2)])
                omg = np.array([0,0,(np.abs(dst)).mean()])
                vel = np.cross(-omg,cir2wd)
```

```
                vx = vel[0]
                vy = vel[1]
                vth = -omg[2]
            elif W == 1:
                dst = R*EncRes*(dg)
                dst = (1/dt)*np.array([dst[0]/(cir2wd[1]-width/2),
                                       dst[1]/(width/2+cir2wd[1]),
                                       dst[2]/np.sqrt(depth**2+(cir2wd[1]+width/2)**2),
                                       dst[3]/np.sqrt(depth**2+(cir2wd[1]-width/2)**2)])
                omg = np.array([0,0,(np.abs(dst)).mean()])
                vel = np.cross(omg,np.array([cir2wd[0],-cir2wd[1],cir2wd[2]]))
                vx = vel[0]
                vy = vel[1]
                vth = omg[2]
            else:
                ds = R*EncRes*(dg.mean())
                vx = ds/dt
                vy = 0
                vth = 0
        elif state == "4wd":
            if W == -1:
                dst = R*EncRes*(dg)
                dss = (1/dt)*np.array([dst[0]/np.sqrt((depth/2)**2+(cir4wd[1]+width/2)**2),
                                       dst[1]/np.sqrt((depth/2)**2+(cir4wd[1]-width/2)**2),
                                       dst[2]/np.sqrt((depth/2)**2+(cir4wd[1]-width/2)**2),
                                       dst[3]/np.sqrt((depth/2)**2+(cir4wd[1]+width/2)**2)])
                omg = np.array([0,0,(np.abs(dss)).mean()])
                vel = np.cross(-omg,cir4wd)
                vx = vel[0]
                vy = vel[1]
                vth = -omg[2]
            elif W == 1:
                dst = R*EncRes*(dg)
                dss = (1/dt)*np.array([dst[0]/np.sqrt((depth/2)**2+(cir4wd[1]-width/2)**2),
                                       dst[1]/np.sqrt((depth/2)**2+(cir4wd[1]+width/2)**2),
                                       dst[2]/np.sqrt((depth/2)**2+(cir4wd[1]+width/2)**2),
                                       dst[3]/np.sqrt((depth/2)**2+(cir4wd[1]-width/2)**2)])
                omg = np.array([0,0,(np.abs(dss)).mean()])
```

```python
                vel = np.cross(omg,np.array([cir4wd[0],-cir4wd[1],cir4wd[2]]))
                vx = vel[0]
                vy = vel[1]
                vth = omg[2]
            else:
                ds = R*EncRes*(dg.mean())
                vx = ds/dt
                vy = 0
                vth = 0
        elif state == "side":
            if W== -1:
                ds = R*EncRes*(dg.mean())
                vx = 0
                vy = -ds/dt
                vth = 0
            else:
                ds = R*EncRes*(dg.mean())
                vx = ds/dt
                vy = 0
                vth = 0
        th = th+vth*dt
        odom.header.stamp = rospy.Time.now()
        odom.header.frame_id = "odom"
        odom_quat = tf.transformations.quaternion_from_euler(0, 0, th)
        quat = odom_quat
        V_odom = quaternion_multiply(quat,[vx,vy,0,0])
        V_odom = quaternion_multiply(V_odom,[-quat[0],-quat[1],-quat[2],quat[3]])
        x = x + V_odom[0]*dt
        y = y + V_odom[1]*dt
        odom_broadcaster.sendTransform((x, y, 0.), odom_quat, rospy.Time.now(), "base_link", "odom")
        odom.pose.pose = Pose(Point(x, y, 0.), Quaternion(*odom_quat))
        odom.child_frame_id = "base_link"
        odom.twist.twist = Twist(Vector3(vx, vy, 0), Vector3(0, 0, vth))

        odom_pub.publish(odom)
    except:
        pass

    lt = ct
    le = ce


def vel_callback(data):
```

```
    global W
    W = data.angular.z
    #rospy.loginfo(W)

def state_callback(data):
    global state
    state = data.data
    #rospy.loginfo(state)

def odom_publisher():
    global odom_pub
    global state
    global W
    global odom_broadcaster
    global x
    global y
    global th
    x = 0
    y = 0
    th = 0
    rospy.init_node('odom_publisher')
    odom_pub = rospy.Publisher("/robo_explorer/odom", Odometry,
    queue_size=10)
    velsub = rospy.Subscriber('/robo_explorer/cmd_vel', Twist,
    vel_callback)
    statsub = rospy.Subscriber('/robo_explorer/state', String,
    state_callback)
    encsub = rospy.Subscriber('/robo_explorer/enc', String,
    enc_callback)
    odom_broadcaster = tf.TransformBroadcaster()
    rospy.spin()

if __name__ == '__main__':
    odom_publisher()
```

B.7 Kalman Filter

```
clear all, clc
t = zeros(length(cellbag{19, 2}),1);
for i=1:size(cellbag{19, 2},1)
    distance(i,1) = cellbag{19, 2}{i, 1}.Distance;
    NormalVector = cellbag{19, 2}{i, 1}.NormalVector;
    angle(i,1) = rad2deg(atan2(NormalVector(1),NormalVector(3)));
    t(i,1) =    double(cellbag{20, 2}{i, 1}.Header.Stamp.Sec) +...
```

```
                    double(cellbag{20, 2}{i, 1}.Header.Stamp.Nsec)*10^(-9) -...
                    double(cellbag{20, 2}{1, 1}.Header.Stamp.Sec) -...
                    double(cellbag{20, 2}{1, 1}.Header.Stamp.Nsec)*10^(-9);
end
t(1,1) = 0;

dt = 0.15;
A = [0 0 0 0; dt 1 0 0; 0 0 0 0; 0 0 dt 1];
C = [0 1 0 0; 0 0 0 1];

R = 0.5*diag([1 1]);
Q = 0.1*diag([1 1 1 1]);

x0 = [0 0 0 1.5]';
P0 = 0.1*eye(4);

x_k_k = x0;
P_k_k = P0;

y_kp1_kp1 = zeros(51,2);

Ns = length(t);
y = [angle distance]';

for k = 1:Ns
    x_kp1_k = A * x_k_k;
    P_kp1_k = A * P_k_k * A' + Q;


    K = P_kp1_k * C' * inv(C * P_kp1_k * C' + R);
    x_kp1_kp1 = x_kp1_k + K * (y(:,k) - C * x_kp1_k);

    P_kp1_kp1 = (eye(4) - K * C)*P_kp1_k;

    y_kp1_kp1(k,:) = (C * x_kp1_kp1)';

    x_k_k = x_kp1_kp1;
    P_k_k = P_kp1_kp1;

end

%%
close all
set(gcf,'color','w');
set(gca,'fontsize',25)

figure(1)
```

```
nexttile
grid on, hold on
plot(t,y_kp1_kp1(:,1),'b','linewidth',1.5)
plot(t, angle,'r','linewidth',1.5)

title("Kalman Filter")
legend("Filtered angle","Measured angle")
xlabel("Time [s]")
ylabel("Angle [ ]")

figure(1)
nexttile
grid on, hold on
plot(t,y_kp1_kp1(:,2),'b','linewidth',1.5)
plot(t, distance,'r','linewidth',1.5)

legend("Filtered distance","Measured distance")
xlabel("Time [s]")
ylabel("Distance [m]")
```

Bibliography

[1] R. Gebbers and V. Adamchuk. *Precision Agriculture and Food Security*. 2010 (cit. on p. 1).

[2] R. Casa. *Agricoltura di Precisione, Metodi e tenologia per migliorarne l'efficienza*. 2018 (cit. on p. 2).

[3] L. Tamburino, G. Bravo, Y. Clough, and Kimberly A Nicholas. *From population to production: 50 years of scientific literature on how to feed the world*. eng. Vol. 24. Elsevier B.V, 2020, p. 100346 (cit. on p. 3).

[4] D. De Wrachien, B. Schultz, and M. B Goli. *Impacts of population growth and climate change on food production and irrigation and drainage needs: A world-wide view*. eng. 2021 (cit. on p. 3).

[5] M. A Gehan and E. A Kellogg. *High-throughput phenotyping*. eng. Vol. 104. 4. United States: Botanical Society of America, 2017, pp. 505–508 (cit. on p. 3).

[6] Naïo Technologies. *https://www.naio-technologies.com* (cit. on p. 4).

[7] I. Beloev, D. Kinaneva, G. Georgiev, G. Hristov, and P. Zahariev. *Artificial Intelligence-Driven Autonomous Robot for Precision Agriculture*. eng. Vol. 24. 1. Sciendo, 2021, pp. 48–54 (cit. on p. 4).

[8] Geert Verhoeven. *Taking computer vision aloft - archaeological three-dimensional reconstructions from aerial photographs with photoscan*. eng. Vol. 18. 1. Chichester, UK: John Wiley Sons, Ltd, 2011, pp. 67–73 (cit. on p. 6).

[9] Zhihua Xu, Lixin Wu, Yonglin Shen, Fashuai Li, Qiuling Wang, and Ran Wang. *Tridimensional Reconstruction Applied to Cultural Heritage with the Use of Camera-Equipped UAV and Terrestrial Laser Scanner*. eng. Vol. 6. 11. MDPI AG, 2014, pp. 10413–10434 (cit. on p. 6).

[10] Adam Mathews and Jennifer Jensen. *Visualizing and Quantifying Vineyard Canopy LAI Using an Unmanned Aerial Vehicle (UAV) Collected High Density Structure from Motion Point Cloud*. eng. Vol. 5. 5. MDPI AG, 2013, pp. 2164–2183 (cit. on p. 6).

[11] J. Bendig, A. Bolten, and G. Bareth. *UAV-based imaging for multi-temporal, very high resolution crop surface models to monitor crop growth variability.* 2013 (cit. on p. 6).

[12] Marie Weiss and Frédéric Baret. *Using 3D Point Clouds Derived from UAV RGB Imagery to Describe Vineyard 3D Macro-Structure*. eng. Vol. 9. 2. MDPI, 2017, p. 111 (cit. on p. 6).

[13] Yi Lin. *LiDAR: An important tool for next-generation phenotyping technology of high potential for plant phenomics?* eng. Vol. 119. Elsevier B.V, 2015, pp. 61–73 (cit. on p. 7).

[14] David Deery, Jose Jimenez-Berni, Hamlyn Jones, Xavier Sirault, and Robert Furbank. *Proximal Remote Sensing Buggies and Potential Applications for Field-Based Phenotyping*. eng. Vol. 4. 3. MDPI AG, 2014, pp. 349–379 (cit. on p. 7).

[15] Michael Schaefer and David Lamb. *A Combination of Plant NDVI and LiDAR Measurements Improve the Estimation of Pasture Biomass in Tall Fescue (Festuca arundinacea var. Fletcher)*. eng. Vol. 8. 2. MDPI AG, 2016, p. 109 (cit. on p. 7).

[16] Greg J Rebetzke, Jose A Jimenez-Berni, William D Bovill, David M Deery, and Richard A James. *High-throughput phenotyping technologies allow accurate selection of stay-green*. eng. Vol. 67. 17. England: Oxford University Press, 2016, pp. 4919–4924 (cit. on p. 7).

[17] Andrew N French, Michael A Gore, and Alison Thompson. *Cotton phenotyping with lidar from a track-mounted platform*. eng. Vol. 9866. SPIE, 2016, 98660B–98660B-8. ISBN: 9781510601079 (cit. on p. 7).

[18] Matthew H Siebers, Everard J Edwards, Jose A Jimenez-Berni, Mark R Thomas, Michael Salim, and Rob R Walker. *Fast Phenomics in Vineyards: Development of GRover, the Grapevine Rover, and LiDAR for Assessing Grapevine Traits in the Field*. eng. Vol. 18. 9. Switzerland: MDPI, 2018, p. 2924 (cit. on p. 7).

[19] M. G. Bekker. *Theory of land locomotion*. 1956 (cit. on p. 9).

[20] M. G. Bekker. *Introduction to terrain-vehicle systems*. 1969 (cit. on p. 9).

[21] J. Wong. *Theory of ground vehicles*. 2008 (cit. on p. 9).

[22] J. Wong A. and R. Reece. *Prediction of wheel performance based on the analysis of soil-wheel stresses*. 1967 (cit. on pp. 9, 11, 12).

[23] V. Vattiata. *Modeling and identification of wheel-soil interaction for precision agriculture robotics*. 2020 (cit. on p. 10).

[24] M. G. Bekker. *Off-the-Road Locomotion*. 1960 (cit. on p. 11).

[25] Z. Janosi and B. Hanamoto. *The analytical determination of drawbar pull as a function of slip for tracked vehicle in deformable soils*. 1961 (cit. on p. 11).

[26] G. Ishigami, A. Miwa, K. Nagatani, and K. Yoshida. *Terramechanics- based model for steering maneuver of planetary exploration rovers on loose soil*. 2007 (cit. on p. 12).

[27] H. Shibly, K. Iagnemma, and S. Dubowsky. *An equivalent soil mechan- ics formulation for rigid wheels in deformable terrain with application to planetary exploration rovers*. 2005 (cit. on p. 12).

[28] L. Ding, K. Yoshida, K. Nagatani, H. B. Gao, and Z. Q. Deng. *Parameter identification for planetary soil based on a decoupled ana- lytical wheel-soil interaction terramechanics model*. 2009 (cit. on p. 14).

[29] K. R. Xia, L. Ding, H. B. Gao, and Z. Q. Deng. *Motion-control- based analytical model for wheel-soil interaction mechanics of lunar rover*. 2011 (cit. on p. 14).

[30] S. Hutangkabodee, Y. H. Zweiri, L. D. Seneviratne, and K. Althoefer. *Performance prediction of a wheeled vehicle on unknown terrain using identified soil parameters*. 2006 (cit. on p. 14).

[31] S. Hutangkabodee, Y. H. Zweiri, L. D. Senviratne, and K. Althoefer. *Validation of soil parameter identification for track-terrain interaction dynamics*. 2007 (cit. on p. 14).

[32] K. Iagnemma, S. Kang, H. Shibly, and S. Dubowsky. *Online terrain parameter estimation for wheeled mobile robots with application to planetary rovers*. 2004 (cit. on p. 14).

[33] J. Y. Wong. *Terramechanics and Off-Road Vehicle Engineering*. 1989 (cit. on p. 14).

[34] Y. Li, L. Ding, and G. Liu. *Error-Tolerant Switched Robust Extended Kalman Filter With Application to Parameter Estimation of Wheel-Soil Interaction*. 2014 (cit. on pp. 14, 15).

[35] G. Reina, L. Ojeda, A. Milella, and J. Borenstein. *Wheel Slippage and Sinkage Detection for Planetary Rover*. 2006 (cit. on p. 15).

[36] L. Wang, X. Dai, and H. Ju. *Homography-based visual measurement of wheel sinkage for a mobile robot*. 2010 (cit. on p. 15).

[37] D. M. Bevly and J. C. Gerdes. *The Use of GPS Based Velocity Measurements for Improved Vehicle State Estimation*. 2000 (cit. on p. 17).

[38] J. V. Alcantar, F. Assadian, and M. Kuang. *Vehicle Velocity State Estimation using Youla Controller Output Observer*. 2018 (cit. on p. 17).

[39] Martin A Fischler and Robert C Bolles. *Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography*. eng. 1981 (cit. on p. 24).

[40] Y. LeCun, Y. Bengio, and G. Hinton. *Deep learning*. 2015 (cit. on p. 36).

[41] A. Kamilaris and F. Prenafeta-Boldú. *Deep learning in agriculture: a survey*. 2018 (cit. on p. 36).