` *MINISTERY OF THE HIGHER EDUCATION AND THE SCIENTIFIC RESEARCH*

UNIVERSITY MOULOUD MAMMERI, TIZI – OUZOU

*F*aculty of Electrical Engineering and Computer Science

Automatic Department

Master Project Report

Option: Control Systems

Prsented by

IMADOUCHENE YAZID

visual servoing of a mobile robot using the Differential Steering System Direction Dialog and the Speech Recgnition controls

Sustained Master Project Report on 04 *July* 2018, *Presented in Fulfillment of Master degree*

ALI BEY Mohand	*MCB*	*U.M.M.T.O*	*Examiner*
GUERMAH Said,	*MCA*	*U.M.M.T.O*	*President*
MELLAH Rabah,	*Professor*	*U.M.M.T.O*	*director of thesis*

To my mother, may God have mercy upon her soul

To my father, my brothers and my sisters.

To everyone who loves and supports me.

Abstract

The aim of this work is to present a visual servoing of a mobile robot using the controls: DSS Direction Dialog and *SpeachRecognizerGui*. In the first part we talk about how to control a mobile robot by using different methods and a bunch of models chosen for illustrating some controllers. After that we will see an example of a differential drive robots, then we will study their behaviors in the environment which is supposed they will move in. we will talk about making a simple robot. After that we will study the linearization. Also we talk about the stability of the robot when driving it.

In the second part we will talk about visual *odometry*, and see the camera models and some projections of the optic flux.

In the third part, we deal with sensors we may use in the field of mobile robotics by illustrating different types of them and some of their functions the robot may use in its displacement.

The fourth part is about the introduction to control theory. In this part we start to explain dynamical models, after that we will see the different controllers the robot or a car may have to use for their behaviors when they move. We will build different models of regulators and place them in a feedback system which permit us to see if we reach the objectives we have fixed to the robot. We use Simulink and MATLAB to simulate and elaborate the results.

The fifth part is about showing the simulation of a pioneer robot (PDX3) by using a VPL (Visual Programming Language) as a graphical programming language, which is a part of Microsoft Robotics Developer Studio 4 and that is a software we have chosen for our simulation.

In this part we will see how to control a robot according to the scene the camera of the robot will receive when moving. The first simulation is about controlling the robot by using a DSS Direction Dialog. The goal of the second simulation is to show us how to control a robot by using a Speech command (speech recognition). We will see also how the robot may avoid obstacles and going directly to the desired goal.

Acknowledgment

This master thesis is the final course in the Master science program in automatic. When I had started this thesis I didn't know where I was going exactly. It was an open project for me. And I felt that I have to find a specific domain I will need to work with. I was a little bit confused about the next steps to follow. By the time and a little hard work, I have managed to find and write something about how to control mobile robots. My love and the desire I have for that field helped me so much to find an inspiration and a lot of imagination to make some frames and models what I hope that may help students in their works in robotics. Also I always have been attracted by moving robots and knowing that domain will be very useful in the future on what concerning every developing area.

The project in such a magnitude required from me a hard work. I have to thank the few people who encouraged me to finish such Interesting work.

First, I would like to thank my supervisor MELLAH RABAH for giving me a chance to realize such a project.

Also I would like to thank my brother Karim from the USA who send me the computers and the means I have used to accomplish my thesis.

For the people who may read this thesis especially the students who may find interest in robotics, I wish to them a good luck to understand and find something helping them for what they are looking for. Just enjoy your time to read it carefully.

Contents

Chapter III Sensors used for Mobile Robotics

Figure lists

List of tables

General introduction

In modern time, different kinds of mobile robots remain the alternative solution to replace the humans when accomplishing so many hard tasks in appropriate way. That can be done without resting, with best performances and more precision in shorter time. But going faster makes the maintaining of stability [1] a big issue. Usually in real world we have to deal with nonlinear systems. Linearization is required for best understanding.

Providing the robots with a good hardware and software may help them to do a good processing for a huge amount of data, but not enough to accomplish smart tasks. We use Microsoft Kinect [48] which is a computer vision technology that combines a variety of hardware to capture image, depth, and sound. This objective is aimed towards gathering a complete knowledge of what the Kinect is capable of and how this is applicable to this robot design. To make the robots robust and efficient [1], we need constantly to enhance their behaviors by making them as a learning machines. To go to any next step robots need to require to their own memorized data and compare them with what the sensors may provide them from the environment. The complex algorithms [5] are commonly used even sometimes for achieving tasks seen as simple for humans but difficult for the robots. An artificial brain also called a processor is used instead of the intelligence of the living creatures.

The fundamental challenge so many engineers have to overcome in robotics world is how to control [7] [14] such a machines when facing harsh conditions. The solution for that is to build a bunch of controllers and making a libraries [1] for different applications when that needed. The same problem has been practically seen in every domain like in the marine, space and in land.

Autonomous robots [35] are used to fill up the lack of control we have on such a devices. That is the best way to ensure a better result when we have affair to a critical environment. So many space missions has been conducted successfully when exploring other part of the

-universe. We have been a witness for a robots sent to explore other planets or there moons like Europa one of the Jupiter's moons. The signals the scientist may have to use to communicate with robots for a long distances require too much time. By depending only on such a method that may lead to lose control over the robots. We assume that mobile robots must know where they are, and localizing [4] their pose (position and orientation) or trying to reach tracking [1] requires a good measurements. The better way is to choose a good sensors [17].

So many engineers are working on various drive mechanism in order to monitor those robots. The solution to ensure a best driving is using two wheels-drive with differential steering and a free balancing wheel [1]. Also to drive well the robots and manoeuver in good conditions, it is better to control the tow motors independently.

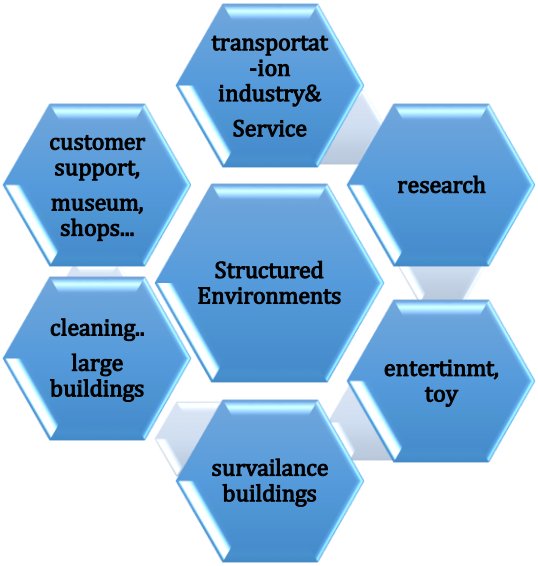
To make our robots well accepted in the world of market, we need to reach so many important objectives, like maintaining a good stability, tracking a reference signal, building a robust device, find a way to avoid a disturbances, reaching goals in optimal way (crossing a short path in a short time), finely effectiveness is required [1] [7]. And we have to make a better controller designs and different methods and models for our robots.

We look forward to find a way helping the robots avoiding obstacles. For that so many engineers are working to improve the behaviors of the robots by using different kind of methods. Also to reach a better tracking to the input signals, we need to use feedback control by using different kinds of controllers, PID regulators are introduced [14]. We also use visual *odometry* [4] [5] [6] [10] [22] [21] [28] for better interpretations of the images or the videos the robots may receive from the real environment according to the camera scene.

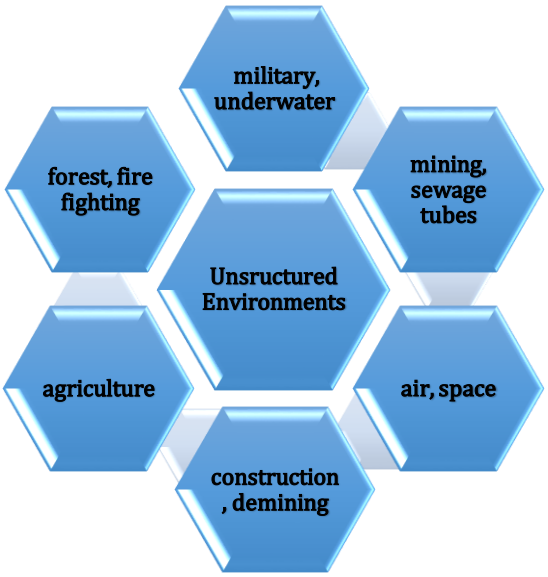
For our simulation we use Microsoft Robotics Developer Studio commercial developers to create robotics applications for a variety of hardware platforms 4 (MRDS 4) [36] is a Windows-based environment for hobbyist, academic and. Its Components are: VPL (Visual Programming Language, VSE (Visual Simulation Environment), DSS (Decentralized Software Services), CCR (Concurrency and Coordination Runtime).

Now we have resumed some application domains of the mobile robot as shown into the following:

Indoor



Outdoor



CHAPTER I. The modeling and controlling a mobile robot

I.1.Introduction

In order to control a mobile robot we have to use a model for that. We can start with the common used model. We will use a model of control a mobile robot, which is a simplified modeling approach modified from the differential drive mobile robots. Instead of controlling the right speed v_l and the left speed v_r of the drive systems. The unicycle model using u and ω as the controller parameters. Tracking is much easier in this model, and its dynamic is controlled by using a PID controller.

I.2.Building a model

I.2.1.Differential drive robots

We use a differential drive wheeled robot which is a very common type as we can illustrate in the $Fig\ I.1$. A unicycle type robot is in general a robot moving in a 2D world, having some forward speed but zero Instantaneous lateral motion. In other words, it is a Non-holonomic [1.] system [2][41].

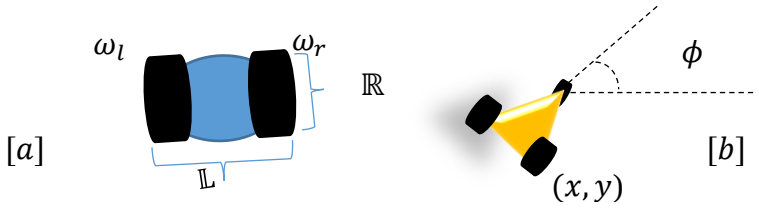


Fig I. 1.The models of the unicycle differential drive robots

As mentioned in the figure above we have to know first the dimension of the robot. Here we have two parameters $\mathbb{L}$ and $\mathbb{R}$. But as we will see in the fourth chapter that a good controller shouldn't have to know exactly what particular parameters are, because typically we don't know what the friction coefficient is.in this example we have to know how far the wheels are from each other, and that is the parameter base of the robot, we called it $\mathbb{L}$. We have to know too how big are the wheels which known as the radius of the wheels, we called it $\mathbb{R}$. These two parameters have a little role to play when we are designing a controllers. We

[1.] Non holonomic systems are where the velocities (magnitude and or direction) or other derivatives of the position are constraint. In physics and mathematics is a system whose state depends on the path taken in order to achieve it.

-want to know what is it about the robot we are be able to control, like moving. The control signals that we have are ω_r which is the rate the right wheel is turning, and ω_l which is the rate the left wheel is turning. These are two inputs of our system. We look now for the state of our robot. We have x and y indicating for us the position of the robot. And the direction will be indicated by ϕ. What we need to do is to connect the inputs to the states. We write now the differential equation of the robot and it is shown as in the following differential model (a):

$$\begin{cases} \dot{x} = \frac{\mathbb{R}}{2}(\omega_r + \omega_l)\cos\phi \\ \dot{y} = \frac{\mathbb{R}}{2}(\omega_r + \omega_l)\sin\phi \\ \dot{\phi} = \frac{\mathbb{R}}{\mathbb{L}}(\omega_r - \omega_l) \end{cases} \quad (1.1)$$

For the unicycle model we have the inputs as v and ω. For the unicycle dynamics [2] [27] [41], we can write the differential equation as the following simple model (b):

$$\begin{cases} \dot{x} = v\cos\phi \\ \dot{y} = v\sin\phi \\ \dot{\phi} = \omega \end{cases} \quad (1.2)$$

This model is highly useful.

If we put $\phi = 0$ then the robot will move in straight way in the x direction.

Now we are going to design our controller. From the comparison of the two models we can have the following equations:

$$v = \frac{\mathbb{R}}{2}(\omega_r + \omega_l) \Rightarrow \frac{2v}{\mathbb{R}} = \omega_r + \omega_l \quad (1.3)$$

$$\omega = \frac{\mathbb{R}}{\mathbb{L}}(\omega_r - \omega_l) \Rightarrow \omega\frac{\mathbb{L}}{\mathbb{R}} = \omega_r - \omega_l \quad (1.4)$$

We have v as the translation velocity, and the real velocities are v_r and v_l . In the end we can get our real velocities from the equations (1.3) and (1.4), and we write them as following:

$$\omega_r = \frac{2v + \omega\mathbb{L}}{2\mathbb{R}} \quad (1.5)$$

$$\omega_l = \frac{2v - \omega\mathbb{L}}{2\mathbb{R}} \quad (1.6)$$

Those two velocities (v_l, and v_r) are the commands send to the robot

As an example if we take the linear velocity $v = 0$, and the angular velocity $\omega = constant$. We will find the corresponding angular wheel velocities ω_r, and ω_l as the following:

$$\omega_r = \frac{\omega\mathbb{L}}{2\mathbb{R}} \quad (1.7)$$

$$\omega_l = \frac{-\omega\mathbb{L}}{2\mathbb{R}} \quad (1.8)$$

In this case the robot will not move straight forward, but it will continue to spin around with a constant angular velocity.

I.3.Odometry

I.3.1.The position of the robot

We represent the state of the robot as (x, y, ϕ). The request here is how really we could get that state of the robot. There are many ways permitting us to obtain that, but absolutely we need sensors for such a task. We can use two types of sensors, the external sensors and the internals. The external sensor would be sensor measuring something in the environment. We can use for examples infrared sensors, ultraviolet sensors, vision sensors and laser scanner. Those sensors may indicate for us where the robot is situated in its environment. Other type of sensors we use is GPS that gives for us location position which we can obtain

-instantaneously from satellites with a good precision. But using GPS alone is not enough to know the position of the robot. The solution for that is to add the internal sensors which are embarked in the robot. For example we use to localize the position accelerometers and gyroscopes. For the orientation a compass…etc. another useful way is wheel encoders.

Basically we have tick counts to figure out how many revolutions the wheels are doing in certain amount of time. The wheel encoder will give us the distance has been crossed by each wheel. In the case the wheels are following an arc the velocity of the two wheels should be different, one is fast than the other. We illustrate that in the *Fig I*. 2

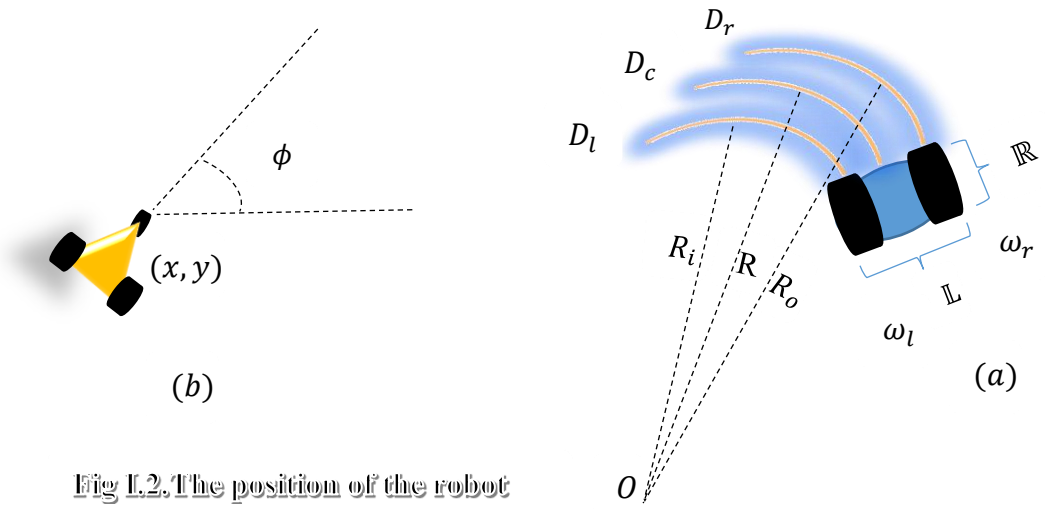


Fig I.2.The position of the robot

We have D_l is the distance the left wheel has turned. D_r is the distance the right wheel has crossed. The right wheel is turning quicker than the left wheel because it has to cross more distance in the same time, and as we know the two wheels belong to the same system. We have D_c is the distance the center of the robot has crossed, and it is described by the following relation:

$$D_c = \frac{D_r + D_l}{2} \qquad (1.9)$$

$$D_r = 2\pi\mathbb{R}\frac{\Delta tick_r}{N} = \mathbb{R}\omega_r dt \qquad (1.10)$$

$$D_l = 2\pi\mathbb{R}\frac{\Delta tick_l}{N} = \mathbb{R}\omega_l dt \qquad (1.11)$$

The equation (1.12) shows for us how far the wheels of the robot and its position have been moved after a period of time:

$$\begin{cases} x' = x + D_c \cos\phi \\ y' = y + D_c \sin\phi \\ \phi' = \phi + \dfrac{D_r - D_l}{\mathbb{L}} \end{cases} \qquad (1.12)$$

The wheel encoder will give the robot the only information concerning its position. Now we need to know what is D_r and D_l in order to know where the robot is. We assume that each wheel has N ticks per revolution. $2\pi\ degrees$ is N ticks.

The most wheel encoders give the total tick count since the beginning. What we measure is how many ticks since we have started the system up [1].

For both wheels we have $\Delta tick = tick' - tick$ (1.13) . We can get to the distance as we can express it in (1.14):

$$D = 2\pi\mathbb{R}\frac{\Delta tick}{N} \qquad (1.14)$$

That gives us a way of mapping tick on to distance as traveled.

To find a position at the next time when the robot will move on, also by depending on the equation (1.14) as we have seen about the velocities and the position, we can write:

$$x(t + dt) = x(t) + \dot{x}dt \qquad (1.15)$$

As a conclusion we can say a system depending only in wheel encoder will drift, it is very unprecise. It is not robust.

I.3.2.Kinematic equations

The robot takes a turning which has a ray R as central, also has an internal ray as R_{in}, and external ray as R_{ex}. We get the internal speed from the left wheel speed written as $v_{in} = v_l$. The external speed is also taken as the right wheel speed and we have $v_{ex} = v_r$. We can write the following equations:

$$\begin{cases} \dfrac{v_{ex}}{R_{ex}} = \dfrac{v_{in}}{R_{in}} \\ R_{ex} - R_{in} = \mathbb{L} \end{cases} \qquad (1.16)$$

With

$$R_{in} = R - \frac{\mathbb{L}}{2} \qquad (1.17)$$

$$R_{ex} = R + \frac{\mathbb{L}}{2} \qquad (1.18)$$

Where

$$\frac{v_{ex}}{v_{in}} = \frac{2R+\mathbb{L}}{2R-\mathbb{L}} = constante \qquad (1.19)$$

If we want the robot to describes a turn on the right. The relation between the speeds of the two wheels has to verify the following equations:

$$\frac{v_l}{v_r} = \frac{2R + \mathbb{L}}{2R - \mathbb{L}} \qquad (1.20)$$

For the left turn we have

$$\frac{v_r}{v_l} = \frac{2R+\mathbb{L}}{2R-\mathbb{L}} \qquad (1.21)$$

Using the central speed of the robot , we can write for the internal speed

$$v_{in} = \frac{2v}{1+\frac{2R+\mathbb{L}}{2R-\mathbb{L}}} \qquad (1.22)$$

For the external speed we can write:

$$v_{ex} = \frac{2v}{1+\frac{2R+\mathbb{L}}{2R-\mathbb{L}}} \frac{2R+\mathbb{L}}{2R-\mathbb{L}} \qquad (1.23)$$

With

$$v = \frac{v_{in}+v_{ex}}{2} \qquad (1.24)$$

For a right turn we have to check that

$$v_r = \frac{2v}{1+\frac{2R+\mathbb{L}}{2R-\mathbb{L}}} \qquad (1.25)$$

And

$$v_l = \frac{2v}{1+\frac{2R+\mathbb{L}}{2R-\mathbb{L}}} \frac{2R+\mathbb{L}}{2R-\mathbb{L}} \qquad (1.26)$$

For a left turn

$$v_l = \frac{2v}{1+\frac{2R+\mathbb{L}}{2R-\mathbb{L}}} \qquad (1.27)$$

And

$$v_r = \frac{2v}{1+\frac{2R+\mathbb{L}}{2R-\mathbb{L}}} \frac{2R+\mathbb{L}}{2R-\mathbb{L}} \qquad (1.28)$$

Those relations let us to control the mobile robot in order to describe a circle with a ray R with an average speed v which is a speed of robot's center.

In order to make the robot moving in straight line, we have just to set the velocity v to

$$v = v_r = v_l \qquad (1.29)$$

I.3.3. An error model for odometric position estimation

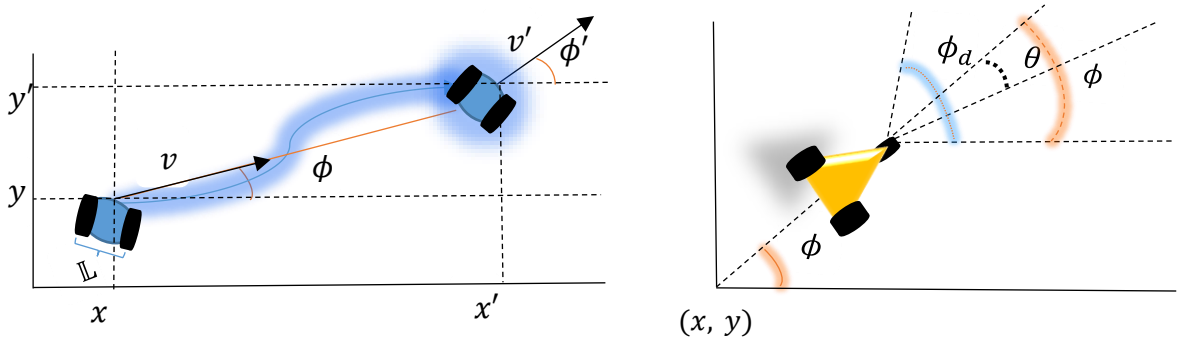


Fig I.3. an error model for odometric position estimation using unicycle differential drive robots

An error model and a processing method for *odometric* position estimation is given in [4]. For the position of the unicycle differential-drive robot as shown in the *Fig I.3*, we can take generally the position as $p = [x \quad y \quad \phi]^T$

In a discreet system with one period of sampling fixed the increments of distances browsed are:

$$\Delta x = \Delta s \cos\left(\phi + \frac{\Delta\phi}{2}\right) \qquad (1.30)$$

$$\Delta y = \Delta s \sin(\phi + \frac{\Delta\phi}{2}) \qquad (1.31)$$

$$\Delta\phi = \frac{\Delta s_r - \Delta s_l}{\mathbb{L}} \qquad (1.32)$$

$$\Delta s = \frac{\Delta s_r + \Delta s_l}{2} \qquad (1.33)$$

$(\Delta x, \Delta y, \Delta\phi)$ is the path browsed during the last interval of time.

$(\Delta s_r, \Delta s_l)$ are the distances browsed from the right and the left wheel, respectively.

$\mathbb{L}$ is the distances between the wheels of the robot.

The update of the pose $p'$is made as:

$$p' = \begin{bmatrix} x' \\ y' \\ \phi' \end{bmatrix} = p + \begin{bmatrix} \Delta s \cos(\phi + \frac{\Delta\phi}{2}) \\ \Delta s \sin(\phi + \frac{\Delta\phi}{2}) \\ \Delta\phi \end{bmatrix} \qquad (1.34)$$

While using the relations Δs and $\Delta\phi$ we have written previously. We get the equations of basis for updating the position $odometry$ for the unicycle differential-drive robot [26] [4]:

$$p' = \begin{bmatrix} x \\ y \\ \phi \end{bmatrix} + \begin{bmatrix} \frac{\Delta s_r + \Delta s_l}{2} \cos(\phi + \frac{\Delta s_r - \Delta s_l}{2\mathbb{L}}) \\ \frac{\Delta s_r + \Delta s_l}{2} \sin(\phi + \frac{\Delta s_r - \Delta s_l}{2\mathbb{L}}) \\ \frac{\Delta s_r - \Delta s_l}{\mathbb{L}} \end{bmatrix} \qquad (1.35)$$

We don't find interest for these equations unless when we deal with weak displacement. The more the displacement between two instants will be important, the more the evaluation of the position will not be precise. Because of the mistakes of integration of the uncertainties on the pose and mistakes of movement achieved during the increment (Δs_r, Δs_l) the mistake in based position on the integration of the $odometry$ increase by time.

I.4.Behavior-Based robotics

When a mobile robot move in any environment we can't predict how it will behave exactly, because the area where we chose to drive the robot is changing.

We may find obstacles or something may disturb the behavior of the robot in such unknown environment. For that the designer needs to develop a library for useful controllers. The controllers can do different things, like going to landmarks or avoiding obstacles. We can switch from one controller to another as the situation required that, like charging a battery when we see that the power is going down…etc.

I.4.1.Building a behavior

We assume that we have a differential-drive wheeled mobile robot driving at constant speed v_c . ϕ_d is the desired angle which is supposed the robot has to reach. ϕ is the actual angle of the robot, that also means its direction.

(x, y) is the position of the robot. The placement of the mobile robot wheels over the angles is traduced by:

$$\tan\phi = \frac{\dot{y}}{\dot{x}} \qquad (1.36)$$

The differential equations [42] are written as the following:

$$\begin{cases} \dot{x} = v_c \cos\phi \\ \dot{y} = v_c \sin\phi \\ \dot{\phi} = \omega = \frac{v_c}{L}\tan\theta \end{cases} \qquad (1.37)$$

We take the reference as $r = \phi_d$ (1.38)

The error is $e = \phi_d - \phi$ (1.39)

We have also the dynamics as $\dot{\phi} = \omega$ (1.40)

If we apply the PID controllers to the angular velocity then we got:

$$\omega = k_p e + k_i \int e d\tau + k_d \dot{e} \qquad (1.41)$$

k_p is the proportional gain, it respond to the actual error. Taking k_p large, the system will respond quickly. But taking k_p too large, that will induce oscillations.

k_i is the integral of the error. It will integrate up all the tiny tracking errors that we may have. And after a while this integral will become large enough that it pushes the system up to no tracking errors. Taking k_i too large may will induce oscillations.

k_d is the derivative gain, this makes the system very responsive. But can be a little bit over sensitive to noise.

This type of controller will not working, because we are dealing with angles. For example we can take:

$$\phi_d = 0, \qquad \phi = 10\pi \Longrightarrow e = -10\pi \qquad (1.42)$$

That is the same as zero radian. Taking the error too large will make the robot spinning around.

We have to insure that the error $e \in [-\pi, \pi]$ (1.43). For that we can use a trick and we write

$$e' = atan2(\cos(e), \sin(e)) \in [-\pi, \pi] \qquad (1.44)$$

I.4.2.Drive the robot to a desired position

If we have a differential drive robot. And using a unicycle model with constant velocity v_c. We use a PID controller to control the variation of the tracking error [1] and we write

$$\omega = PID(\dot{e}) \qquad (1.45)$$

We can see the desired position as it is shown in $Fig\ I.4$

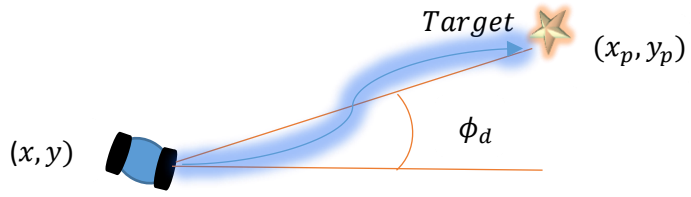


Fig I.4. driving a robot to a desired positon

We have $e = \phi_d - \phi$ (1.47)

And also for the simple model we can write:

$$\begin{cases} \dot{x} = v_c \cos\phi \\ \dot{y} = v_c \sin\phi \\ \dot{\phi} = \omega \end{cases} \qquad (1.46)$$

We can write

$$\phi_d = \arctan\left(\frac{y_p - y}{x_p - x}\right) \qquad (1.47) \quad [43]$$

We should have

$$\omega = k(\phi_d - \phi) \qquad (1.48)$$

In this case we are dealing with a proportional regulator using only the proportional gain. The result is that the robot may not go to the position which is supposed that it has to reach. As we said before that the tracking error should be fixed between $[-\pi, \pi]$. To deal with that problem and maintain the tracking error in the desired interval, we have made before a trick and we took the arctangent of the angles in order to stay steady in that interval. But the problem is not fully resolved, because again the robot may not doing what we want it to do for us, and it still little bit spiraling around the position we have fixed before. The problem may be the robot is turning enough faster, because we have taken its speed as a constant, or we should slow down when it is close enough to an obstacle. As we have seen we need to control v, and ω. But we need to make the gain high enough which permit us in the end to make the robot reaching its final desired position.

I. 5. Obstacle avoidance

We have to avoid driving into obstacles as we see in the *Fig I*. 5

The direction we want the robot to steer into is not too pretty much clear. The first goal is to run away from the obstacles, and the second choice is to choose the direction the robot has to follow. As the *Fig I*. 5 shows it, we can go to the desired position, and the goal for the robot has to reach here is the shining star. Our first choice is to take the direction of the angle

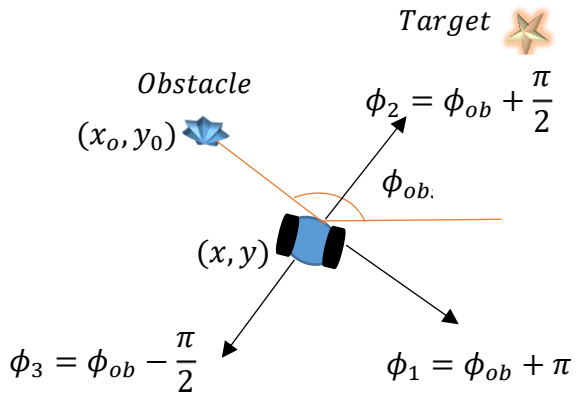


Fig I.5. Avoiding obstacles

ϕ_1.the point (x_o, y_o) is the position of the obstacle. And the position of the robot is (x, y). We have $\phi_{ob} = \phi_{obstacle}$, and our condition should be $\phi_{ob} \epsilon [-\pi, \pi]$ (1.49).

$$\phi_{obs} = \text{atan}\left(\frac{y_o - y}{x_o - x}\right) \quad (1.50)$$

$$\phi_1 = \phi_{ob} + \pi \quad (1.51)$$

That will help us to make the robot run away from the obstacle much easily. Now we choose another direction by adding or subtracting $\frac{\pi}{2}$

$$\phi_2 = \phi_{ob} + \frac{\pi}{2} \quad (1.52)$$

$$\phi_3 = \phi_{ob} - \frac{\pi}{2} \quad (1.53)$$

If the robot is going into the direction of ϕ_2, that means it is going straight closer to the position of the goal which is here the shining star.

And if the chosen direction is ϕ_3, that means the robot is going in the opposite direction of the area where it is supposed to find its goal. If we choose to go straight into the goal or just trying to avoid the obstacles, that is called hard switches.

The other choice called blended behavior which is the combination to choose between the direction of the goal in one part and the direction of the obstacle in the other. The other example as we have seen is going perpendicularly to the obstacle and being closer to the goal, that is too a blended behavior. The different choices we have taken as the behavior for the robot are basically called the arbitration mechanisms.

As a conclusion we can say that choosing the hard switches tasks is much easier for the robot to do in analysis point view than the other choices. But the performance required from the robot to do in that case is not quite guaranteed.

I.6.Making a simple robot

I.6.1. Introduction

We need to show the behaviors of our robot in a systematic way. All the models we can get by trying to build any controller is just an approximation. We can have a different characteristics to our developed model class, and that is quite general enough, In simple way in order to have a behavior to understand , Expressive enough, and useful enough. In other words we need our robot to accomplish its task as we want it to do for us.

I.6.2. Linear systems

I.6.2.1. controlling a point mass

We consider our robot as a point mass in a line, and we have to control directly its acceleration which has been taken as the second derivative of its position, we name it as p.

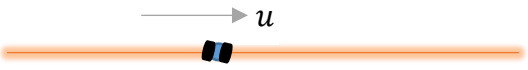


Fig I. 6. Controlling a point of mass

We have the acceleration as a second derivative of the position. We can write

$$\ddot{p} = u \qquad (1.54)$$

To have state space form we can take:

$$\begin{cases} x_1 = p \\ x_2 = \dot{p} \end{cases} \Rightarrow \begin{cases} \dot{x}_1 = \dot{p} = x_2 \\ \dot{x}_2 = \ddot{p} = u \end{cases} \qquad (1.55)$$

$$\dot{x} = \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} x_2 \\ u \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u \qquad (1.56)$$

$$y = p = x_1 = [1 \quad 0]\, x \qquad (1.59); \qquad x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \qquad (1.57)$$

In general way, if we introduce a notion of a matrix, we can get

$$A = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad C = [1 \quad 0] \qquad (1.58)\,.$$

In the end we can write

$$\begin{aligned} \dot{x} &= Ax + Bu \\ y &= Cx \end{aligned} \qquad (1.59)$$

In two dimensional point mass we can write:

$$\begin{cases} \ddot{p}_x = u_x \\ \ddot{p}_y = u_y \end{cases} \qquad (1.60)$$

$$\begin{aligned} x_1 &= p_x \\ x_2 &= \dot{p}_x \\ x_3 &= p_y \\ x_4 &= \dot{p}_y \\ u_1 &= u_x \\ u_2 &= u_y \\ y_1 &= p_x \\ y_2 &= p_y \end{aligned} \qquad (1.61)$$

The matrixes will became as the following:

$$A=\begin{bmatrix}0&1&0&0\\0&0&0&0\\0&0&0&1\\0&0&0&0\end{bmatrix},\quad B=\begin{bmatrix}0&0\\1&0\\0&0\\0&1\end{bmatrix},\quad C=\begin{bmatrix}1&0&0&0\\0&0&1&0\end{bmatrix} \qquad (1.62)$$

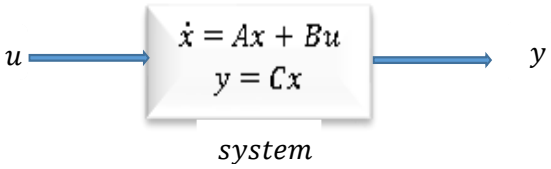
The state-space form will still the same, and the system is called Linear Time-Invariant (LTI) system.

For the dimensions we have

$$\begin{cases}x\in\Re^n\\u\in\Re^m\\y\in\Re^p\end{cases}\Longrightarrow\begin{matrix}A:n\times n\\B:n\times m\\C:p\times n\end{matrix} \qquad (1.63)$$

I.6.3.State-Space models

The general form of the model is given as the following:



We have x is taken as a state, u is a control signal taken as an input, and y as an output.

A, B, C are the matrixes of the system. Ax is the physics of the system, and is given to us by the laws of physics. B tell us how the input affects the state, that means too how the actuator behave. C shows us how the sensors behave or what they are measured.

If we take the car model and try to measure its velocity, we will have then:

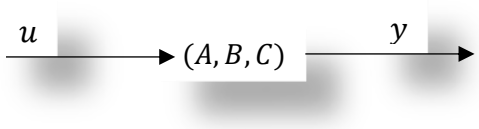
$$\dot{v}=\frac{\beta}{m}u-\rho v \qquad (1.64)$$

We take $x=v$, for the system we take $A=-\rho$, $B=\frac{\beta}{m}$ and $C=1$ (1.65)

That will give us a one dimensional simple system.

Now if we take both the position and the velocity, we can turn a dynamical equation into a system in state space form:

$$\dot{v} = \frac{\beta}{m}u - \rho v \Longrightarrow \begin{cases} \dot{x} = Ax + Bu \\ y = Cx \end{cases} \qquad (1.66)$$



$$x = \begin{bmatrix} p \\ v \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, y = p = x_1 \ , \ (A, B, C) = \left(\begin{bmatrix} 0 & 1 \\ 0 & -\rho \end{bmatrix} , \begin{bmatrix} 0 \\ \frac{\beta}{m} \end{bmatrix}, [1 \quad 0] \right) \qquad (1.67)$$

$$\begin{cases} \begin{bmatrix} \dot{p} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & -\rho \end{bmatrix} \begin{bmatrix} p \\ v \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{\beta}{m} \end{bmatrix} u \\ y = [1 \quad 0] \begin{bmatrix} p \\ v \end{bmatrix} \end{cases} \qquad (1.68)$$

We have $\dot{p} = \dot{x}_1 = x_2, \ \ddot{p} = \dot{x}_2 = -\rho x_2 + \frac{\beta}{m}u$ (1.69)

In the end we can write:

$$\begin{cases} \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & -\rho \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{\beta}{m} \end{bmatrix} u \\ y = [1 \quad 0] \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \end{cases} \qquad (1.70)$$

For the unicycle robot we have:

$$\begin{cases} \dot{x} = v_c \cos\phi \\ \dot{y} = v_c \sin\phi \\ \dot{\phi} = \omega \end{cases} \qquad (1.71)$$

As we can see the system is not linear, and if we take a small angles, that will give us:

$$\phi \ll 1 \Longrightarrow \cos\phi = 1, \quad \sin\phi = \phi \qquad (1.72)$$

$$\begin{cases} \dot{x} = v_c \\ \dot{y} = v_c\phi \\ \dot{\phi} = \omega \end{cases} \tag{1.73}$$

Here too we have a multiplication $v_c\phi$ that means the system is not linear. In this case the simplification is not enough to get a linear system.

I.6.4.Linearization

We have the most system in the universe are not linear. The system behave as linear around an equilibrium point.

We take a general example for a nonlinear system, and we write:

$$\dot{x} = f(x,u), \qquad y = h(x) \tag{1.74}$$

The goal here is to find a linear model around an operating point (x_0, u_0)

We have $x = x_0 + \delta x, \;\; u = u_0 + \delta u$ (1.75)

The new equations of motion become:

We use $Taylor\ expansion$

$$\begin{aligned} \delta\dot{x} = \dot{x} - \dot{x}_0 = \dot{x} &= f(\dot{x}_0 + \delta x, u_0 + \delta u) \\ &= f(x_0,u_0) + \underbrace{\frac{\partial f}{\partial x}(x_0,u_0)}_{A}\delta x + \underbrace{\frac{\partial f}{\partial u}(x_0,u_0)}_{B}\delta u + H.O.T \rightarrow (Higher\ Order\ Terms) \\ y = h(x_0,\delta x) &= h(x_0) + \underbrace{\frac{\partial h}{\partial x}(x_0)}_{C}\delta x + H.O.T \end{aligned} \tag{1.76}$$

When δx and δu are large, we take account of $H.O.T$ otherwise that doesn't matter.

If we take:

$$f(x_0,u_0) = 0, \; h(x_0) = 0\,, \; A = \frac{\partial f}{\partial x}(x_0,u_0), \;\; B = \frac{\partial f}{\partial u}(x_0,u_0), \;\; C = \frac{\partial h}{\partial x}(x_0) \tag{1.77}$$

Then we can write:

$$\begin{matrix} \dot{\delta}x = A\delta x + B\delta u \\ y = C\delta x \end{matrix} \quad (1.78)$$

I.6.4.1. Computation of the Jacobians

$$x \in \Re^n, u \in \Re^m, y \in \Re^p \quad f = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{bmatrix}, \quad h = \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_m \end{bmatrix} \quad (1.79)$$

$$\mathrm{A} = \frac{\partial f}{\partial x} = \underbrace{\begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & & \frac{\partial f_2}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix}}_{n \times n}, B = \frac{\partial f}{\partial u} = \underbrace{\begin{bmatrix} \frac{\partial f_1}{\partial u_1} & \cdots & \frac{\partial f_1}{\partial u_m} \\ \frac{\partial f_2}{\partial u_1} & & \frac{\partial f_2}{\partial u_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial u_1} & \cdots & \frac{\partial f_n}{\partial u_m} \end{bmatrix}}_{n \times m}, C = \frac{\partial h}{\partial x} = \underbrace{\begin{bmatrix} \frac{\partial h_1}{\partial x_1} & \cdots & \frac{\partial h_1}{\partial x_n} \\ \frac{\partial h_2}{\partial x_1} & & \frac{\partial h_2}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial h_p}{\partial x_1} & \cdots & \frac{\partial h_p}{\partial x_n} \end{bmatrix}}_{p \times n} \quad (1.80)$$

I.6.4.2.Application to the unicycle robot

We have the differential equations (1.84)

$$\begin{cases} \dot{x} = v\cos\phi \\ \dot{y} = v\sin\phi \\ \dot{\phi} = \omega \end{cases} \quad (1.81)$$

Now look for the state model, and we can put the variables as the following:

$$\begin{cases} x = x_1 \ , \ y = x_2 \ , \ \phi = x_3 \\ y_1 = x_1 \ , \ y_2 = x_2 \ , \ y_3 = x_3 \\ u_1 = v, \ u_2 = \omega \\ (x_0, u_0) = (0,0) \end{cases} \quad (1.82)$$

$$A = 0 \ , \quad B = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix}, \quad C = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (1.83)$$

We replace the matrixes A, B, and C in the system then we get

$$\begin{cases} \dot{x} = Ax + Bu \\ y = Cx \end{cases} \quad (1.84)$$

$$\begin{cases} \begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{pmatrix} = 0.\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} \\ \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \end{cases} \quad (1.85)$$

As a result we get $\dot{x}_2 = 0$, we have also x_2 is the y direction. If we point our robot straight to the x direction, then apparently we can't make the robot drive in the y direction. The linearization here didn't give what we want because we lose control if we point the robot to y direction. As a result we can say when the linearization work that is useful to our robots.

I.6.5.Stability

To get some intuition about what is going on, we start to use the scalar systems.

$$\dot{x} = ax \Longrightarrow x(t) = e^{at}x(0) \quad (1.86)$$

For the matrix A, we write

$$\dot{x} = Ax \Longrightarrow x(t) = e^{At}x(0) \quad (1.87)$$

We can write

$$Av = \lambda v \quad (1.88)$$

λ is an eigenvalue $\in C$.

v is a eigenvector $\in \Re^n$

The eigenvalue tell us how the matrix A acts in different directions (eigenvectors).

I.6.5.1.Asymptotic stability

We write $a < 0 \Longrightarrow x(t) \longrightarrow 0, \forall\, x(0)$ (1.89)

The system is asymptotically stable if x go to zero for all initial condition. We take in this case a strictly negative.

For the matrix A we have the system is asymptotically stable if and only if

$$Re(\lambda) < 0, \forall \lambda \in eig(A) \quad (1.90)$$

I.6.5.2.Unstable system

For the a scalar

$$a > 0 \Longrightarrow \exists\, x(0): \|x(t)\| \longrightarrow \infty \quad (1.91)$$

That means there exist initial condition for which the systems blows up. In that case we have a strictly positive

For the A matrix we have the system is unstable if

$$\exists \lambda\ \in eig(A) :\ Re(\lambda) > 0 \quad (1.92)$$

I.6.5.3.The system is critically stable

For that case we have $a = 0$. The system doesn't go to zero or blow up either. But it goes between them.

For the matrix A we have:

$$Re(\lambda) \leq 0, \forall \lambda \in eig(A) \quad (1.93)$$

CHAPTER II. Visual *Odometry* for robot's navigation

II.1.Introduction

Visual *Odometry* is the process of determining a visual sensor orientation and position in three dimensional space from a sequence of images, or simply put motion estimation using visual information only. So many living creatures, especially for us as human being, the eyes play a basic role for any attempt to localize ourselves in the environment situated around us. We can also see and distinguish different shapes in our vision field. But our eyes can betray us and might has difficulties when it is something about the real scale of the shapes we perceive. The scale factor is also called "metric", which can let us to transform any unknown distance to a measured distance. The distances are measured by meter. The robots are using cameras for such a localization. As the objects get far the smaller they are in our eyes.

If we don't care about the metrical problems. The approaches based on the vision technics are interesting, because they let us to obtain a complete results concerning the localization and the estimation of the geometry of the environment with a good precision, what is called visual *odometry* technics.

The successive images of a camera are the basic information in visual *odometry*. When such images ones are treated, they will serve to calculate the displacement of the camera between each two screen shot. We look for the similarities among points in a many successive images the camera has got. Those similarities among points constitute the input of the system, which allows to obtain the relative displacement of the camera from the first image to the second one.

II.2.Thin lens camera model

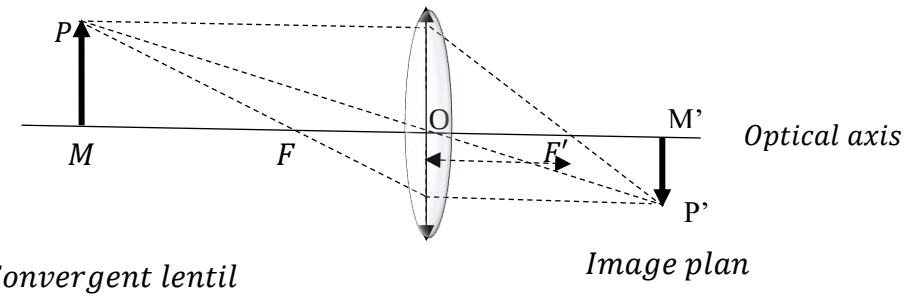


Fig II.1. Convergent lentil

We have rays parallel to the optical axis are deflected through a point at distance λ which is a focal length. The rays passing through the optical center O are not deflected, as it is shown in [6], and we can write a fundamental equation of a thin lens [23] as following:

$$\frac{1}{OM} + \frac{1}{OM'} = \frac{1}{OF} = \frac{1}{OF'} \qquad (2.1) \qquad\qquad \frac{M'P'}{MP} = \frac{OM'}{OM} \qquad (2.2)$$

II.3.Pinhole camera model

II.3.1.Definition

The pinhole camera model is the basic camera model used in computer vision. Its name originates from the concept of pinhole camera and it models perspective projections.

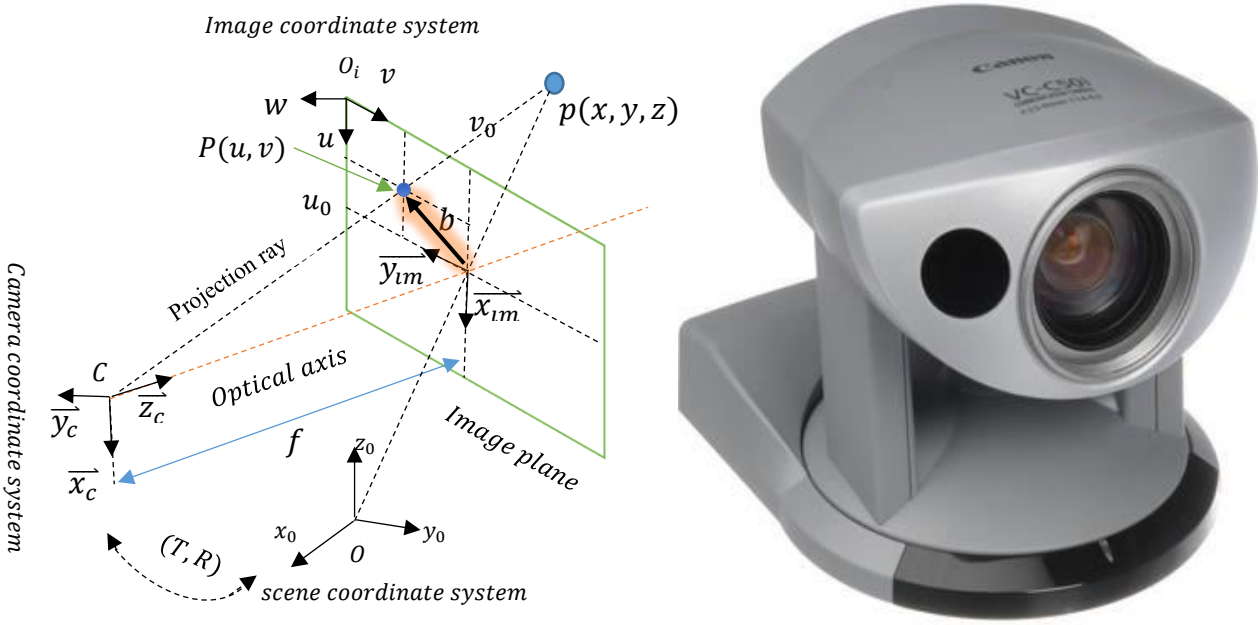
The pinhole camera model describes the mathematical relationship between the coordinates of a 3D point and its projection onto the image plane of an ideal pinhole camera [12], where the camera aperture is described as point and no lenses are used to focus light. The model does not include, for example, geometric distortions or blurring of unfocused objects caused by lenses and finite sized apertures. It also does not take into account that most practical camera have only discrete image coordinates. This means that the pinhole camera model can only be used as a first order approximation of mapping from a 3D scene to a 2D image.

Its validity depends on the quality of the camera and, in general decrease from the center of the image to the edges as lens distortion effects increase.

Some of the effects that the pinhole camera does not take into account can be compensated, for example by applying suitable coordinate transformation. A pinhole camera model has been illustrated in [12] [16].We can show all of that in the illustrated Fig II.2.

A camera could be approximated by a projective model, often called pinhole model.

The simplest representation of a camera is a light sensible surface (sensor), see([22],



(a) *pinhole camera model* (b) *Canon camera* $VC-C50i$ [39]

Fig II. 2. The camera

[12]): an image plane, a lens (projective projection) at a given position and orientation in space. in the pinhole model as shown in the Fig II.2 (a), when thin lens widness is neglected. All rays pass trough optical center which is the center of projection we chose here as C. The points are projected therefore on the plan according to a perspective projection. From the scene we take a point p with coordinates $p^{Rc} = (x, y, z)^T$ in the R_c reference mark. It is projected on the plan in one P point of metric coordinates $(u, v)^T$, and we can find P using similar triangles[3] as:

$$\frac{f}{z} = \frac{u}{x} = \frac{v}{y} \qquad (2.3)$$

$$u = \frac{f}{z}x \qquad (2.4)$$

$$v = \frac{f}{z}y \qquad (2.5)$$

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} \frac{f}{z} & 0 & 0 \\ 0 & \frac{f}{z} & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} \qquad (2.6)$$

Using homogeneous coordinates for P we can write this as

$$\begin{pmatrix} u \\ v \\ w \end{pmatrix} = \begin{pmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} \qquad (2.7)$$

We can verify that indeed generates the point $P(u, v, w) = (\frac{f}{z}x, \frac{f}{z}y, 1)$. Note that p is still not in homogeneous coordinates.

Now, if the origine of the 2D image coordinate system does not coincide with where the z axis intersects the image plane, we need to translate P to the desired origine. We difine this translation by (t_u, t_v). Hence, now (u, v) is

$$u = \frac{f}{z}x + t_u \qquad (2.8)$$

$$v = \frac{f}{z}y + t_v \qquad (2.9)$$

This can be expressed in similar form as equation (2.7) as

$$\begin{pmatrix} u \\ v \\ w \end{pmatrix} = \begin{pmatrix} f & 0 & t_u \\ 0 & f & t_v \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} \qquad (2.10)$$

Now, in the equation(2.10), P is expreced in inches since this is a camera image, we need to express it in inches. For this we will need to know the resolution of the camera in pixels/inch. If the pixels are square the resolution will be identical in both u and v directions of the camera image coordinates. However for more general .case, we assume more rectangle pixels with resolution m_u and m_v pixels/inch in u and v direction respectively (I). Thus

$$u = m_u \frac{f}{z} x + m_u t_u \qquad (2.11)$$

$$v = m_v \frac{f}{z} y + m_v t_v \qquad (2.12)$$

This can be expressed in matrix form as

$$\begin{pmatrix} u \\ v \\ w \end{pmatrix} = \begin{pmatrix} m_u f & 0 & m_u t_u \\ 0 & m_v f & m_v t_v \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} \alpha_x & 0 & u_0 \\ 0 & \alpha_y & v_0 \\ 0 & 0 & 1 \end{pmatrix} p = Kp \qquad (2.13)$$

We have K only depends on the intrinsic camera parameters like its focal length, principal axis and thus defines the intrinsic parameters of the camera. Sometime K has a skew parameter s given by

$$K = \begin{pmatrix} \alpha_x & s & u_0 \\ 0 & \alpha_y & v_0 \\ 0 & 0 & 1 \end{pmatrix} \qquad (2.14)$$

This usually comes in if the image coordinate axis u and v are not orthogonal to each other. We note that K is an upper triangular 3×3 matrix. This is usually called the intrinsic parameter matrix for the camera.Now if the camera does not have its center of projection at $(0,0,0)$ and is oriented in arbitrary fashion (not necessarily z perpendicular to the image plane), then we need rotation and translation to make the camera coordinate system coincide with the configuration in Fig II.2.

We take $T(T_x, T_y, T_z)$ as the camera translation of (x, y, z) coordinate. R is the 3×3 rotation matrix, the rotation applied to coincide the principal axis with z axis. We have E as the 3×4 matrix formed by first applying the translation followed by the rotation.

$$E = (R \mid RT) \qquad (2.15)$$

Called the extrinsic parameter matrix. So, the complet camera transformation can be represented as it is shown in (2.16).

$$K(R \mid RT) = (KR \mid KRT) = KR(I \mid T) \qquad (2.16)$$

Hence p the projection of P is given by : $p = KR(I \mid T)P = CP$ (2.17)

C is a 3×4 matrix usuaslly called the complete camera calibration matrix. We not that since C is a 3×4 we need p to be in $4D$ homogeneous coordinates and P derived by CP will be in $3D$ homogeneous coordinates. The exact $2D$ location of the projection on the camera image plane will be obtained by dividing the first two coordinates of P by the third.

II.4.Interaction Matrix

According to the $Fig\ II.2$. We have the absolute reference frame:$\mathcal{F}_O$: $\{O, \overrightarrow{x_0}, \overrightarrow{y_0}, \overrightarrow{z_0}\}$.The camera reference frame: $\mathcal{F}_C$: $\{O_C, \overrightarrow{x_C}, \overrightarrow{y_C}, \overrightarrow{z_C}\}$. The image plane reference frame: $\mathcal{F}_i$: $\{O_i, \vec{u}, \vec{v}\}$

We take p as a point of the space, with its coordinates $p = (x, y, z)^T$ in R_c , it is projected on the plan in one P point of coordnates $(u, v)^T$ in the picture. According to the equation of projection (2.6). after some deffirentiating the equation (2.6), we get the variations in the picture of the $u(t)$ and $v(t)$ of P in relation to the speed of $V_{p/R_c}^{R_c}$ of the p point.

$$\begin{cases} \dot{u}(t) = \left(\frac{f}{z(t)} \quad 0 \quad \frac{-x(t)f}{z(t)^2}\right) V_{p/R_c}^{R_c}\,(\mathrm{t}) \\ \dot{v}(t) = \left(0 \quad \frac{f}{z(t)} \quad \frac{-y(t)f}{z(t)}\right) V_{p/R_c}^{R_c}\,(\mathrm{t}) \end{cases} \qquad (2.18)$$

In other way we can write:

$$\begin{pmatrix} \dot{u}(t) \\ \dot{v}(t) \end{pmatrix} = \begin{pmatrix} \frac{f}{z} & 0 & -\frac{u}{z} \\ 0 & \frac{f}{z} & -\frac{v}{z} \end{pmatrix} \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{pmatrix} \qquad (2.19)$$

Using (2.6) and (2.19) then we get:

$$\begin{pmatrix} \dot{u}(t) \\ \dot{v}(t) \end{pmatrix} = \begin{pmatrix} \frac{f}{z} & 0 & \frac{-x(t)f}{z(t)^2} \\ 0 & \frac{f}{z} & \frac{-y(t)f}{z(t)} \end{pmatrix} \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{pmatrix} \qquad (2.20)$$

We have a point p belonging to a target we concider as immobile. That give us a stationary point while the camera is enlivened of a transfer speed $\vec{V}_{C/R_o}(t)$ and of a rotational speed $\vec{\Omega}_{R_c/R_o}(t)$. We can write the speed of p as

$$\vec{V}_{p/R_c}(t) = -\vec{V}_{C/R_o}(t) - \vec{\Omega}_{R_c/R_o}(t) \times \overrightarrow{CP} \qquad (2.21)$$

In other way we can say that the velocity $(\dot{x}, \dot{y}, \dot{z})$ of appoint p in frame $\mathcal{F}_C$ is actually due to the roto-translation(V, Ω) of the camera (p is assumed fixed in $\mathcal{F}_0$).

The kinematic relation between $(\dot{x}, \dot{y}, \dot{z})$ and (V, Ω) is:

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{pmatrix} = -V - \Omega \times \begin{pmatrix} x \\ y \\ z \end{pmatrix} \qquad (2.22)$$

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{pmatrix} = \begin{pmatrix} -1 & 0 & 0 & 0 & -z & y \\ 0 & -1 & 0 & z & 0 & -x \\ 0 & 0 & -1 & -y & x & 0 \end{pmatrix} \begin{pmatrix} V \\ \Omega \end{pmatrix} \qquad (2.23)$$

Using (2.22) we rewrite (2.18) as

$$\begin{pmatrix} \dot{u}(t) \\ \dot{v}(t) \end{pmatrix} = L_{uv}\big(u(t), v(t), z(t)\big) T^{R_c}_{C/R_o}(t) \qquad (2.24)$$

By replacing (2.23) in (2.19) we get:

$$\begin{pmatrix} \dot{u}(t) \\ \dot{v}(t) \end{pmatrix} = \begin{pmatrix} -\frac{f}{z} & 0 & \frac{u}{z} & \frac{uv}{f} & -\left(f + \frac{u^2}{f}\right) & v \\ 0 & -\frac{f}{z} & \frac{v}{z} & f + \frac{v^2}{f} & -\frac{uv}{f} & -u \end{pmatrix} \begin{pmatrix} V \\ \Omega \end{pmatrix} \qquad (2.25)$$

With $p =$ point (feature), f assumed to be known.

The value z is the depth of the three dimensional point expressed in the camera frame.

$L_{uv}\big(u(t), v(t), z(t)\big)$ is the matrix of interaction of the point, joining the variations of the

-coordinates of one point in the picture to the mechanical torque of the camera [6] [12] [13].

The matrix of interaction of one p point depends on the composing z in the reference mark camera [12].

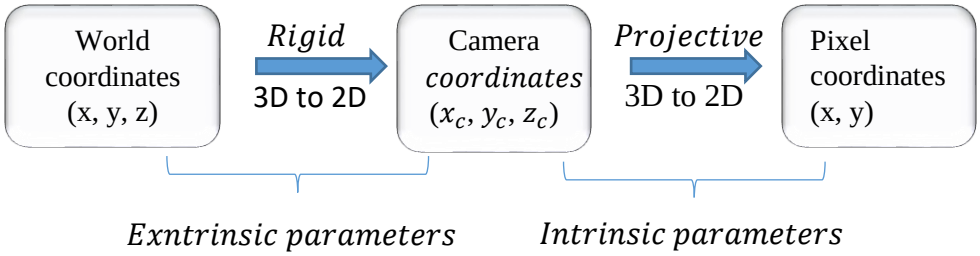
II.5.Camera calibration

II.5.1.Definition

To calibrate a camera consists in estimating the parameters of the model that have been chosen to represent it. It is a problem of parametrical estimation. Geometric camera calibration, also referred to as camera resection, estimates the parameters of a lens image sensor of an image or video camera. We can use these parameters to correct for lens distortion, measure the size of an object in world units, or determine the location of the camera in the scene. This tasks are used in applications such as machine vision to detect and measure objects, in robotics, for navigation systems, and 3D scene reconstruction. The camera parameters include intrinsic, extrinsic, and distortion coefficients.

The calibration algorithm calculates the camera matrix using the extrinsic and intrinsic parameters. The extrinsic parameters represent a rigid transformation from 3D world coordinate system to 3D camera‘s coordinate system. The intrinsic parameters represent a projective transformation from the 3D camera’s coordinates into the 2 D image coordinates.

II.5.2.Camera calibration parameters



II.5.2.1.Extrinsic parameters

The extrinsic parameters consist of rotation, R, and a translation, t. the origin of the camera's coordinate system is at its optical center and its $x-$ axis and $y-$ axis define the image plane.

II.5.2.2.Intrinsic parameters

The intrinsic parameters include the focal length, the optical center, also known as the principal point, and the skew coefficient.

II.5.2.3.Perspective transformation using homogeneous coordinates

$$b = \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \sim \underbrace{\overbrace{\begin{pmatrix} \alpha_x & s & u_0 \\ 0 & \alpha_y & v_0 \\ 0 & 0 & 1 \end{pmatrix}}^{Intrinsic\ camera\ parameters} \underbrace{\overbrace{\begin{pmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{pmatrix}}^{Extrinsic\ camera\ parameters} \overbrace{\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}}^{World\backslash scene\ coordinate\ system}}_{Camera\ coordinate\ system}}_{Image\ coordinate\ system} \qquad (2.26)$$

A transformation matrix is used which includes a rotation and transformation [36]. That matrix is written as:

$$T = \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{pmatrix} \qquad (2.27)$$

k is the matrix of intrinsic parameters of the camera, and we write:

$$k = \begin{pmatrix} \alpha_x & s & u_0 \\ 0 & \alpha_y & v_0 \\ 0 & 0 & 1 \end{pmatrix} \quad see\ (2.14)$$

We can write:

$$\alpha_x = m_u f \qquad (2.28)$$

$$\alpha_y = m_v f \qquad (2.29)$$

$$u_0 = m_u t_u \qquad (2.30)$$

$$v_0 = m_v t_v \qquad (2.31)$$

α_x , α_y are the horizontal and vertical focal lengths respectively, and also we have u_0 , v_0 are the x, y coordinates of the camera projection center see (I).

II.6.Obstacle avoidance using algorithms of vision

For the robot, avoiding any obstacle requires to know its absolute distance. The estimation of the relative distances is generally sufficient. Using monocular vision is a good way for such an estimation, just we have to do a good interpretation for the indications we use. Geometrically speaking, the optical flux is the projection of the motion of the objects on the camera screen, see [5]. Generally a camera is fixed on the mobile robot who moves and that the other objects of the environment are immobile. The apparent movement is described by the displacement of the robot which we consider its motion in a horizontal plan, and so the fixed camera on it. But in real world there may will be vibrations, and the optical access will be affected too. In the figure II.3, x is the optical. The observed object in the point O.

The camera is fixed in the origin R of the Euclidean reference with a directory vector $(\overrightarrow{u_x}, \overrightarrow{u_y}, \overrightarrow{u_z})$. The coordinate of the point O in the reference mark are (x, y, z). $\vec{V}$ is the linear drift of the robot. $\overrightarrow{\omega}$ is the angular velocity. $D = \left\|\overrightarrow{RO}\right\|$ is the distance between the camera and the object. δ is the angle between the optical axis and the direction of the displacement of the robot. The optic flux is represented by the angular velocity $\vec{F}$, obvious movement of the object O since the point R.

We can decompose it in a component $\overrightarrow{F_T}$ that comes from the transfer of the robot. And $\overrightarrow{F_R}$ that comes from its rotation. $\vec{F} = \overrightarrow{F_T} + \overrightarrow{F_R}$ (2.32)

$$\overrightarrow{F_R} = -\left(\vec{\omega} + \dot{\delta}\right)\overrightarrow{u_z} \qquad (2.33)$$

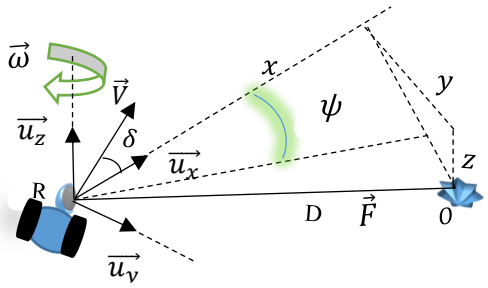


Fig II. 3. Displacement of the robot generating optical flux

$\overrightarrow{F_R}$ depend only on the speeds of the robot and the camera which takes place only in the horizontal plan.

The obvious movement bound to the transfer is situated in the plan defined by the vectors $\vec{V}$ and $\overrightarrow{RO}$:

$$\overrightarrow{F_T} = \frac{\vec{V} \wedge \overrightarrow{RO}}{D^2} \qquad (2.34)$$

The total obvious movement is given therefor by the formula shown in (2.35):

$$\vec{F} = -(\vec{\omega} + \dot{\delta})\overrightarrow{u_z} + \frac{\vec{V} \wedge \overrightarrow{RO}}{D^2} \qquad (2.35)$$

We have only the part bound to the movement of transfer depends on the distance. Only that part may interest us when it is about obstacle avoidance.

Now we can express the displacement of the point O in the reference mark $(R, \overrightarrow{u_x}, \overrightarrow{u_y}, \overrightarrow{u_z})$ as the following:

When we consider only the movement of translation $\vec{V}$ then we get:

$$\begin{aligned} \dot{x} &= -V\cos\delta \\ \dot{y} &= -V\sin\delta \\ \dot{z} &= 0 \end{aligned} \qquad (2.36$$

Now we consider only the movement of the rotation and we can write:

$$\begin{aligned} \dot{\psi} &= -(\omega + \dot{\delta}) \\ \dot{z} &= 0 \\ \dot{D} &= 0 \end{aligned} \qquad (2.37)$$

We express x and y according to ψ and we can write:

$$\begin{aligned} x &= \sqrt{D^2 - z^2}\cos\psi \\ y &= \sqrt{D^2 - z^2}\sin\psi \end{aligned} \qquad (2.38)$$

By taking the derivatives of these expressions we can get

$$\begin{aligned} \dot{x} &= -\sqrt{D^2 - z^2}\,\dot{\psi}\sin\psi \\ &= (\omega + \dot{\delta})y \\ \dot{y} &= \sqrt{D^2 - z^2}\dot{\psi}\cos\psi \\ &= -(\omega + \dot{\delta})x \end{aligned} \qquad (2.39)$$

In the end by taking account of the translation and the rotation we can get the final relations as shown in (2.40):

$$\begin{aligned} \dot{x} &= -V\cos\delta + (\omega + \dot{\delta})y \\ \dot{y} &= -Vsin\delta - (\omega + \dot{\delta})x \\ \dot{z} &= 0 \end{aligned} \qquad (2.40)$$

For the distance of the obstacles we have: $D^2 = x^2 + y^2 + z^2$ (2.41)

By deriving that relation we can get:

$$\begin{aligned} 2D\dot{D} &= 2x\dot{x} + 2y\dot{y} + 2z\dot{z} \\ \dot{D} &= \frac{x}{D}(-V\cos\delta + (\omega + \dot{\delta})y) + \frac{y}{D}(-V\sin\delta - (\omega + \dot{\delta})x) \end{aligned} \qquad (2.42)$$

Finally we can write:

$$\dot{D} = -\frac{V}{D}(x\cos\delta + y\sin\delta) \qquad (2.43)$$

II.6.1.Spherical projection

The projection is gotten in projecting the picture on a spherical surface. It is the case of the human eye. This projection is adapted therefor when we try to reproduce to best the human vision.

In the next figure we use α as an angle between the direction of the optic axis and the direction of the object, and β is the angle between the formed plan by $(\overrightarrow{RO}, \overrightarrow{u_x})$ and the horizontal plan. What we observe really on a spherical projection is the length have $\alpha_i = \lambda\alpha$ where λ is a focal length of the objective. The geometric properties being the same to the factor near. We use here directly the angle α.

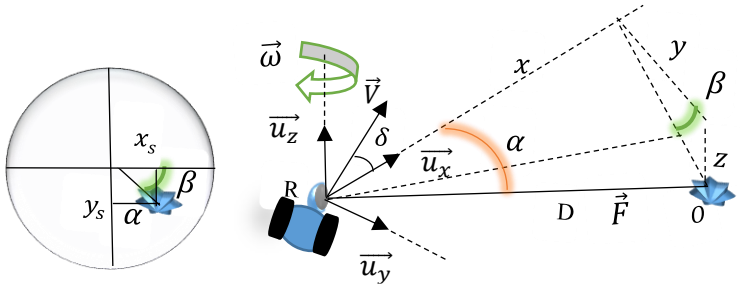


Fig II. 4. Representation of the picture discerned in spherical coordinates

The optic flux is represented by the spherical coordinates by the following relations:

$$\alpha = \sqrt{{x_s}^2 + {y_s}^2}$$
$$\beta = \tan^{-1}\left(\frac{y_s}{x_s}\right) \qquad (2.44)$$

From the figure we can get:

$$x = D\cos\alpha$$
$$y = D\sin\alpha\cos\beta \qquad (2.45)$$

Also we have:

$$\cos\alpha = \frac{x}{D}$$
$$\tan\beta = \frac{z}{y} \qquad (2.46)$$

To get the representation of the optic flux in spherical coordinates. It is sufficient to derive these two last expressions.

First we start with $\cos\alpha = \frac{x}{D}$ (2.47)

$$\begin{aligned}
-\dot{\alpha}\sin\alpha &= \frac{\dot{x}}{D} - \frac{\dot{D}x}{D^2} \\
&= \frac{\dot{x}}{D} - \frac{\left(-\frac{V}{D}(x\cos\delta + y\sin\delta)\right)x}{D^2} \\
&= \frac{-V\cos\delta + (\omega + \dot{\delta})y}{D} + \frac{V(x\cos\delta + y\sin\delta)x}{D^3} \\
&= \frac{V}{D}\left(\left(-1 + \frac{x^2}{D^2}\right)\cos\delta + \frac{yx}{D^2}\sin\delta\right) + (\omega + \dot{\delta})\frac{y}{D} \\
&= \frac{V}{D}\left((-1 + cos^2\alpha)\cos\delta + \frac{yx}{D^2}\sin\delta\right) + (\omega + \dot{\delta})\frac{y}{D} \\
&= \frac{V}{D}\left((-sin^2\alpha)\cos\delta + \frac{y}{D}\frac{x}{D}\sin\delta\right) + (\omega + \dot{\delta})\sin\alpha\cos\beta \\
&= \frac{V}{D}\left((-sin^2\alpha)\cos\delta + \sin\alpha\cos\beta\cos\alpha\sin\delta\right) + (\omega + \dot{\delta})\sin\alpha\cos\beta \\
&= -\sin\alpha\left(\frac{V}{D}(\sin\alpha\cos\delta - \cos\beta\cos\alpha\sin\delta) - (\omega + \dot{\delta})\cos\beta\right)
\end{aligned} \qquad (2.48)$$

In the end we can get

$$\dot{\alpha} = \frac{V}{D}(\sin\alpha\cos\delta - \cos\beta\cos\alpha\sin\delta) - (\omega + \dot{\delta})\cos\beta \qquad (2.49)$$

Now we do the same for having $\dot{\beta}$:

$$\tan\beta = \frac{z}{y} \qquad (2.50)$$

$$\begin{aligned}\frac{\dot{\beta}}{cos^2\beta} &= -\frac{z\dot{y}}{y^2}+\frac{\dot{z}}{y}\\ &= -\frac{\frac{z}{y}}{y}\dot{y}+\frac{\dot{z}}{y}\\ &= -\frac{\tan\beta}{y}\left(-V\sin\delta-\left(\omega+\dot{\delta}\right)x\right)\\ &= \frac{\tan\beta}{D\sin\alpha\cos\beta}\left(V\sin\delta+\left(\omega+\dot{\delta}\right)D\cos\alpha\right)\end{aligned} \qquad (2.51)$$

Finally we can get:

$$\begin{aligned}\dot{\beta} &= \frac{\frac{\sin\beta}{\cos\beta}cos^2\beta}{D\sin\alpha\cos\beta}\left(V\sin\delta+\left(\omega+\dot{\delta}\right)D\cos\alpha\right)\\ &= \frac{\sin\beta}{D\sin\alpha}\left(V\sin\delta+\left(\omega+\dot{\delta}\right)D\cos\alpha\right)\\ &= \frac{V\sin\beta\sin\delta}{D\sin\alpha}+\left(\omega+\dot{\delta}\right)\frac{\sin\beta}{\tan\alpha}\end{aligned} \qquad (2.52)$$

If we take $\beta=\frac{\pi}{2}$ then we get $D=\frac{V}{\dot{\alpha}}\cos\delta\sin\alpha$ (2.53)

If we eliminate the movements of the rotation and having $(\omega=0,\delta=0,\dot{\delta}=0)$ we get directly the distance of the obstacles in all point of the picture by the relation

$$D=\frac{V}{\dot{\alpha}}\sin\alpha \qquad (2.54)$$

In this case we have $\dot{\beta}=0$ (2.55).

That permits to verify if the movements of rotation have been eliminated well.

II.6.2.Polar projection

In that case the coordinate of the point are the azimuth ψ and the elevation θ as mentioned in the $Fig\ II.5$

To get a picture in polar coordinates, a transformation is therefore necessary to get one

[5]. Here we simplify the calculations using directly ψ and θ instead of $\psi_i = \lambda\psi$ (2.56) , and $\theta_i = \lambda\theta$ (2.57), where λ is the focal length.

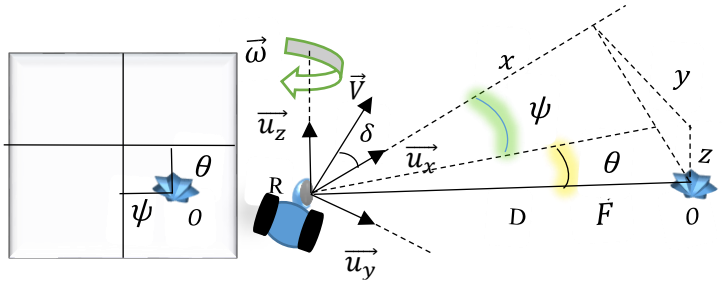


Fig II. 5. Representation of the picture discerned in polar coordinates

From the representation of the optic flux when it is projected in polar coordinates as shown in the Fig II.5 we can get:

$$\begin{matrix} x = D\cos\theta\cos\psi \\ y = D\cos\theta\sin\psi \end{matrix} \Longrightarrow \tan\psi = \frac{y}{x} \qquad (2.58)$$

We have too:

$$\sin\theta = \frac{z}{D} \qquad (2.59)$$

We want to get the representation of the optic flux in polar coordinates by deriving these last two relations

First we start by deriving $\tan\psi = \frac{y}{x}$ (2.60)

$$\begin{aligned} \frac{\dot{\psi}}{cos^2\psi} &= -\frac{y\dot{x}}{x^2} + \frac{\dot{y}}{x} \\ &= -\frac{\tan\psi}{x}\left(-V\cos\delta + \left(\omega + \dot{\delta}\right)y\right) + \frac{1}{x}\left(-V\sin\delta - \left(\omega + \dot{\delta}\right)x\right) \\ &= \frac{V}{x}(\tan\psi\cos\delta - \sin\delta) - \left(\omega + \dot{\delta}\right)(tan^2\psi + 1) \end{aligned} \qquad (2.61)$$

Now we deduce

$$\begin{aligned}\dot{\psi} &= \frac{V cos^2\psi}{x}(\tan\psi\cos\delta - \sin\delta) - cos^2\psi\left(\omega + \dot{\delta}\right)\left(\frac{sin^2\psi + cos^2\psi}{cos^2\psi}\right) \\ &= \frac{V cos^2\psi}{D\cos\theta\cos\psi}\left(\frac{\sin\psi}{\cos\psi}\cos\delta - \sin\delta\right) - \left(\omega + \dot{\delta}\right) \\ &= \frac{V}{D\cos\theta}(\sin\psi\cos\delta - \cos\psi\sin\delta) - \left(\omega + \dot{\delta}\right) \\ &= \frac{V\sin(\psi - \delta)}{D\cos\theta} - \left(\omega + \dot{\delta}\right)\end{aligned} \quad (2.62)$$

Now we do the same by deriving the following equation: $\sin\theta = \frac{z}{D}$ (2.63)

$$\begin{aligned}\dot{\theta}\cos\theta &= \frac{\dot{z}}{D} - \frac{z\dot{D}}{D^2} \\ &= \frac{zV}{D^3}(x\cos\delta + y\sin\delta) \\ &= \frac{V\sin\theta}{D^2}(D\cos\theta\cos\psi\cos\delta + D\cos\theta\sin\psi\sin\delta)\end{aligned} \quad (2.64)$$

Finally we can write:

$$\begin{aligned}\dot{\theta} &= \frac{V\sin\theta}{D}(\cos\psi\cos\delta + \sin\psi\sin\delta) \\ &= \frac{V\sin\theta\cos(\psi - \delta)}{D}\end{aligned} \quad (2.65)$$

In the end we can resume that the optic flux is represented by the polar coordinates as:

$$\begin{aligned}\dot{\psi} &= \frac{V\sin(\psi - \delta)}{D\cos\theta} - \left(\omega + \dot{\delta}\right) \\ \dot{\theta} &= \frac{V\sin\theta\cos(\psi - \delta)}{D}\end{aligned} \quad (2.66)$$

We deduce from the last equation that $\dot{\theta}$ is not depending on the rotation movement. But we can calculate the distances of the obstacles even if the robot is turning. Besides, a rotation of the robot according of the vertical axis entails a horizontal transfer of the picture in polar coordinates.

II.6.3. Plane projection

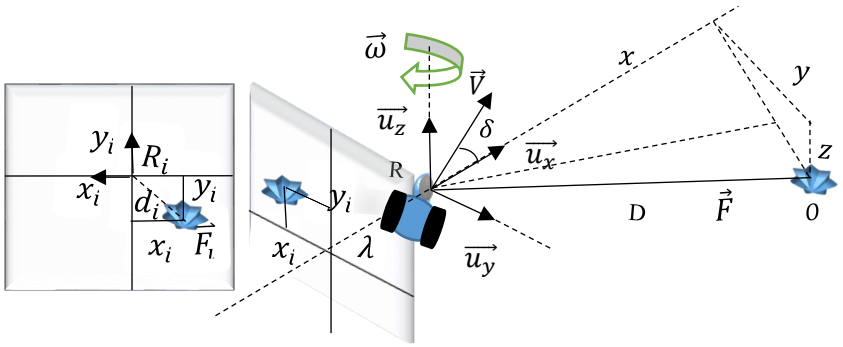


Fig II. 6. Representation of the optic flux projected on the plan

In the Fig II.6 we have x_i, y_i are the coordinates of the projection of the point O in the picture.

$d_i = \sqrt{{x_i}^2 + {y_i}^2}$ (2.67) is the distance of this point in the center of the picture. λ is the focal length of the objective used, here we use the obstacle, see also [5]. As the figure shows it we take the origin of the reference mark is in the center of the picture. Also we take the axis of the y_i oriented upwards and the axis of x_i is oriented to the right. From the figure we can get the two following relations

$$x_i = -\lambda \frac{y}{x}$$
$$y_i = \lambda \frac{z}{x} \qquad (2.68)$$

Now in order to have the representation of the optic flux in the image, we have to derive these last two expressions. For that we proceed by starting with the first expression:

$$\begin{aligned}
\dot{x}_i &= -\lambda\frac{\dot{y}x}{x^2} - \lambda\left(-\frac{\dot{x}y}{x^2}\right) \\
&= -\frac{\left(-\lambda\frac{y}{x}\right)}{x}\dot{x} - \frac{\lambda}{x}\dot{y} \\
&= -\frac{x_i}{x}\dot{x} - \frac{\lambda}{x}\dot{y} \\
&= -\frac{x_i}{x}\left(-V\cos\delta + \left(\omega + \dot{\delta}\right)y\right) - \frac{\lambda}{x}\left(-Vsin\delta - \left(\omega + \dot{\delta}\right)x\right) \\
&= \frac{V}{x}(x_i\cos\delta + \lambda\sin\delta) + \left(\omega + \dot{\delta}\right)\left(\lambda - \frac{x_i y}{x}\right) \qquad (2.69) \\
&= \frac{V}{x}(x_i\cos\delta + \lambda\sin\delta) + \left(\omega + \dot{\delta}\right)\left(\lambda - \frac{x_i\left(-\frac{xx_i}{\lambda}\right)}{x}\right) \\
&= \frac{V}{x}(x_i\cos\delta + \lambda\sin\delta) + \left(\omega + \dot{\delta}\right)\left(\lambda + \frac{{x_i}^2}{\lambda}\right) \\
&= \lambda\frac{V}{x}\left(\frac{x_i\cos\delta}{\lambda} + \sin\delta\right) + \left(\omega + \dot{\delta}\right)\left(1 + \frac{{x_i}^2}{\lambda^2}\right)
\end{aligned}$$

Now we derive the expression $y_i = \lambda\frac{z}{x}$ (2.70) as we can see in the following

$$\begin{aligned}
\dot{y}_i &= \lambda\frac{\dot{z}x}{x^2} - \lambda\frac{\dot{x}z}{x^2} \\
&= \lambda\frac{\dot{z}x}{x^2} - \frac{\dot{x}\left(\frac{\lambda z}{x}\right)}{x} \\
&= \lambda\frac{\dot{z}x}{x^2} - \frac{\dot{x}y_i}{x} \\
&= \lambda\frac{(0)x}{x^2} - \frac{\left(-V\cos\delta + \left(\omega + \dot{\delta}\right)y\right)y_i}{x} \qquad (2.71) \\
&= -\frac{\left(-V\cos\delta + \left(\omega + \dot{\delta}\right)\left(-\frac{x_i x}{\lambda}\right)\right)y_i}{x} \\
&= \frac{Vy_i\cos\delta}{x} + \left(\omega + \dot{\delta}\right)\frac{x_i y_i}{\lambda}
\end{aligned}$$

6.4. Diagram showing Obstacle avoidance using algorithms of vision

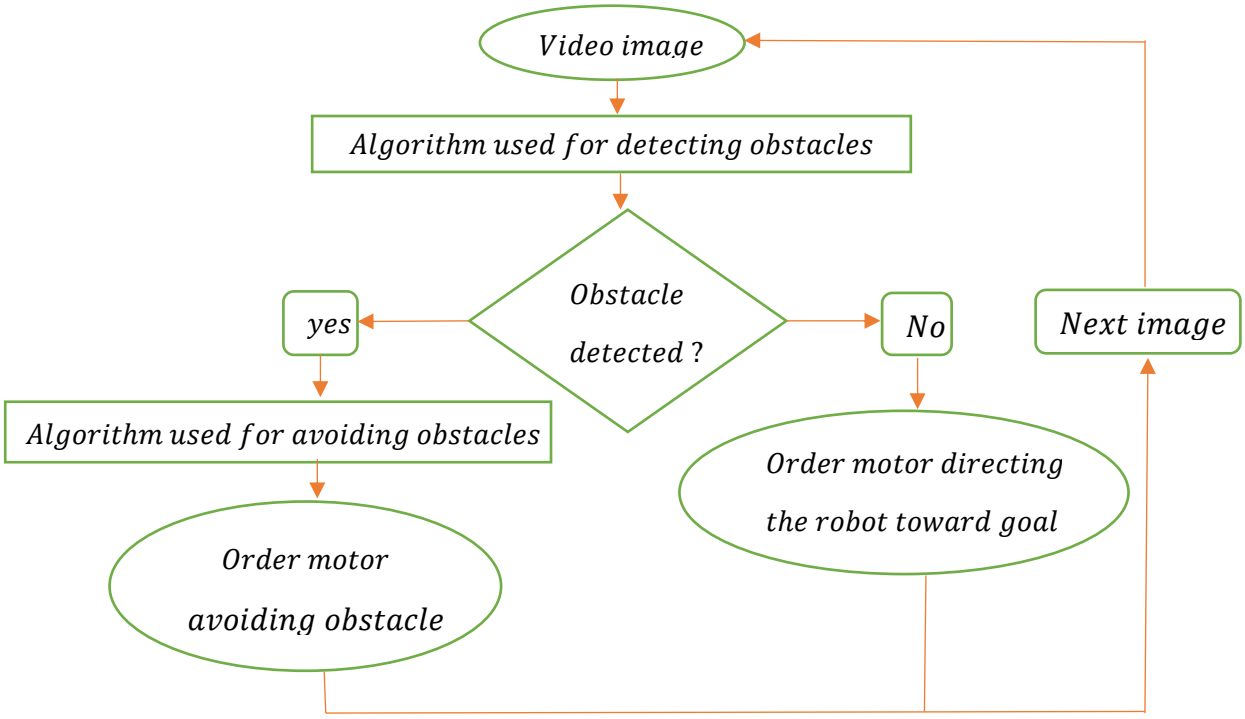


Fig II.7. Illustration of the algoritmic structure used to facilitate the compromise between loop line of obstacles and displacement toward the goal

The algorithms are conceived by the genetic programming system, we can find more in [5]. The first algorithm used for detecting obstacles has been simplified to use only one image. The second algorithm used for avoiding obstacles has been made to be able to reuse the image filtered exit out of the first algorithm.

Most known genetic algorithms of evolutionary algorithms are probably the genetic algorithms developed by John Holland in the beginning of the seventies.

As there name indicates it, these algorithms are inspired directly by genetics. The individuals are represented by one binary sequence called genome.

CHAPTER III Sensors used for Mobile Robotics

III.1 Introduction

The robots need to know what the world around them looks like. Thus we have a different kind of sensors [36] which are known as devices that can sense and measure physical properties of the environment. We use contact sensors like bumpers. Internal sensor like accelerometers (spring-mounted masses), gyroscopes (spinning masse, laser light), or compasses, and inclinometers (earth magnetic field, gravity). Also we can use proximity sensors like the sonar (time of flight), a radar (time and frequency), or laser range-finders (triangulation, TOF, and phase), or also Infra-red sensor (intensity). We have visual sensors like cameras. And finally we can require to the satellite-based sensors like GPS.

III.2. Sensor types

First, we can start to classify the sensors using two important functional axis: $proprioceptive/exteroceptive$ and $passive/active$ sensors [32].

III.2.1. Proprioceptive sensors

The robot measure a signal originating from using proprioceptive sensors. These sensors are responsible for monitoring self-maintenance and controlling internal status of the system. As a common uses of a proprioceptive measurements we can talk about the battery monitoring, current sensing, heat monitoring, wheel load, motor speed, and robot arm joint angles. We can take some examples as the following

* Global positioning System (GPS)

* Shaft Encoder, also known as a rotary encoder, is an electro-magnetic device that works as a transducer to covert the angular position of a shaft or axle to an analog or digital code.

* Compass: A compass sensor is used to detect direction and accurately correct motion.

* Inclinometer: An inclinometer sensor measures the tilt or angle of an axis.

III.2.2. Exteroceptive Sensors

Exteroceptive Sensors uses information taken from the robot's environment. For example: we take the distance measurements, light intensity, amplitude. For the sensors we take as examples: cameras (single, stereo…), laser, scanner, sonar, tactile…etc.

III.2.3. Active sensors

Active sensors emit their proper energy into the environment and measure the reaction. More robust, less efficient.

III.2.4. Passive sensor

Measure the energy coming from the environment. Less intrusive, but depends on environment Like receiving the light for camera.

Table III.1 provides a classification of the most useful sensors for mobile robots application [42] [35]. We start by giving the meaning of the abbreviations we have written in the table and that concerns the letters $A, P, and\ \ A/P$

$A: active$; $P: passive$; $P|A\ passive|active$; $PC: proprioceptive$; $EC: exteroceptive$.

Table III.1

Classification of sensors used in robotic application

General classification (*typical use*)	*Sensor* *Sensor system*	*PC or EC*	*A or P*
Tactile sensors (*detection of physical contact or closeness; security switches*)	Contact switches, bumpers, optical barriers	*EC*	*P*
		EC	*A*
	Noncontact proximity sensors	*EC*	*A*

General classification (typical use)	Sensor Sensor system	PC or EC	A or P
Wheel/motor sensors (wheel/motor speed and position)	Brush encoders	PC	P
	Potentiometers	PC	P
	Synchros, resolvers	PC	A
	Optical encoders	PC	A
	Magnetic encoders	PC	A
	Inductive encoders	PC	A
	Capacitive encoders	PC	A
Heading sensors (orientation of the robot in relation to fixed reference frame)	Compass	EC	P
	Gyroscopes	PC	P
	Inclinometers	EC	A|P
Ground − based beacons (localization in a fixed reference frame)	GPS	EC	A
	Active optical or RF beacons	EC	A
	Active ultrasonic beacons	EC	A
	Reflective beacons	EC	A
Active ranging (reflectivity, time of flight, and geometric triangulation)	Reflectivity sensors	EC	A
	Ultrasonic sensor	EC	A
	Laser rangefinder (1D)	EC	A
	Structural light (2D)	EC	A
Motion/speed sensors(speed relative to fixed or moving Objects)	Doppler radar	EC	A
	Doppler sound	EC	A
Vision-based sensors (visual ranging, hole image- analysis, segmentation, object recognition)	CCD/CMOS cameras Visual ranging packages Object tracking packages	EC	P

III. 3. *GPS sensor*

GPS is an acronym for Global Positioning System many people use the term GPS when referring to the receivers that's found in their car. But it's not entirely true. GPS is accurately

-is more defined as a navigational system that relies on satellites signals to show an objects location. The GPS network is made up by twenty four operating satellites. Four satellites always have a line of visibility to our receiver at any time of the day. Our receiver gets the information on the location at least from three satellites around us and the distance between us and those satellites.

We can use those measurements to pinpoint where we are, that's called trilateration. All the receivers and GPS satellites don't just have to know where they are but they have also to know what time it is when they sending their positions. Every satellite got an atomic clock on it accurate to a billion of a second. And every receivers use a quartz clock that constantly resets itself to match its time to the last one got from the satellite. And that's why also why GPS is not just using positioning anymore. It's also use time keeping situations where everything need to be synchronized perfectly. As we know stock markets need to use that every day. In other way we can say that the GPS satellites continually emit signals that state two pieces of information.one is the current time and the second is where exactly about the earth the satellite is at that time. GPS receivers use this information to do a special calculation called triangulation. The receiver compare the time that was send from the satellite with the time was the signal was received. The deference in time indicate the distance between the receiver and the satellite. A GPS receiver need to get signals from three satellites to calculate latitude and longitude. If we get a signal from four satellite we can also calculate altitude. We can illustrate the position of our robot as using three GPS satellites as shown in the *Fig III*. 1

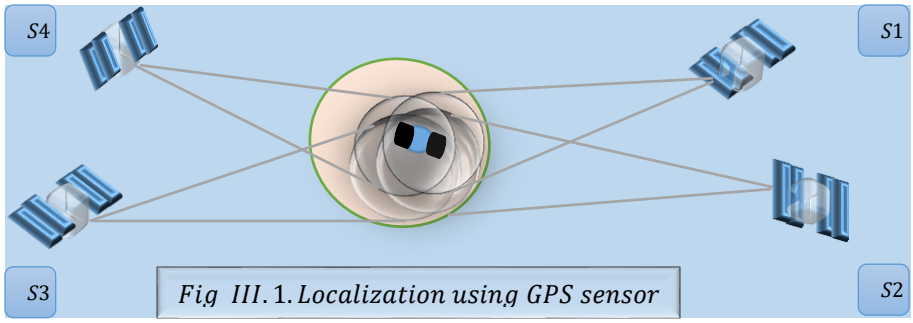


Fig III. 1. Localization using GPS sensor

If we suppose that our robot posed in a circular lot. We have got inside the robot a receiver with which it receives signals from four satellites in order to indicate for us its position and its time. Our lot is now the region inside the satellite orbits (including the Earth).

We take the center of the Earth as the origin in our coordinate system. We are working in three dimension that means we need information from four satellites. We name these S_1, S_2, S_3, and S_4; and we suppose that S_i is located at (X_i, Y_i, Z_i) when it transmits a signal at time T_i. If the signals are received at times T'_i, according to the to clock in our receiver, we let $\Delta t_i = T'_i - T_i$, and lets ε represent any error in our clock's time. The receiver allows for the mean of passage through the Earth's atmosphere and computes distances $d(\Delta t_i, \varepsilon)$ that indicate for us how far the robot is far from the satellites. Our position (x_0, y_0, z_0) is located on each of four huge spheres. In most situations. There will be one sensible value of ε that allows the spheres to have a point in common. The robots location is determined by solving a system equations as it is introduced in [17] [30] [31].

$$\begin{Bmatrix} (x_0 - X_1)^2 + (y_0 - Y_1)^2 + (z_0 - Z_1)^2 = d(\Delta t_1, \varepsilon)^2 \\ (x_0 - X_2)^2 + (y_0 - Y_2)^2 + (z_0 - Z_2)^2 = d(\Delta t_2, \varepsilon)^2 \\ (x_0 - X_3)^2 + (y_0 - Y_3)^2 + (z_0 - Z_3)^2 = d(\Delta t_3, \varepsilon)^2 \\ (x_0 - X_4)^2 + (y_0 - Y_4)^2 + (z_0 - Z_4)^2 = d(\Delta t_4, \varepsilon)^2 \end{Bmatrix}$$

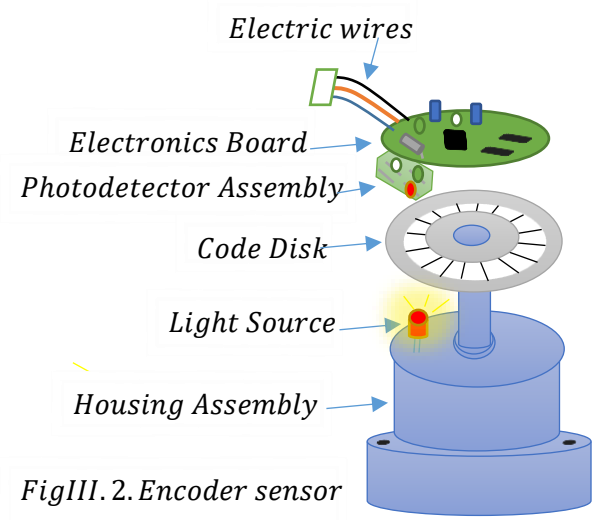
When we find a numerical solution, the rectangular coordinates (x_0, y_0, z_0) are converted into spherical coordinates of latitude, longitude, and altitude above sea level.

III.4.Encoders

An encoder is a sensing device that provides digital feedback signals in response to movement of a mechanical position of a tool or workpiece and convert it into an electrical signal that can be read or measured by some type of control device in a motion control system such as a counter. The encoder sends a feedback signal that can be used to determine

-position, count, speed, or direction. A control device can use this information to send a command for a particular function. Encoders may produce either incremental or absolute signals.

Now we show an example of a rotary encoder used in robotics.



FigIII. 2. Encoder sensor

III.4.1.Difference between absolute and incremental encoders

The major difference between an incremental and an absolute encoders is that the absolute encoder keep track of its position at all time, and provides it as soon as power is applied. This characteristic is particularly useful in applications where the equipment runs infrequently and has power turned off between uses. For the incremental encoders, if we make a comparison, do not provide any information about the position at start-up. But simply keeps track of how far it has moved. The only way to determine the absolute position of incremental encoder is to set the equipment to a known reference position and then zeros the counters. Also we can say that applied incremental signals do not indicate specific position, only that the position has changed.

Absolute encoders, on the other hand, use different word for each position, meaning that an

-absolute encoder provides both the indication that either the position has changed and of the absolute position of the encoder. We note that encoders measures one degree of freedom, just the angle.

III.5. Infra – Red sensor

Infra-Red (IR) Sensors are probably the simplest type of non-contact sensor (widely used in mobile robotics for obstacle avoidance). They work by using a specific light sensor to detect a select light wavelength in the Infra-Red spectrum. By using a LED which produces light at the same wavelength as what the sensor is looking for, we can look at the intensity of the received light. When an object is close to the sensor, the light from the LED bounces off the object and into the light sensor. This results in large jump in the intensity which we already know can be detected using a threshold. We note that IR are short range (typical maximum range is 50 to 100cm).

We can see in the *Fig III.3* how the IR sensors measure the brightness

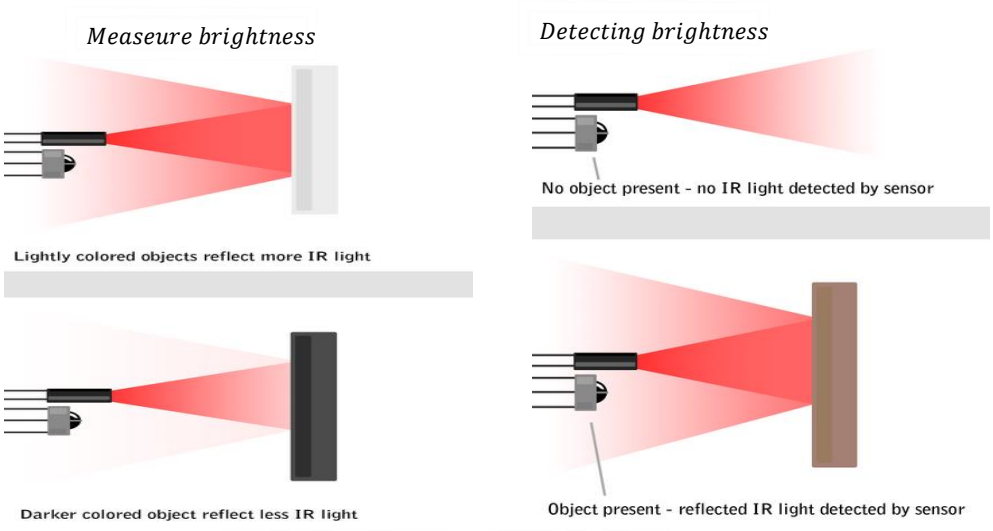


Fig III.3. Infra – Red sensor

III. 5. 1. *Detecting brightness*

Since the sensor works by looking for reflected light, it is possible to have a sensor that can return the value of the reflected light. This type of sensor can be used to measure bright the object is. This is useful for tasks line tracking. To differentiate emitted IR from ambient IR (e.g. light, sun, etc.), the signal is modulated with a low frequency (100 Hertz).

In certain environments, IR sensors can be used to measure the distance to the object. That require uniform colors and structures.

III.6.Active ranging

Active ranging sensors are commonly used in mobile robotics. Most mobile robots rely heavily on active ranging sensors when it is about to avoid an obstacles. In the following we can see to time-of-flight active ranging sensors: the ultrasonic sensor and the laser range finder.

III.6.1.Time –of –flight active ranging

Time-of-flight ranging makes the use of the propagation speed of the wave sound or an electromagnetic wave. And the distance crossed by those waves is given by the following relation:

$d = v_s . t$ (3.1)

$d =$ the distance the waves may travel; $v_s =$ the speed of sounds; $t =$ time of flight

III. 6. 2. *Ultrasonic Sensor*

An Ultrasonic sensor is a device that can measure the distance to an object by using sound waves. It measures distance by sending out a (ultrasonic) pressure wave at a specific frequency and listening for that sound wave to bounce back. By recording the –

-elapsed time between the sound wave being generated and the sound wave bouncing back, it is possible to calculate the distance between the sonar sensor and the object. An ultrasonic sensor has been introduced and explained in [48].

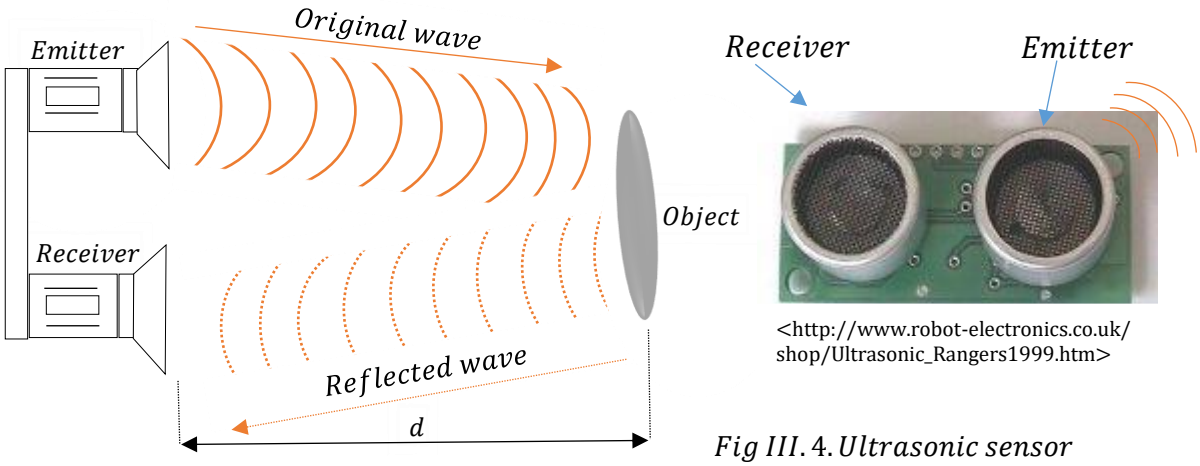


Fig III. 4. Ultrasonic sensor

These sensors are very similar to the SONAR (*Sound Navigation And Ranging*) used by the submarines and their principle of working is the same. The sensor is equipped of an emitter of ultrasonic sounds (non-audible mechanical waves for humans, that means more than 20000 *Hertzes*) and of a receptor contrary to the infrared sensors, we don't measure the received intensity, but the time that goes by between the moment of broadcast and the instant where the reflexive waves come back to the receptor. From there we can calculate the distance of the object if we know the speed of the sound in the environment we are situated. The sound travels through air at about 343 meter per second. And that in a standard pressure and the temperature of $20C^0$ .The sound wave will travel two time from the sensor until back to it again. We have to divide all the distance by the number two. And that will give us the distance between the sensor and the object the sound wave will hit before reflection. The temperature and the humidity of the air may affect the accuracy of the Ultrasonic sensor.

Some objects might not be detected by Ultrasonic sensors. The reason for that looks like some objects are shaped or positioned in such a way that the sound wave bounces off the object, but are reflected away from the Ultrasonic sensor. Some objects can also absorb the sound wave, other small objects are too small to be able to reflect enough sound wave back to the sensor to be reflected. Which means that there is no way for the sensor to detect them accurately. These are important factors to consider when designing and programming a robot using an ultrasonic sensor. To calculate the distance as the following equation shows:

$$d = \frac{v_{sound} * \Delta t}{2} \qquad (3.2)$$

We have here the speed of sound as it is introduced in [26]:

$$v_{soud} = \sqrt{\gamma RT} \qquad (3.3)$$

$\gamma = ratio\ of specific\ heat$

$R = gas\ constante$

$T = temprature\ in\ degree\ s\ kelven$

The time taken is described with Δt

The distance of the object is d

III.6.3. Laser range finder

Laser range finder is a time-of-flight sensor [24]. It is commonly used to measure the distance [44], velocity and acceleration of objects. It consists of a transmitter which illuminates a target with a collimated beam (e.g., laser) and a receiver capable of detecting the component of light which is essentially coaxial with transmitted beam [20]. It is often known as laser radar and also known as a *lidar* as acronym of light detection and ranging. It uses laser to measure the elevation of things like the ground and forest and even buildings.

It's like a sonar which uses sound waves to map things or radar which uses radio waves to map things. But a *lidar* system use light send out from a Laser which is too used to surf things like scanning bar codes, removing hair…etc. there is three different ways to collect data: from the ground, from a plane, and from the space. Airborne *lidar* data are the most commonly available data which are freely available for the National Ecological Observatory Network in the United States of America. To get how Laser used to calculate height in airborne *lidar* we need to understand the four part of the system. First the airplane contain the *lidar* unit itself which uses the Laser to scan the earth from side to side as the plane fly. The Laser system uses either green (532nm) or infrared light (1064nm) for mapping the vegetation. The next component of the *lidar* system is the GPS that tracks the altitude also x and y location of the airplane. The GPS help us to find where the latterly factions are on the ground. The third component of the *lidar* system is what is called an Inertial Measurements Unit (or IMU). We define the IMU as a device that uses measurement systems such as gyroscopes and accelerometers to estimate the relative position (x, y, z) [24]. The IMU tracks tells that the plane in the sky fly, which is Important for The accurate elevation calculations. Finely the *lidar* systems include a computer which records all of that important high information that the *lidar* collect as it scans the earth surface.

In a laser system we use a word pulse which is simply referred to the burst of light energy that emitted by the *lidar* system. And second we find the word return which is referred to reflected energy that has been recorded by the *lidar* sensor. So pulse is a light energy travel to the ground and return back to the *lidar* sensor. To get height the *lidar* system record the time that it takes for the light energy to travel to the ground and back. The system then uses the speed of light to calculate distance between the top of an object and the plane. The distance is equal to the result of the multiplication of the speed of light by the travel time divided by two, because we have the light travel to the ground and back. This calculation gives us how far the light actually travel to the ground.

To figure out the actual elevation of the ground we take the plane's altitude calculated using

-the GPS receiver and then subtract the distance that the light travel to the ground. And that coverage the basics of how the *lidar* system uses Lasers to measure height.

We have presented in *Fig III*. 5. (*b*) the sick LMS 200 laser scanner [44]. And its angular resolution is 0.25 deg. Also we have its depth resolution ranges between 10 mm and the typical accuracy is 35 mm, over a range from 5 cm up to 20 m or more (up to 80 m), depending of the reflectivity on the object being ranged [24].

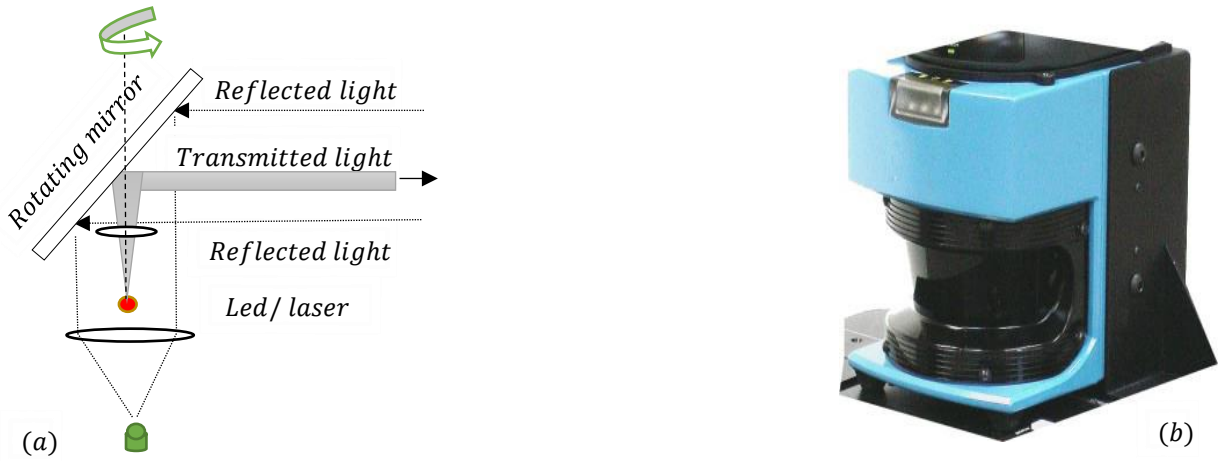


Figure III. 5. (*a*) *Schematic drawing of laser range sensor with rotating mirror*; (*b*)*Industrial* 180 *degree laser range sensor* [40].

III.7.Radar

Radar usually uses electromagnetic energy in the 1-12.5 GHz frequency range. This corresponds to wavelengths of 30cm-2cm. It may also use microwaves energy which are unaffected by fog, rain, dust, haze, and smoke. It may use a pulse time-of-flight methodology of sonar and Lidar.

III.8.Compass sensors

Compass sensors is a navigational device capable of showing directions of reference which is relative to the earth's surface, it can valuable sensor used in in robots for navigating

-its surroundings. The compass can measure the horizontal component of the earth's magnetic field. The most common type of compass is a simple magnetic compass that uses the magnetic pull of the earth's North Pole to determine which direction is north. Usually a magnetic compass and a digital compass are used in the field of robotics.

A magnetic compass is aligned with respect to the earth's magnetic field. This compass consists of magnetized pointer that is usually marked at the north end, and a magnetized bar or needle turning freely upon a pivot. The magnetic field exert a torque on the needle which moves the needle so that one end of this component is pulled toward the earth's north magnetic pole, and the other toward the south magnetic pole.

The digital compass provides the measurements based on the earth's magnetic field for the navigation of the robots. This digitalized version of a standard compass consists of a tiny MEMS Nano-structures that tend to bend due to the electromagnetic field.

The major applications of a navigation compass in the robotic systems include motion control, and detection of the heading of mobile robots.

III.9. Gyroscopes sensor

In our days two type of gyroscopes are used in robotics

III.9.1. Mechanical gyroscope

A mechanical gyroscope are a heading sensors that preserve their orientation to a fixed reference frame [33]. They provide an absolute measure for the heading of a mobile system. Gyroscopes are classified in two categories. Mechanical gyroscopes and optical gyroscopes.

Mechanical gyroscopes known as a fast spinning wheel around its vertical axis, and having the most of its mass concentrated in the outer periphery (e.g. bicycle wheel). Due to the law of conservation of momentum the spinning wheel will stay stable in its original orientation, and to rotate the gyroscope a force will be required. Then we can use the

-gyroscope to maintain orientation or to measure the rate and direction of rotation. A harsh reaction can be felt in its horizontal axis due the angular momentum which is associated with a spinning wheel. We take T as a reactive (applied input) torque and thus the tracking stability with inertial frame are proportional to the spinning speed ω, the rate (speed) of precession Ω, and the rotational inertia of rotor I [26].

$$T = I\omega\Omega \qquad (3.4)$$

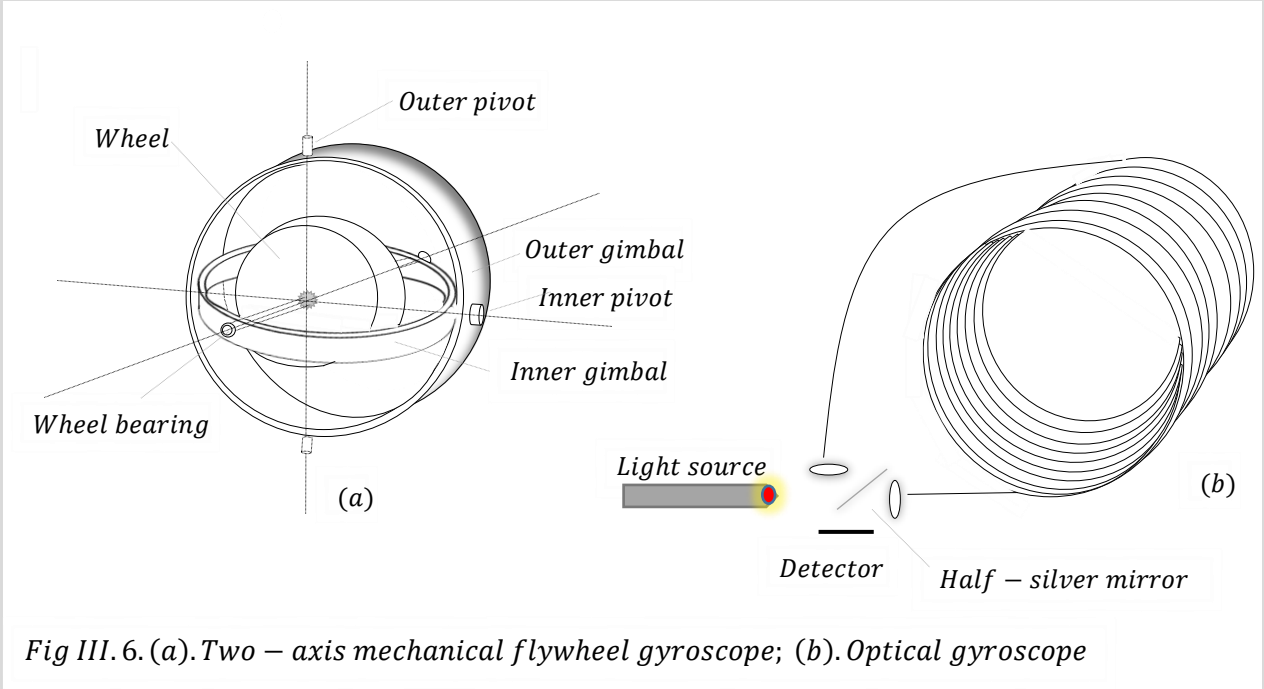


Fig III. 6. (a). Two − axis mechanical flywheel gyroscope; (b). Optical gyroscope

III.9.2.Optical gyroscope

Optical gyroscopes are angular speed sensors that use two monochromatic light (or laser) beams from the same source, instead of moving, mechanics parts [26] [33]. Knowing that the speed of light is constant in a vacuum and it was taken as $c = 299.792.458\, m/s$. Those sensors are based on the *Sagnac* effect, also called *Sagnac interference* [34]. That phenomenon manifest itself in a setup called a ring interferometer. A beam of light is split

-and the two beams are made to follow the same path [1] but in opposite directions (one is travelling in a fiber clockwise, the other counterclockwise around a cylinder).The laser traveling in the rotation has a slightly shorter path. On return to the point of entry the two light beams are allowed to exit and undergo interference. The relative phases of the two existing beams, and thus the position of the interference fringe, are shifted [34][35] according to the angular velocity of the apparatus (cylinder). In other word the phase shift of the two beams is proportional to the angular velocity of the cylinder which its coil consists of as much as 5 km optical fiber.

III.10.Accelerometer sensor

Accelerometers are an electro-mechanical devices used to measure the acceleration not by calculating how speed changes over time but by measuring forces acting upon them, in other word by sensing how much a mass presses on something when a force acts on it, whether they are static like gravity pulling an object lying at the ground or dynamic caused by motion or vibration. They are used to measure acceleration along one or more axis and are relatively insensitive to orthogonal directions. In robotics they are applied for self-balancing.

III.10.1.Types of accelerometer

They are many different types of accelerometer, we can take for example either a mechanical or solid state device.

III.10.1.1.Mechanical accelerometer

Mechanical accelerometer are a bit like scaled-down versions of passengers sitting in cars shifting back and forth as forces act on them. They have something like a mass (m) (often called seismic-mass or proof-mass) attached to a spring with stiffness (k) suspended inside

[1]: We use a (FOG) known as a Fiber Optical Gyroscope system which is a cylinder coil made for such an experience.

-an outer casing. In most cases the system includes a dashpot [2] with a damping coefficient (c) to provide a desirable damping effect, and it is also normally attached to the mass in parallel with the spring. When the spring mass system is subjected to linear acceleration, the proof-mass will receive a direct force, causing it to deflect, which is sensed by a suitable means (generally a piezo-electric sensor used) and converted into an equivalent electrical signal. The damping will help the system to find its stability as soon as possible under applied acceleration. When they accelerate, the casing moves off immediately but the mass lags behind and the spring stretches with a force that corresponds to the acceleration. The distance the spring stretches (which is proportional to the stretching force) can be used to measure the force and the acceleration in a variety of different way. We use Newton's second law, where all real forces acting on the proof-mass are equal to the inertia force on the proof-mass. The equation of motion of the mechanical system is a second order linear differential equation with constant coefficients

$$m\ddot{x} + c\dot{x} + kx = F \qquad (3.5)$$

By using Laplace transform (3.5)

$$(ms^2 + cs + k)x = F \qquad (3.6)$$

where:

$m\ mass\ of\ the\ proof - mass, x\ is\ a\ relative\ movement\ of\ the\ p$

$- mass\ with\ respect\ to\ frame, c\ is\ a\ damping\ coefficient, k\ is\ a\ spring\ constant,$

$F\ is\ a\ force\ applied$

The general solution $X(t)$ is the sum of the complementary function $X_c(t)$ and the particular integral $X_p(t)$.

$$X(t) = X_c(t) + X_p(t) \qquad (3.7)$$

[2] A dashpot is a mechanical device, a damper which resist motion via viscous friction. The result force is proportional to the velocity, but acts in the opposite direction, slowing the motion and absorbing energy. It is commonly used with a spring

The complementary function satisfies the homogeneous equation.

$$m\ddot{x} + c\dot{x} + kx = 0 \qquad (3.8)$$

With Laplace transform we get

$$(ms^2 + cs + k)x = 0 \qquad (3.9)$$

The solution for $X_c(t)$ is

$$X_c(t) = Ce^{st} \qquad (3.10)$$

Substituting (3.9) in (3.8)

$$(ms^2 + cs + k)Ce^{st} = 0 \qquad (3.11)$$

(3.11) is taken as the characteristic equation of the system, and its solution may take S as a values.

$$S_{1,2} = \frac{1}{2}m(-C \pm \sqrt{C^2 - 4mk}) \qquad (3.12)$$

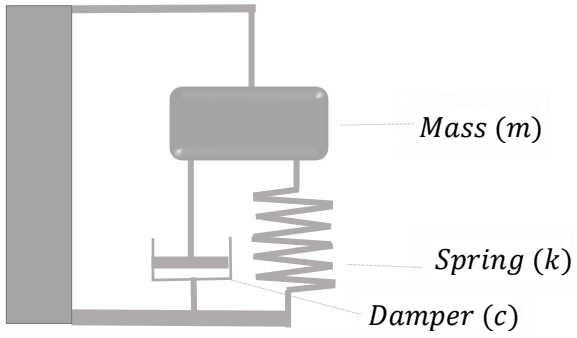


Fig III. 7. Mechanical accelerometer

III.10.1.2. Solid State Accelerometer

Solid state accelerometer can be divided to various subgroups, including surface vocal wave, vibratory, silicon and quartz devices. They are small, trustworthy and rough. An example of a solid-state accelerometer is the surface acoustic wave (SAW) accelerometer [38].

CHAPTER.IV. Introduction to control theory

IV.1.The basic building blocks

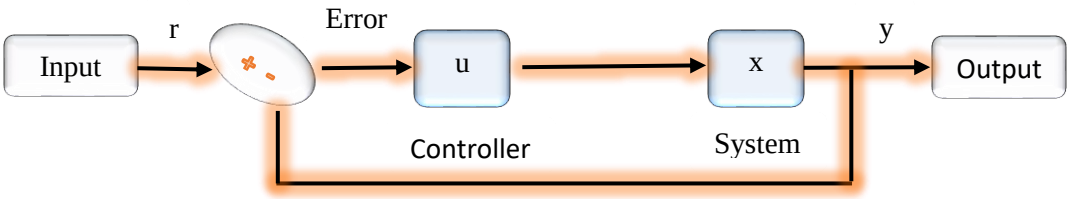
The question here is how to make a mobile robot move in effective, safe, predictable, and collaborative ways using modern control theory. In order to be precise and being able to have better understanding for what the robot may will be doing in place, and ultimately control deals with dynamical systems what is describe something that is changing over time [1], it could be a car that is moving or a market which is going to be evolving like a particular price of an object, or a climate with its changing. But ultimately controls are dealing with how we can influence that change of the system. Some systems are harder to control than others and depend on the environment the system has to deal with. As an example of a systems we can find Robots, circuits, engines, stock markets, epidemics…etc.

If we consider a robot like a system, and when we want to control that robot like driving it from a place to another we need to know where the system is and where it will be supposed to be in the next situation and how the system may will behave, that means we need to describe the state of the system like its velocity or its positions [1] [2], or it's orientation. In other examples like in the stock markets we need to know the price of a specific products in order to make a decision of buying or selling the product we desire.

In any way the state is the key thing to describe what the system is up to. What is the system actually doing is dynamics, in other way the description of the change as a function all the time.

We need now to tell the system what it has going to do that means we have to make some influencing toward it, for that we need referencing. For the robot as example we can set the cruise controller to seventy centimeter per second. In other example like a room we can set the temperature to twenty five degree Celsius, that is not going to work until we make sure that how fast the system will going to work, that means we need an output for our system. The outputs are the things which are able to get out of the systems what is telling us in the end what the system is doing, we need some way of mapping signals in actual control signals the inputs.

So we have to be the things or the means which is able to text the reference and produce us the reference signal and then hits the states of the system. Until what we have for now the control signal have nothing to do with the measurements, for that we need the feedback which is the mapping from outputs to inputs. Subtracting the input from the output will give us the error which is in term our system's performance. That error is transmitted is translating into some control signal that's then hitting the system. So that feedback mapping is really the key to do any kind of controls in an effective way. We will see the basic building blocks. Finally we resume all of that in a simple mapping blocks, and we have to make it with a feedback which is illustrated in the $Fig\ IV.1$



$Fig\ IV.1$. The basic building blocks

As a conclusion we can say that controls are generally dealing with dynamical systems as we have seen in the case of mobile robots.

IV.2.Making the models

In order to have a clear understanding or to get a better explanations for the behavior or the need to control of any system we have to seek for a best and precise tools, and the one we are going to use here is mathematics. We have to make a model to our system and that is an approximation in mathematical way for which state the system remain and for what really that last one is going to do. So models are actually basic key when it comes to design a controllers. The real question when we want to design a model is how to pick the input signal after the controller has to take the reference from the input and making a comparison with the output(or the measurement) signal.

And the goal here is to do some correction of what the system is doing. There is many

-objectives by taking that control input signal, generally we write the symbol u for the control input signal.

IV.3.The objectives

The first one is basically always stability for objective control that means the system doesn't blow-up. If the robot hit an obstacle that means the robot may lose its stability. After getting a stability for our system we will look for more like following a path in the case of the mobile robot, or executing a reference the system has been received, that second objective is about tracking a reference signal. For the third type we have robustness which is also taken as an important objectives, we know that the models we design will never going to be perfect. We can't let our controller too highly dependent in what the particular parameters in the model are. When we design our controller we have to take in account the variation of those parameters. It is like giving immune system to the controller what has the ability to deal with every change in the parameters, because we know too that the most of the existing systems are nonlinear and using a best and strong algorithm which are able to process a huge amount of data, and also the ability to make a rapid calculation and giving us in the end a better results.

Adding to the robustness we have too disturbance rejection. We are acting in measurements and sensor have measurements noise.

The robot may be disturbed by the environment like the air or a bad visibility especially in bad weather or a lack of sight for the robots moving in a deep water. For that we need the controller to be effective. We can also add another objective known as optimality, the robot has to move in its environment in the best possible way like choosing to move in a short path at a minimum time. The robot has to choose the optimal solution to insure its efficiency for getting a best behavior. To do all of that we need that the model to be effective, such effective strategies rely on predictive models [1].

IV.4.Dynamical models

For the predictive models we have to use a dynamical models, and what we mean with

-dynamical is change over time [1]. We can use a discrete time that is what the robot use to process data in distinct times, so for that we always refer to the word next. Or continuous that means we have to know how the robot will behave in the interval continuous time we have to precise. For the continuous time there is no next because the division of time is infinite and continuous but we have to focus to how the robot may behave in time, or how the change will be with respect to time.

In the mathematical model we use for discrete time the difference equation. As an example we can write:

$$x_k = f(x_k, u_k) \quad (4.0) \longleftarrow \text{Difference equation}$$

We can explain that for the mobile robot which is moving in a field as for its state position in time k and its behavior with U in the same time and its state position in time $k+1$ later. And the function f tell us how the robot go from one state to another.

As we know that all the laws of physics are in continuous time that means if the robot has to move from one state to another we have to require here to the derivatives with respect to time. Then for the mathematical model we will use the differential equation as shown in the next example:

$$\frac{dx}{dt} = f(x, u) \quad \sim \quad \dot{x} = f(x, u) \quad (4.1) \longleftarrow \text{Differential equation}$$

The equation here is about the instantaneous change not the next state, because we are in continuous time. But in implantation everything is discrete and time is sampled.

The sample time is δt . And now we are going from continuous to the discrete time as shown in the next written equations:

$$x_k = x_k \delta t \Longrightarrow x_{k+1} = x_{k+1} \delta t \quad (4.2)$$

$$x(k\delta t + \delta t) \approx x_k \delta t + \delta t \dot{x}_k \delta t \quad (4.3)$$

$$x_{k+1} = x_k + \delta t f(x_k, u_k) \qquad (4.4)$$

IV.5.Dynamical equations

Now as an example for an application of what we have just seen above, we try to apply all of that to build a cruise controller for a car and its job is to make a car drive at a desired reference speed. We always know that the law of physics ultimately dictate us how the objects like the robots may behave. For the car, with a mathematical model we can write the equations for its behavior.

We begin by writing the second newton's law

$$F = m\gamma \qquad (4.5)$$

F : is a newton force (unity measurement is newton or simply we can write N)

m : is a mass of the system (kilogram or kg) (2)

γ : is an acceleration (meter per second or m/s)

We will take the state of the car as its velocity with the symbol v

The driver need to control the speed of his car, for that he has to control the gas pedal (or the brake), and that will be the input of the car as a system. We take the symbol of the input as u, also for having such an input the driver has to apply some force in the gas pedal we name it F

The force as what we have just written in the second newton's law. The ratio between F and u will give us β which is an electro-mechanical transmission coefficient (1). Finally we can write an equation resuming what we have just explained above:

$$F = \beta u \qquad (4.6)$$

Now because we have the same force in the precedent equations we deduce the following:

$$m\gamma = \beta u \qquad (4.7)$$

For the acceleration we have $\dot{v} = \gamma$ when we replace:

$$m\dot{v} = \beta u \Rightarrow \dot{v} = \frac{\beta}{m}u \qquad (4.8)$$

This deferential equation describe to us how could the input may influence the system.

IV.6.Control design

Now we assume that we have taken the output measure as $y = v$. We have also taken the reference velocity as r , and after making the comparison between the input and the output we obtain the force as what we have just write in the second newton's law the error dictated as: $e = r - v$. The control signal design should be always a function of the error e, for that the control design should have some properties we have to respect. The first property that the controller should not over react that means if we have a small error we should not have a large control signal because we will be close to reach the reference velocity, and then we take a risk to destabilize our system. We should be nice and slow when we are close to reach the reference target.

The second property that we shouldn't be very rapid all the time, for example if the road is slippery like if there is some oil in the ground, then we have to slow down our vehicle otherwise we take a risk to slip and go outside the road.

For the autopilots and the aircrafts there is limits on the exact acceptable accelerations that are directly related to cups of coffees standing on the tray tables in the aircraft [1]. For the third property u should not depend on us knowing β (described in (1)) and m (described in (2)) exactly, that means the controllers has to be built robust to variations of mass across a certain spectrum. It is the same thing for the car or the elevator that the cruise controller should work no matter how many people are inside.

Now we want to reach the reference velocity what means, as soon as we get closer to the required reference as soon as we get the error canceled, in other way: when $v \to r$ as $t \to \infty \Longrightarrow e \longrightarrow 0 \qquad (4.9)$

IV.6.1.Basics control design

$$u = \begin{cases} u_{max} & if\ e > 0 \\ -u_{max} & if\ e < 0 \\ 0 & if\ e = 0 \end{cases} \qquad (4.10)$$

If $e > 0$ that means the reference velocity is bigger than the state velocity, which means that we are driving slower, then, in that case we should then have to accelerate.

If $e < 0$ that means the reference velocity is shorter than the states velocity, which means that we are driving faster, then we should brake.

If $e = 0$ that means we have reached the reference velocity.in that case we are in perfect drive, then we should do nothing.

The switching between two max called bang-bang control.

IV.7.The regulators

IV.7.1.P-regulator

The controller over react to small errors, here the regulator is nice and smooth and also stable.

First we make: $u = ke$ (4.11)

We know as it is supposed that small errors yield small control signal. What we mean with smooth and nice we have just mentioned above that we are not widely fluctuating our controller. And in fact we called it proportional regulator [1], because u is directly proportional to the error e threw the k control gain. And the problem here that we are not reaching the reference we want.

$\dot{v} = \frac{\beta}{m}u - \rho v$ (4.12)

If we are going fast that means we will going to face a wind resistance negative force, and it will be against our direction drive. And that explain the minus which is written in the

-equation (4.12). For the ρ coefficient, we don't know its real value.

In the real environment the system (like the car here) will face a wind resistance. And we add that to our model as it is shown in (4.12):

At steady-state v is not changing any more. That means $\dot{v}=0$. If we replace the error $e=r-v$ in the equation (4.12), then we will have:

$$\begin{cases}\dot{v}=\frac{\beta}{m}k(r-v)-\rho v \\ \dot{v}=0\end{cases} \Rightarrow v=\frac{\beta k}{\beta k+m\rho}r<r \qquad (4.13)$$

From the model we understand that in steady-state, the state velocity is strictly less than the reference velocity. In other words we will never reach really the reference velocity, and that even if we make the gain k stronger and bigger. In the end the problem is that the proportional controller regulator can't overcome to reach tracking.

We try to apply Laplace transform to the dynamical equation, and we get a transfer function shown in (4.14):

$$\dot{v}=\frac{\beta}{m}k(r-v)-\rho v \Longrightarrow sv=\frac{\beta}{m}kr-\frac{\beta}{m}kv-\rho v \qquad (4.14)$$

$$\Rightarrow \left(s+\frac{\beta}{m}k+\rho\right)v=\frac{\beta}{m}kr \qquad (4.15)$$

$$\Rightarrow \frac{v}{r}=\frac{\frac{\frac{\beta k}{m}}{\frac{\beta k}{m}+\rho}}{\frac{1}{\frac{\beta k}{m}+\rho}s+1}=Gs \qquad (4.16)$$

Now we can fix some values for the parameters of Gs. We can choose for example:

$\beta=1, m=1, k=1, \rho=0.1$. we obtain the following transfer function equation:

IV.7.1.1.Make a simulation for the P-regulator by using Simulink

$$Gs=\frac{0.91}{0.91s+1} \qquad (4.17)$$

We have done both the simulation and the result of the simulation in the $Fig\ IV.2$

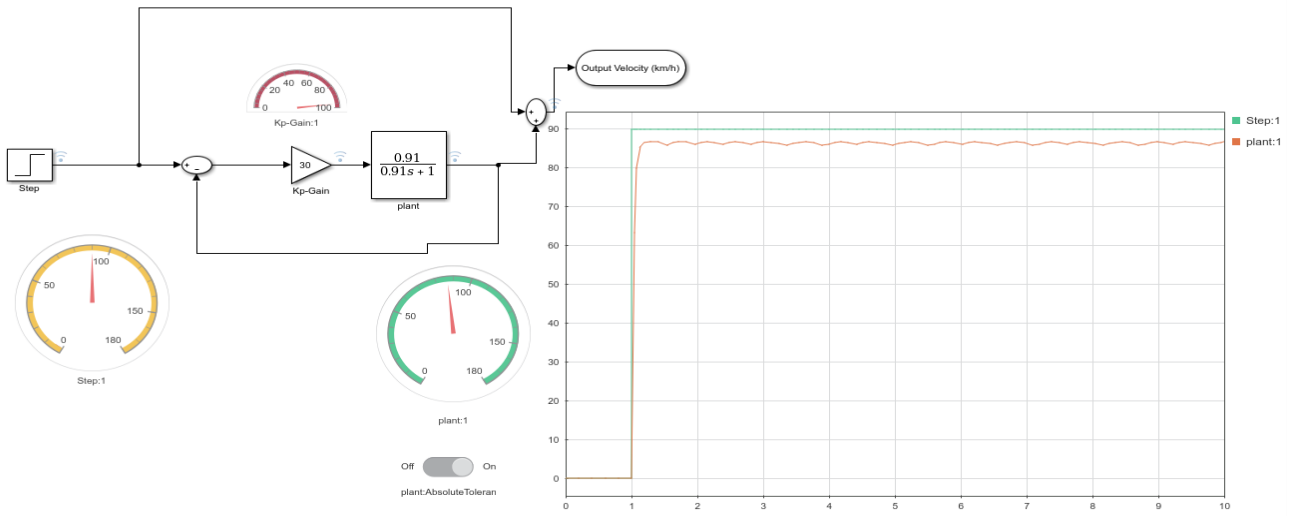


Fig IV. 2. Proportional regulator with simulink

IV.7.1.2.Make a program for the P-regulator by using Matlab.

Now we use a *Matlab* program

```
%P − controller program
 Clear all
Clc
 num = [0.91]
 denom = [0.91 1]
 G_s = tf(num, denom)
 H = [1]
 M = feedback(G_p, H)
 step(M)
 hold on
%%
```

$K_p = 30$

$K_i = 0$

$K_d = 0$

$G_c = pid(K_p, K_i, K_d)$

$M_c = feedback(G_c * G_s, H)$

$step(M_c)$

hold on

$tilte('\text{step response using a P} - \text{regulator}')$

We can see the result of the simulation in the *Fig IV*. 3

IV.7.1.3.Result of the simulation of the Matlab program

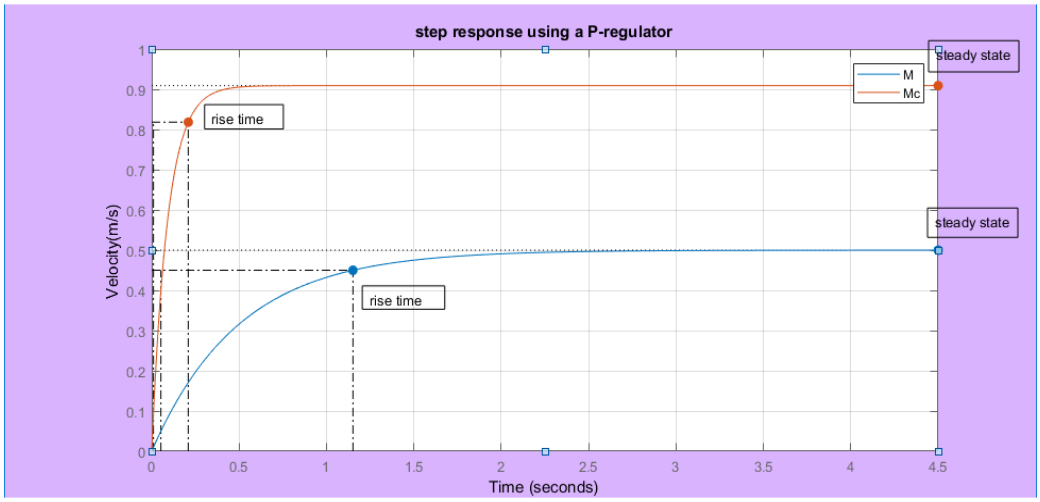


Fig IV. 3. *Proportional regulator with Matlab*

IV.7.2.PI-regulator

Now we try to cancel the wind resistance that means we are going to have a new controller. We can do that by adding to a new model the following:

$$u = ke + \rho\frac{m}{\beta}v \qquad (4.18)$$

We multiply now the equation (4.18) by $\frac{\beta}{m}$, and we replace in the equation (4.12), also when the system reach the steady-state that will lead us to the following result:

$$\dot{v} = 0 = \frac{\beta}{m}k(r - v) + \rho v - \rho v \Rightarrow v = r \qquad (4.19)$$, that give us a tracking.

But we don't know the physical parameters $\beta, m \text{ and } \rho$, so this is not a robust control design what also means that the tracking is not perfect. That means too a failure. As we have seen in the proportional regulator that the proportional error is pushing the system up, too close to where it should be, but it can't push hard enough to overcome the effect of the wind resistance.

Now if we find a way to collect all the accumulated errors over time that should be enough to push all the way up. And the way for that specific task is to use the integral.

IV.7.2.1.Make a simulation for the PI-regulator by using Simulink.

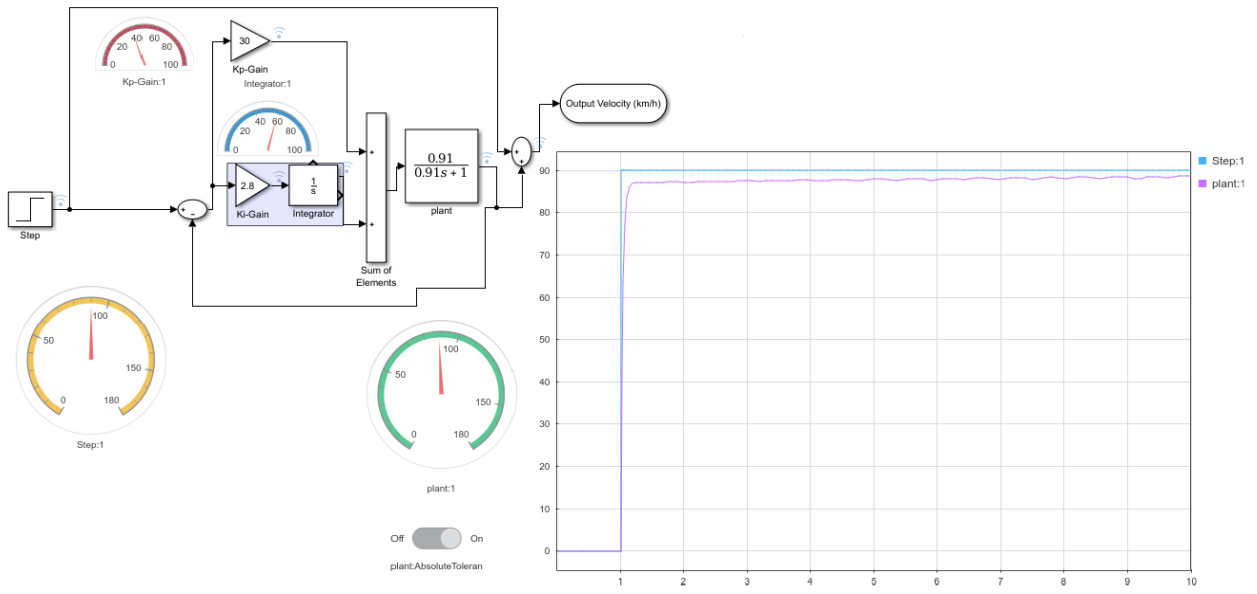


Fig IV.4. Proportional Integral controller with simulink

If we take the integral over the error we are collecting the error over time, and that gives enough pushing power to overcome the wind resistance. So for the proportional integral regulator we take the following model:

$$u = k_P e(t) + k_I \int_0^t e(\tau) d\tau \qquad (4.20)$$

With the PI regulator we get quickly to the reference value. We have reached tracking. Because we don't know parameters. The system is robust and does not crashing, then we can say the system is stable.

IV.7.3.PID-regulator

$$u(t) = k_P e(t) + k_I i \int_0^t e(\tau) d\tau + k_D \frac{de(t)}{dt} \qquad (4.21)$$

We have seen for the produced effect of the regulators toward the system that the proportional regulator contribute to stability and medium-rate responsiveness, but it is not enough to achieve tracking.

We find for the integral regulator that is really good for tracking. In fact if our system is stable then having an integral regulator is enough to ensure tracking in almost all cases. It is also extremely affective at rejecting disturbances, it has to accumulate error over time to respond to them because it is an integral. So it is responding in slower rate, also if we take its gain k_I too large that surely will produce oscillations.

Since it's not responding to the actual error values but the changing in error values, it's typically faster responsiveness. Also we have the derivative is sensitive to noise.

We don't have to over react to noise with making the derivative gain k_D too large. In the end putting all that together we can have PID controller. We find in almost control applications we are going to process that the PID shows up in some form or another. We find in complex systems like robotics the stability is not guaranteed all the time with the PID regulators which is sometimes is not enough by itself. So we need a lot of thinking and modeling to use it. However it is a very useful type of controller even sometimes we don't know how to pick these gains. But we have the feedback having the ability to fight

-uncertainty in model parameters in remarkable way. The feedback has the ability to overcome the fact that we don't know the parameters as it is the case in our example, we do not know β, ρ, and m

IV.7.3.1.Make a simulation for the PID-regulator by using Simulink.

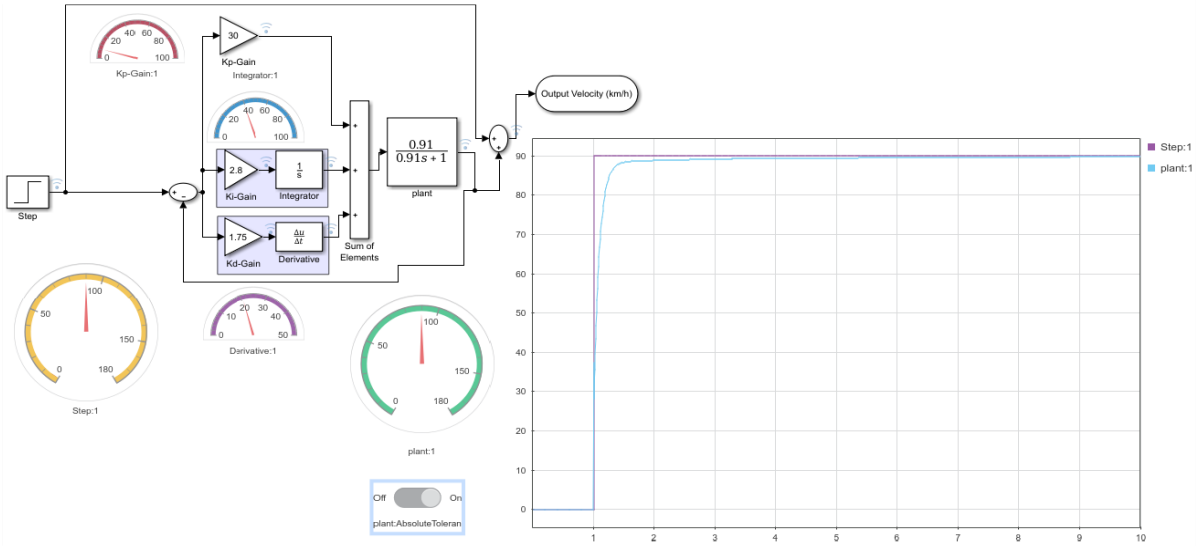


Fig IV.5. Proportional Integral Derivative controller with simulink

IV.7.4.Using Bernard Riemann sum

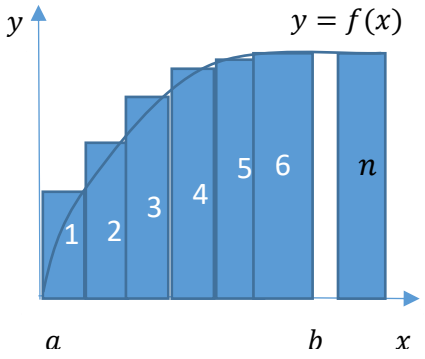


Fig IV.6. Bernard Riemann sum

We can see in the Fig IV.6 that we can approximate the area under the curve by breaking that area into rectangles and finding the sum of the area of those rectangles as the

-approximation. We have each rectangle an equal width so we equal a partition to the interval between two boundaries (a *and* b), and the height of the rectangle is the function evaluated at the left point of each rectangle. And we want to generalize it and write a segmentation and it looks something like (4.22):

$\sum_{i=1}^{n} f(x_{i-1})\Delta x$ (4.22), where $\Delta x = \frac{b-a}{n}$ (4.23) , we have just made trapezoids in the $Fig\ IV.6$, we don't even necessarily equaled the partitioned spaces, we just want to construct our figure simpler. If we take for one rectangle as one instance of the Riemann sum and we have n instances then the more the larger n the best the better an approximation it's going to be. So the Riemann definition of the integral which is the actual area under the curve between a and b is to take the Riemann sum and take the limit as n approach the infinity, we can write that as the following:

$\lim_{n\to\infty} \sum_{i=1}^{n} f(x_{i-1})\Delta x$; We can take in the case dx as infinitely small Δx but not zero, in the end we can right the approximation of the Riemann sum as in (4.24):

$$\lim_{n\to\infty} \sum_{i=1}^{n} f(x_{i-1})\Delta x = \int_a^b f(x)dx \qquad (4.24)$$

Now we can use such a method to get our integral for the error. As we have just seen for the PID controller:

$$u(t) = k_P e(t) + k_I \int_0^t e(\tau)d\tau + k_D \frac{de(t)}{dt} \qquad (4.25)$$

We can write the derivative error as in (4.26):

$\dot{e} = \frac{e_{final} - e_{initial}}{\Delta t}$ (4.26) , where Δt is a sample time.

For the integral we have: $\int_0^t e(\tau)d\tau \approx \sum_0^N e(k\Delta t)\Delta t = \Delta t E$ (4.27)

$$\Delta E_{final} = \Delta t \sum_{k=1}^{N+1} e(k\Delta t) = \Delta t(e(N+1)\Delta t) + \Delta t E_{initial} \qquad (4.28)$$

$$E_{final} = E_{initial} + e \qquad (4.29)$$

IV.8.Using Regulators that permit to a robotic car to follow a trajectory

If we want to follow a line and we are too far to the left, we turn to the right and vice versa, but how much do we turn if the steering wheel start to fix them out the left to the right, the approach is called bang-bang control. For controlling a car (or a robot) this doesn't actually work that well, the response is very jerky and uncomfortable for passengers. Fortunately we don't have to steer like this in a car since there is a range of angle the steering wheel can take.

One way to set the steering angle is to use what is called proportional control rather than turn the wheel and fix them out. Proportional control steers harder the farther away we are from the desired trajectory. We take a measurement of the cross track error e_p to define how far away from the desired trajectory the vehicle is. Therefor the steering angle we use is the track error multiplied by a scaling factor called proportional gain P. The range of values that the proportional gain can take drastically alters the performance of the vehicle. The performance gets better as the gain increases, but if we start far away from the desired trajectory with a high gain the system can spin out of control. With best gains proportional control returns better results than bang-bang control but it still work not good. With proportional control, a car can quickly reaches the center line. The result of this that the controller will repetitively overshoot the actual desired trajectory and not actually follow it. To correct that overshooting problem we need to consider additional error measurements and use them to update our steering command.

The good candidate is to look at the cross track error rate e_D or in other words how fast we are moving in a perpendicular direction for respect to the desired trajectory.

If we perfectly found the trajectory our cross track error will be zero. That is what called in control theory the derivative term. This rate term can be multiplied by its own gain and added to the proportional term to construct an updated controller. Now then we have two terms, we have two gains that we must be tune simultaneously. Conceptually we can think

-that increasing the proportional gain will increase the pull that the vehicle feels for its desired trajectory. Increases the derivative gain increases the resistance the car will feel against moving too quickly towards the line. Fixing the proportional gain, if the derivative is too low the system will be what is called underdamped and it will still oscillating. If the derivative gain is too high the system will be what is called overdamped and will take a lot of time to correct for offsets. Properly choosing the derivative gain allows the car to approach the desired trajectory quickly with a cross track error rate close to zero, this is called critically damped. Despite the success so far for complete controller we haven't done yet.

Environmental factors or mechanical defects can change the vehicle behavior known on the vehicle and thus the performance of the controller. If there is a heavy cross wind the vehicle will drift and thus unless the driver counter axes wind force with the corrective steering command. If we take an example for a vehicle driving along and hits a parallel rocks which lay in front of an alinement and therefor a zero steering command no longer keeps the vehicle driving straight. As we can see with the controller that has been described so far the vehicle experiences build up a lane offset or a steady state error. One way to redress this problem is to add what is called integral term and that is written as $I.e_I$. Now we have the $steering\ angle = P.e_P + D.e_D + I.e_I$. This third measurement sums of the cross track error giving the indication if we are spending more time on one side of the trajectory or the other. We can see that if we sum up the track errors we obviously spend more time on one side of the trajectory. The integral term we propose is than exactly the sum multiplied by a gain. Now three gains will need to be tuned all of ones which can be quite difficult to do by hand.

If the gain is too large the controller can go instable because normal controller fluctuations will be exaggerated.

If the gain is too small it can take too long to respond to these dynamic changes. When the gain is just ruddy the controller will be able to quickly correct for finding trajectory alinement and return to its nominal performance. The combination of the three terms is then what is referred to in control theory as the PID control [14].

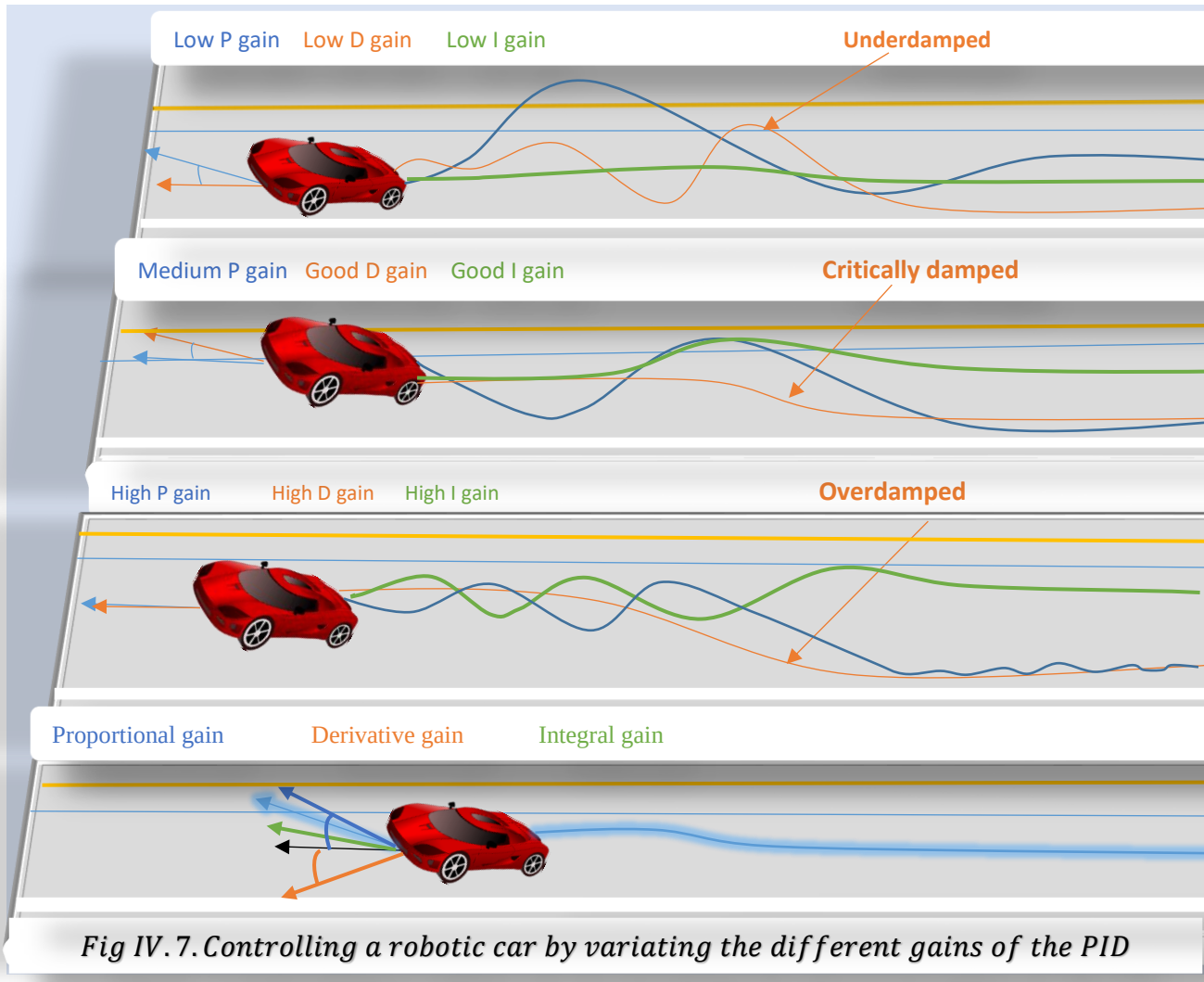


Fig IV.7. Controlling a robotic car by variating the different gains of the PID

IV.9.Using different mode of controllers to Control a robotic car

IV.9.1.The proportional integral derivative controller mode

Here we have a simplified block diagram of what PID controller does:

This system represent a Driver changing lanes on a freeway in a windy day. We assume that we are the driver, and therefore the controller of the process of changing the cars position.

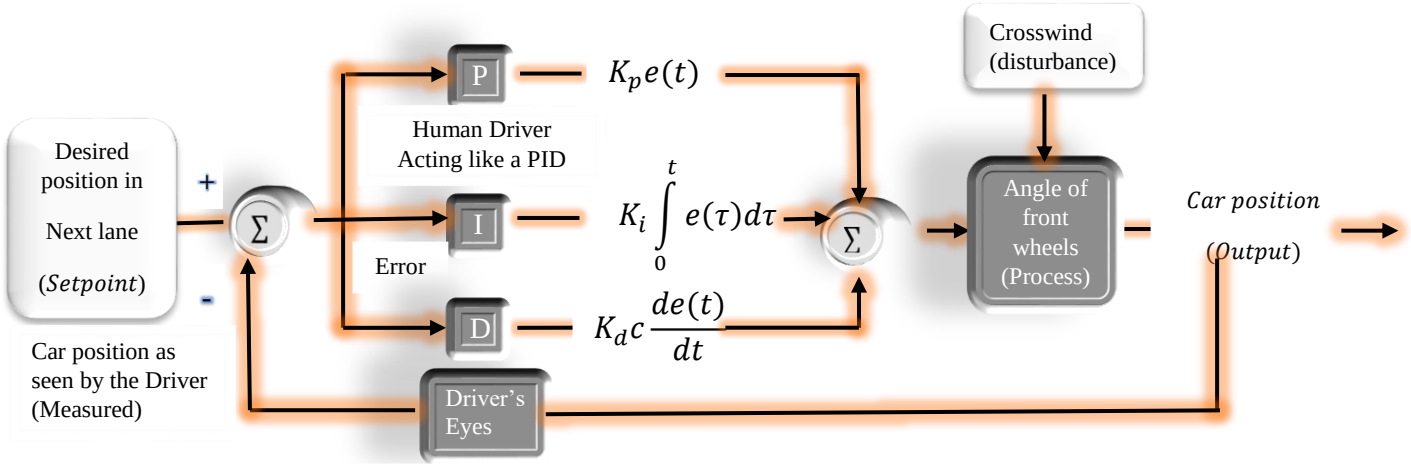


Fig IV. 8. Controlling the robotic car by using PID controller

Notice how important closing the feedback loop is. If we remove the feedback loop we would be in "open loop control", and would have to control the car's position with our eyes closed!

The value that we want the process to be is the *Setpoint*.

The Output must be equal to the *Setpoint*, else the error signal would not be Zero.

The three gain (P, I and D) will be summed together to output one signal that will get the output equal to the *setpoint* (in other words get the output signal to zero).

The process is the Plant/Model of the system. For example the DC motor.

We add a disturbance to the system. Like cross wind or the friction to the shaft of the motor.

The *Setpoint* subtracted from the Measured to create the Error.

The error is simply multiplied by one, two or all of calculated P, I and D actions (depending which ones are tuned on).

Then the resulting "*error* × *control action*" are added together and send to the controller output.

These three modes are used in different combinations: P - Sometimes used; PI - Most often used; PID-Sometimes used; PD-Rare, can be useful for controlling servomotors.

IV.9.2.The proportional controller mode

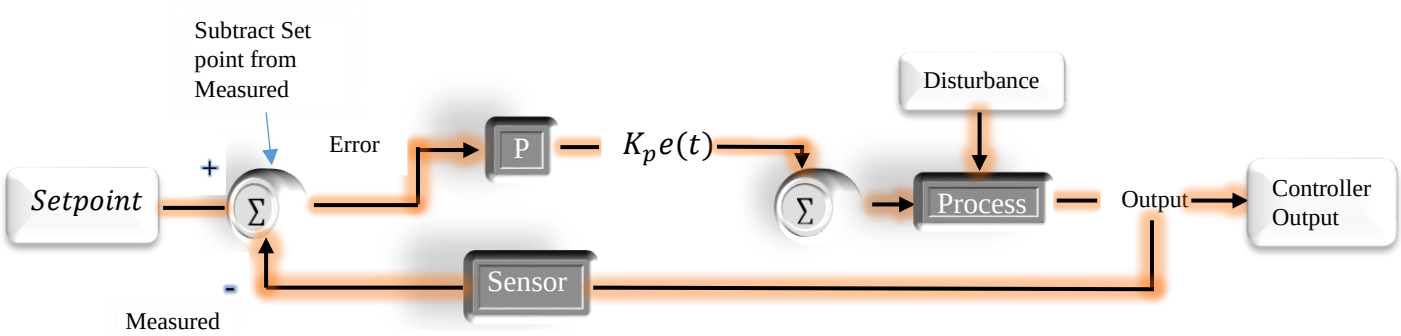


Fig. IV. 9. Controlling the robotic car by using P controller

In proportional mode, the controller simply multiplied the error by the proportional gain k_p to get the controller output.

Small proportional gain is the safest way to get to *setpoint*, but our controller performance will be slow. If the k_p is increased, overshoot in the signal will be present.

IV.9.3.The proportional derivative controller mode

In proportional derivative mode, the controller make the following:

$$u = k_p e(t) + k_d \int_0^t \frac{d}{dt} e(t) \qquad (4.30)$$

Multiply the error by the proportional gain k_p and added to the derivative error multiplied by k_d, to get the controller output.

The derivative term slows the rate of change of the controller output and this effect is most noticeable close to *controller setpoint*. Hence, derivative control is used to reduce the magnitude of the overshoot produced by the integral component and improve the stability.

However differentiation of a signal amplifies noise and thus this is highly sensitive to noise in the error term, and cause a process to become unstable.

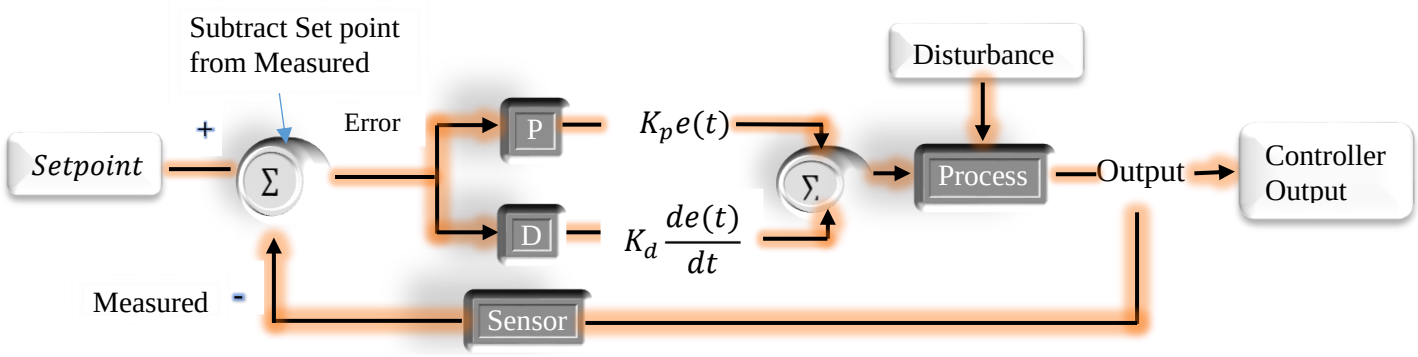


Fig. IV. 10. Controlling the robotic car by using PD controller

IV.9.4.The proportional integral controller mode

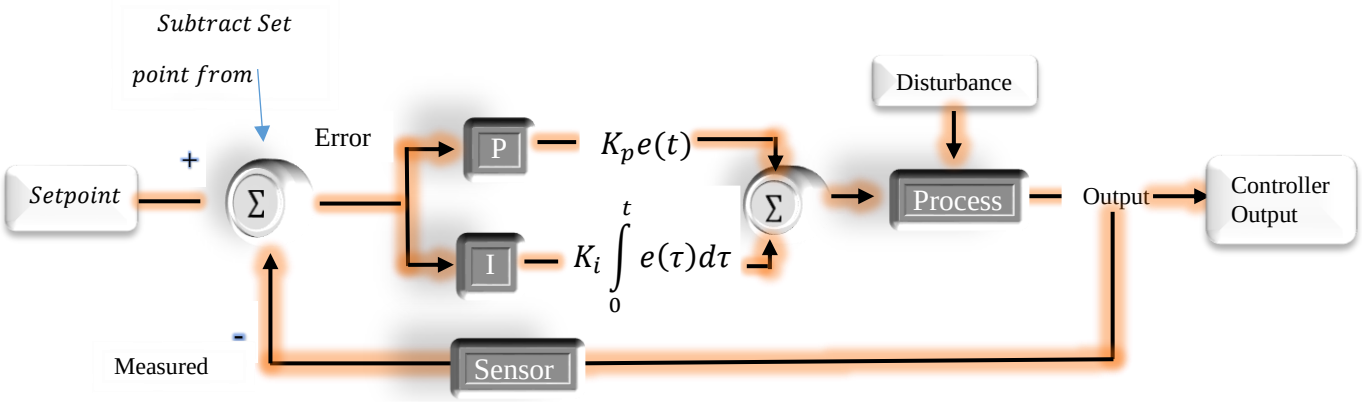


Fig. IV. 11. Controlling the robotic car by using PI controller

In proportional integral mode, the controller make the following:

$$u = k_p e(t) + k_i \int_0^t e(\tau)d\tau \qquad (4.31)$$

Multiplies the Error by the proportional gain (k_p) and added to the integral error multiplied by k_i , to get the controller output.

The integral term (when added to the proportional term) accelerate the movement of the process towards *setpoint* and eliminate the residual steady-state error that occurs with

-only a proportional controller. However, since the integral term is responding to accumulated errors from the past, it can cause the present value to overshoot the *setpoint* value (cross over *setpoint* and then create a deviation in the other direction).

IV.10.Summary characteristics

PD: _Compensator is anticipatory, it respond to the error and its derivative.

_Phase lead is provided starting one decade below the zero.

_Generally increase damping.

_Generally, reduce rise and settling times.

_Increase band width.

_Increase phase and gain margins.

_May render a system susceptible to high frequency noise.

_Act as a high-pass filter.

PI: _Compensator increases the system type by one, which helps with error control.

_Increases phase-lag at low frequencies.

_Generally, increase damping, rise times, and settling times and reduce overshoot.

_Decrease band width.

_Not sensitive to high frequency noise.

_Act as a low-pass filter.

PID: _Combined effects of PI and PD compensation.

_Cascade of a PI and PD compensator.

IV.11.Other example explaining what is PID controller and how it works

A lot of things need to be controlled if our robotic car speeds up or slows down, we need to correct it to come up back to the wanted speed. Many of today's machines are controlled by automatic systems that keeps them doing what should they do to achieve the right result.

To explain how the PID work we can imagine we are driving our robotic car in a flat straight road trying to maintain ninety kilometer per hour.

We ultimately press and release the accelerate pedal according to the speed in the speedometer. We use a proportional control to maintain our speed. If we drive that robotic car in a sloping road then we see that the speed quickly dropping. We need to accelerate quickly to counteract that dropping speed. The fact the speed is dropping the more the acceleration we apply, that is what we called a derivative control. So if we have a large difference in tract of our actual speed we use a proportional control to apply correction. If we solemnly change the speed, the derivative control is used. However if we only have small fixed error between target speed and actual speed, proportional element is too small to be effective and because there is no changing in speed the derivative contribution is zero. So that small error stays incorrected. The error whatever is small will accumulate big enough that we need to take a corrective action, that is called integral control.

We can see the term proportional as the present or what is going right now that means the error will be corrected only in the present time, it is a limited feedback correction. In case of steering wheels, if we have got a slight tilt angle then the amount of correction being applied by the proportional feedback is very small too. If it is small we might not get any actual correction carrying it at all, what that means is that it can actually just hold a little offset for quite a long time, because the faster we get to the right position the less feedback we have getting from the proportional feedback system. It means big angle will give us a lot of feedback, small angle no much feedback.

And if over time we find the angle has been sustained so that is not right then we need to make correction. We need to introduce another corrective feedback force which will force it to approach the expected right position. So where the proportional is constant the integral *feedback over time increases until it is enough forced to push the angle to the right position.* We can't use proportional correction on its own without giving us oscillations. And we can't use integral on its own.

It is too slow because it take too long to get into the right point so we can stop the wheels from giving us undesirable correction angle after applying a corrective force.

Now we add a derivative correction. The term derivative are not looking to the present or to the past but it looks toward the future as it has not happened yet but it is going to. It looks out for what will happened next. For example in the case of driving a mobile robot, if we have got a front wheels rapidly rotating and that may will give us a big angle we don't want, then we need to apply correction to stop it just in right time by anticipating the movement of the wheels. But we don't have to wait too long in order to prevent overshooting.

In the end, if we have the proportional, the integral and the derivative controllers all mix together, in other words using PID controller that will stop the angle at the desired angle position where it is supposed to be, and that has to be done in the right time.

IV.12.Conclusion

For the control theory using a PID controller is very helpful for so many processes, but it will be going more complicated to use it when it is something having relations with much more complicated systems like robotics. So there will be a lot of additional work we have to do in order to take control of our system.

Chapter V. Simulation Using Microsoft Visual Programing Language

V.1.Introducion

Microsoft Visual Programming Language (VPL) [46] is an application development environment designed on a graphical dataflow-based programming model. A dataflow program when executed is more like having a series of workers on an assembly line in a factory, who do their assigned task as the material arrive. As result VPL is well suited to programming a variety of concurrent or distributed processing scenarios.

VPL is targeted for beginner programmers with basic understanding of concepts like variables and logic.

The programming language may appeal to more advanced programmers for rapid prototyping or code development [45]. As a result, VPL may appeal to a wide audience of users including students, enthusiast/hobbyists, as well as possibly web developers and professional programmers. A VPL dataflow consist of a connected sequence of activities represented as blocks with input send outputs that can be connected to our activity blocks.

From the diagram we can see that the activities blocks have connections that represent messages sent from one activity to another. Activities [48] can represent dataflow control or processing functions, prebuilt DSS [45] services or our own defined activities we can create in VPL. The connection between activities pass data as messages. The resulting application is therefore often referred to as dataflow orchestration, the coordination of information between a connected set of processes.

Activities can also include constructions of other activities [47]. This makes it possible to compose our own activities by combining existing ones and then reuse the composition as a building block. In this sense an application build in VPL is itself an activity.

An activity block typically includes the activity's name and its connection points. It may also include graphics to illustrate the purpose of the activity or user interface elements like text boxes that may enable the user to input values, assignments, or transformations for data used in an activity.

Connections invoke actions such as Say Text. Activities are connected through connection pins. A connection pin on the down left side of the diagram an activity represent-

-the connection point for incoming/input messages and pin on the down right side of the diagram represents the connection point for outgoing/output messages.

An activity receives messages containing data through its input connection pins. An activity's input pin are connection points to its predefined internal functions or handlers. These can be either as functions provided by a service or nested data flows.

An activity block activates and processes the incoming messages. All data sent to the activity is consumed by the activity. For data to be forwarded through the activity's output, the receiving activity must replicate the data and put it on its output connection.

An activity may have multiple input connection pins, each with its own set of output connection pins. Input pins are represented as arrows pointing into the block. Output connection pins can be one of two kinds: result output or notification output. Result output are displayed as arrows pointing out of the block. While notification outputs display as round connection pins.

A response output connection pin is used in situations where the outgoing message data is sent as the result of a specific incoming *action* (or request) message. Notification connection pins can send information resulting from an incoming message, but more typically send a message as a result of an internal change to their state. Output pins can be connected to input pins on more than one other activity. Response and notification output connections also differ in how they deliver message data. Since a response connection requires an incoming message, once that request is received and processed and resulting message is sent, it is finished. However, notification outputs can generate messages multiple times.

As a result, response connections are typically used to query the current state of an activity, whereas notification output pins are used for providing continuous updates on an activity's state, eliminating the need to repeatedly request or *poll* for the state of an activity. The activity defines which types of connections which are provided.

V.2.The mobile robot model used for the simulation

This Pioneer 3-DX is a modular mobile robot offering various options like a gripper or an

-on board Camera. It is widely used in research labs. It is a small but very durable and robust robot. It comes with a SICK LMS200 Laser Range Finder installed on top and has a 360 degree sonar and bumpers array in the middle and bottom of the robot respectively [45].

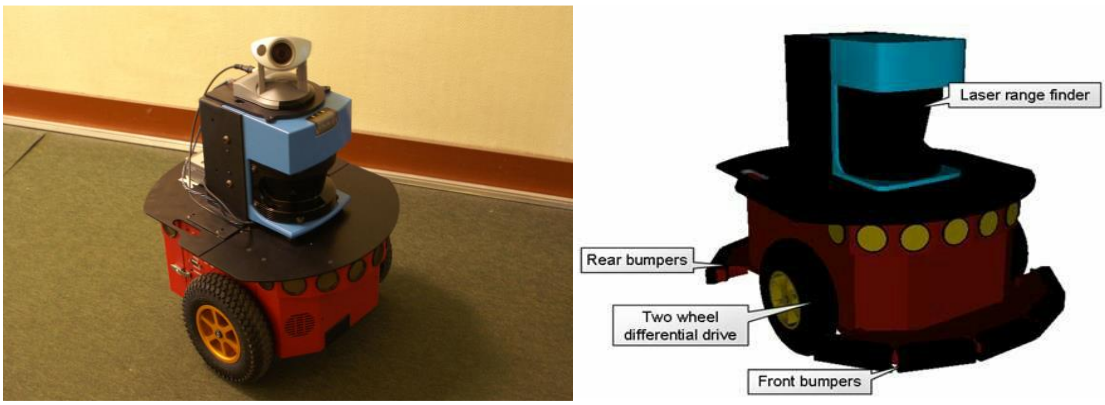


Fig V. 1. Pioneer 3DX used for the experiences in real envirenment equiped with a cannon camera VC − C50i and a laser telemeter

V. 3. Visual servoing for a robotic system components

We can show in the following figure how to control a pioneer DX3 [5] [4] [45] using a feedback in the closed loop we have built

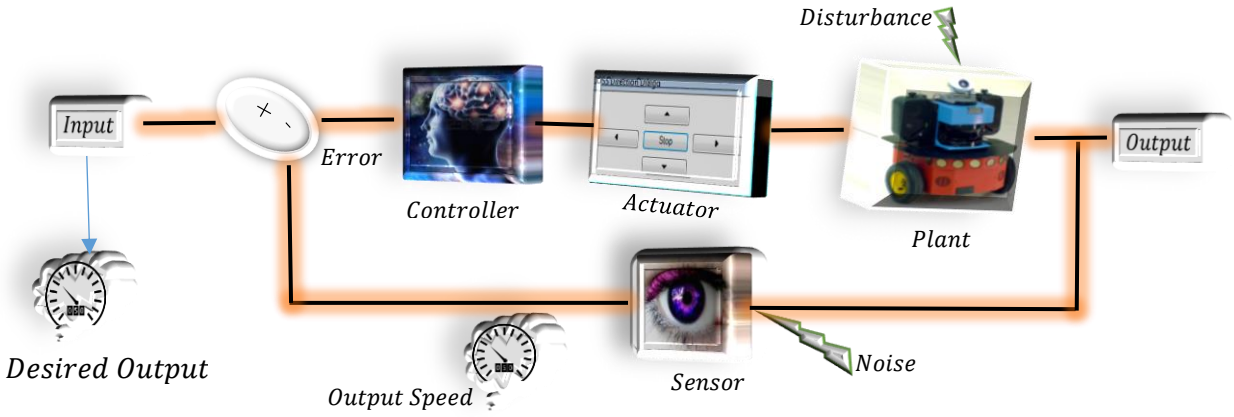


Fig V. 2. visual servoing for a robotic system component.

V. 4. Controlling the robot by using Direction Dialog

V. 4. 1. Direction Dialog

The Direction Dialog service (as we see it in the Fig V.3 and Fig V.7) provides a simple dialog window with a set of five predefined directional buttons (and a Stop button). However, the Direction Dialog by itself does not control anything. It only generates notification messages whenever the buttons on the dialog are pressed, but we can create a diagram to use the button press notifications for whatever purpose we require.

When we run our diagram the Direction Dialog service automatically starts and displays its window as shown in the Fig V.3, and Fig V.7. It does not have an operation that makes it to be displayed.

The Direction Dialog is intended as a simple example of using Windows forms as part of our VPL application.

V.4.1.1.Operations

***Table V. 1* Operation supported by the direction dialog**

The following operations are supported for this service.

Operation	*Description*
Get	*Gets the current state of the set of buttons in the dialog.*
DialogStateChange	*Changes the state of the buttons in the dialog.*
ButtonPress	*Indicates that a button in the dialog has been pressed.*
ButtonRelease	*Indicates that a button in the dialog has been released.*

The ButtonPress and ButtonRelease notifications return Direction and Name.

The Direction will return one of the following values:

- $ButtonDirection.Backwards$
- $ButtonDirection.Forwards$
- $ButtonDirection.Left$
- $ButtonDirection.Right$
- $ButtonDirection.Stop$

Name returns a string that contains the text equivalent line of the directions listed above, e.g. "Backwards", "Forwards", "Stop", "Left", and "Right".

When we receive notifications from the Direction Dialog, there are two ways we can handle them. First, we can use only the ButtonPress notifications and ignore Button Release. This has the effect of executing a task or action when we press a button. The second approach is to initiate the appropriate task or action when a ButtonPress notification is received, and terminate that task/action when the button is released. This has the effect that the associated task/action will only continue as long as we hold down a button.

Now we use another service to build our construct diagram and we can explain it as in the following:

V.4.2.Text to Speech

The Text to Speech service (as it appears in the Fig V.3) is designed to provide our applications with a verbal interface [47]. It can be used in conjunction with the Speech Recognizer service for two-way communication with the computer.

When the service is running we can inspect it by typing the following Uniform Resource Identifier (URI) into our web browser's address bar:

http://localhost:50000/texttospeech [20].
This displays the Text To Speech web page. It will show us the state of the speech object, as well as various other speech parameters. We can make changes using the web page, and also

-test out the TextToSpeech service by entering some text and pressing the Say button.

V.4.2.1.Requests

***Table V*.2**

The available operations for the Text To Speech service include the following:

Request	Description
SayText	Says the specified text
SayTextSych	Says the specified text but does not send a response until the speaking has finished. If we are using the service from VPL we might find this operation useful because program execution can be suspended until the response is received from the Text To Speech service.

V.4.3.Diagrams

V.4.3.1.Creating our Diagram

We start New from the File menu, then VPL displays a new diagram for us. We can drag and drop items from the Basic Actives or Services toolboxes to put on our new diagram.

If an activity or service has values we can set, we can typically change them by selecting the block and those values can be set or changed on the Properties window. For example, to - the name of a service we put on the diagram, selected it and then type its new name into the

Name field in the properties toolbox. Some pre-built activities (for example Data, If, Switch, Variable, Comment, and Calculate.) allow us to set values directly on the block.

If we drag a service from the Services toolbox that already exists in the diagram, we area sked whether we want to create a new instance of the service or create a reference to

-the existing service already in the diagram.

To create a dataflow connection from one activity or service to another, we can simply drag from the response or notification pin on block to an input connection pin of another block. In some cases, a service may offer multiple connections so we will be prompted for the message and the data to transfer through the message.

We can save the edits on our diagram, by choosing the save command on the file menu.

V.4.3.2.Loading our Diagram

We can load a diagram using the Open command on the File menu. This will replace any existing diagrams we have opened. If we have unsaved edits in the existing diagram, we will be asked if we want to save those changes. VPL also keeps track of our most recently saved projects and allows us to select them by name from the Recent Projects command on the File me.

V.4.4.Graphical program using Visual Programming Language with DSS Direction Dialog

In the graphical program we have made in Fig V.3, we use the DSS^1 service [45] [46]

As the following program shows it, we have chosen some values of the speeds we have to communicate to the wheels threw the “Data” service in order to make them move “Forwards”, “Backwards” , “Left”, ”Right” or even “stop” them if they were moving.

To make the robot goes forwards, we have to choose and put a positive value in the Data case. The value will be communicated to the wheels by the differential drive. We take a negative value if we want the robot to go backwards. And if we want the robot to stop or not

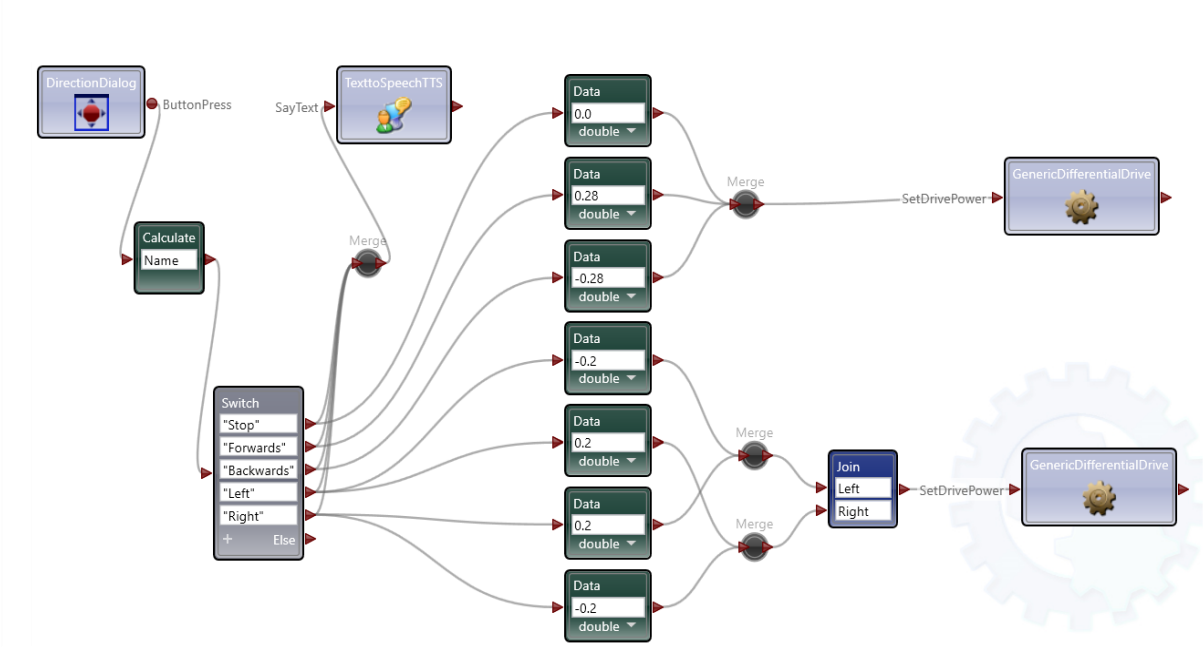


Fig V. 3. Program that controls the robot using DSS Direction Dialog

-moving at all, we have to choose and put a value as null.

For our experience we choose the positive value as "0.28" and the negative one as "-0.28". For the value taken as null we can simply write "0.0".

Now if we want to make the wheels rotate, we choose a positive and a negative value for each wheel. If we communicate a positive value to the right wheel, it will turn and go forward, and if we communicate to the left wheel a negative value, it will turn and go backward. That means the robot has to turn to the left. And if we communicate a negative value to the right wheel, it will turn and go backward, and if we communicate to the left wheel a positive value, it will turn and go forward. That means the robot has to turn to the right.

In our simulated program we have taken "0.2" as a positive value. And "- 0.2"as a negative

---

DSS[1] provides a communication infrastructure for services to run and still be able to communicate with each other and the host. It allows for services such as the laser range finder or the bumper array to "talk" to the host robot and it manages the messages and ports that these services uses to communicate. Without the DSS the services would send the messages but the robot would never receive them.

- value. We are free to choose the values as we want the robot to speed up or slow down.

We have taken a small values in order to control the rotation of the robot and make it goes in the direction we want. Otherwise we are afraid to see it turning around.

V.5.Controlling the robot by using Speech Recognition

V.5.1.Speech Recognizer

Speech Recognition (SR) convert spoken words to written text and as a result can be used to provide user interfaces that use spoken input [47]. The Speech Recognizer service (as we see in Fig V.6) enables us to include speech recognition support for our application. Speech recognition requires a special type of software, called an SR engine. The SR engine may be installed with the operating system.

V.5.1.1.Operations

Table V.3. Speech Recognizer Requests & Notifications

The Speech Recognizer services [48] supports the following requests and notifications.

Operation	*Description*
Get	Returns the entire state of the Speech Recognizer service.
InsertGrammarEntry	Inserts the specified entry of the supplied grammar into the current grammar dictionary. If certain entries exist already a Fault is returned and the whole operation fails without the current dictionary being modified at all.
UpdateGrammarEntry	Updates entries that already exist in the current grammar dictionary with the supplied grammar entries. If certain entries in the supplied grammar do not exist in the current dictionary

	no Fault is returned. Instead, only the existing entries are updated.
UpsertGrammarEntry	Inserts entries from the supplied grammar into the current dictionary if they do not exist yet or updates entries that already exist with entries from the supplied grammar.
DeleteGrammarEntry	Deletes those entries from the current grammar directory whose keys are equal to one of the supplied grammar entries. If a key from the supplied grammar entries does not exist in the current directory no Fault is returned, but any matching entries are deleted.
SetSrgsGrammarFile	Sets the grammar type to SRGS file and tries to load the specified file, which has to reside inside our application's /store folder (directory). If loading the file fails, a Fault is returned and the speech recognizer returns the state it was before it processed this request. SRGS grammars require Windows 7 and will not work with Windows Server 2003.

EmulateRecognize	Sets the SR engine to emulate speech input but by using Text (string). This is mostly used for testing and debugging.
Replace	Configures the speech recognizer service, or indicates that the service's configuration has been changed.
SpeechDetectedprocesse	Indicates that speech (audio) has been detected and is being processed
SpeechRecognized	Indicates that speech has been recognized.
SpeechRecognitionRejected	Indicates that speech was detected, but not recognized as one of the words or phrases in the current grammar dictionary. The duration of the speech is available as Duration in Ticks.

V. 5. 2. *SpeehRecognizerGui*

V.5.2.1.Definition

The *SpeechRecognizerGui* service (as it is shown in the Fig V.6) is a companion service that we can use with the Speech Recognizer service. Including the *SpeechRecognizerGui* service in our project enables us to enter a simple dictionary-style grammar [47] or to upload SRGS (Speech Recognition Grammar Specification) grammar files through a HTML page.

V.5.2.2.How to use *SpeechRecognizerGui*

To access the *SpeechRecognizerGui* service in VPL, we drag a copy of the service block into our diagram. It does not require any connections, and it will start up when we run the diagram. We can also optionally start an instance of the *SpeechRecognizerGui* once we have a DSS nod running by using a web browser and going to the Control Panel page. Starting the service will automatically attempt to load the default SR engine.

V.5.2.3.The use of web browser

Once the *SpeechRecognizerGui* service is running, we browse to the page for the service. To do this we start our browser and enter in http:/localhost:50000 (is the default port setting). Then we click on Service Directory in left column to display the list of running services. We should find the entry/ *SpeechRecognizerGui* in the list. We click this and we should see a page like in the *Fig V*. 4.

At the bottom of the *Speech Recognizer Gui* page we select either a simple dictionary style grammar or load an SRGS grammar. We click save to use the grammar. This will create a copy of the grammar file in the \ Store folder. If we chose to create a Dictionary grammar, the Save command creates a *SpeechRecognizer*. *c*. We have also the save command creates a SpeechRecognizer.config.xml file in the \Store folder. If we chose to load a SRGS grammar file, the file we browse to will be copied to \Store.

If we create a grammar using the *SpeechRecognizerGui* service and do not explicitly configure the Speech Recognizer service to load a grammar file, the Speech Recognizer service will automatically attempt to load and use the grammar file we have created.

The *SpeechRecognizerGui* service page also display the notification generated by the

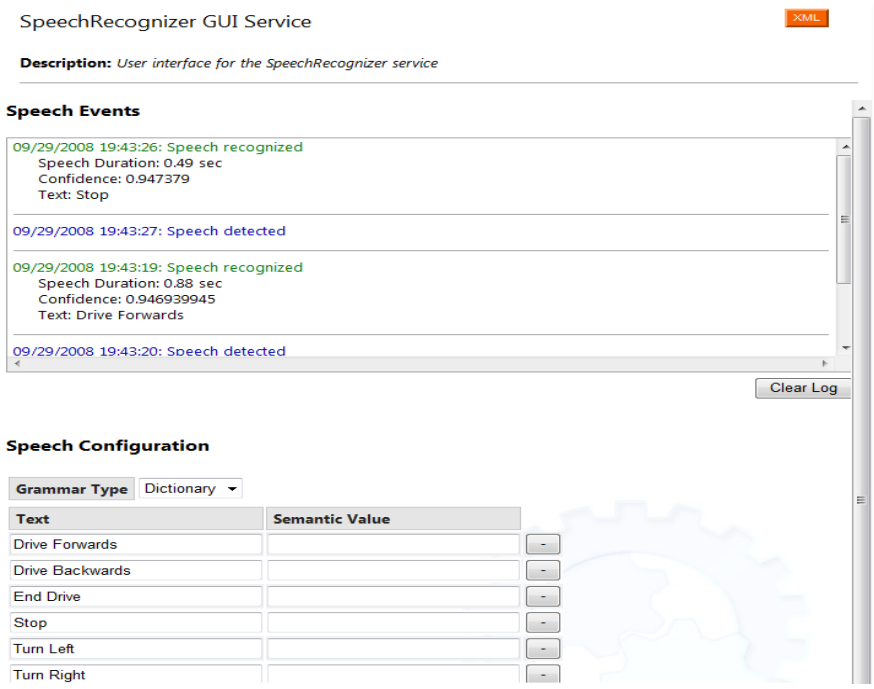


Fig V. 4. Speech Recognizer Gui – Service page

- SR engine such as speech detected or speech recognized in a scrolling area that can be cleared.

We note that the SR engine only recognizes words and phrases that are in its grammar. If the grammar is empty, then nothing will be recognized.

The *SpeechRecognizerGui* service is not designed to be sent requests or to issue notifications. It is only intended to be used via a web browser to build dictionary-style grammars and test speech recognition. Here we can run the DSS service after running the program, and that just with one click in the green button we have in the application. mvplSpeechRecognizerGui. For the robot to receive verbal input, we need to add the speech recognition service to our diagram. We select Speech Recognizer from the Services toolbox and we drag it onto the diagram.

The Speech Recognizer service uses a grammar that defines the words and phrases that should be recognized. Without a grammar, the Speech Recognizer does not recognize anything and our program will not work. The easiest way to set up a grammar is to run the ''SpeechRecognizerGui'' service which provides a web page as an interface.

We drag this service to the diagram as well. It does not require any connections, we just need to put it on the diagram so that it will be started when we run the program. We run the program (after saving it) with just these two blocks. Once the program has started, we open a web browser and we go to the URL http://localhost:50000speechrecognizergui [19] to see the interface to the Speech Recognizer service

V. 5. 3. *Graphical program using Visual Programming Language with SpeechRecognizerGui*

In the following we have made a program using *SpeechRecognizer* and *SpeechRecognizerGui*.

From the toolbox we drag the activity "If " and we write the instruction to communicate to the "*GenericDifferentialDrive*" service which transfer the orders to the wheels.

We put in each case of the "If " service the following: Value=="Forwards"; Value== "Backwards"; Value=="Turn Left"; Value =="Turn Right".

In the last case we write: value=="Stop".

Before we use *SpeechRecognizerGui* we can joystick the robot as we can explain that in the dashboard we have illustrated in the Fig V.6.We have first the motor was off, and the sensors for the Laser Range Finder are not activated, also the same thing for the port which was not connected too.

In the *Fig V*. 6 in order to use the Dashboard we have to connect the port of the robot, also we have to turn on the motor by clicking into the address we have in the Dashboard written -as the following: (P3DXMotorBase)/simulateddifferentialdrive/8ade296a-4c4e-bbae-3781332ca337/drive

Graphical program using Visual Programming Language with *SpeechRecognizerGui*

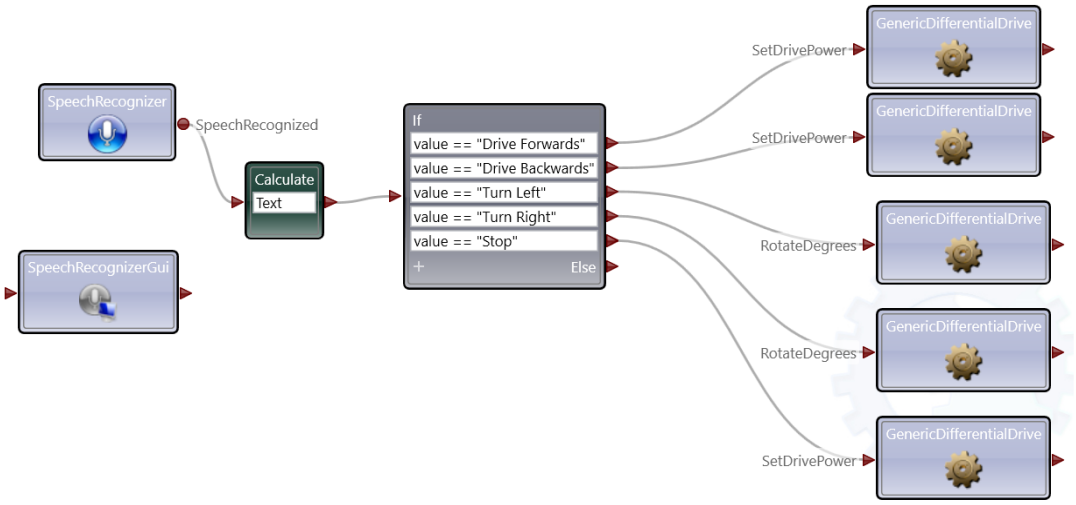


Fig V. 5. *Program that controls the robot using speech recognition*

For the Laser Range Finder we have to click in the following address:

(P3DXLaserRangeFinder)/simulatedlfr5e94ca32-ccfd-4aee-b6e8-4599202b1936/sicklrf.

That shows us each time when the robot has to approach to any obstacles and warn us that there is an object the robot has to avoid collision with. Otherwise the robot has to strike the obstacle in front of him and may smash on piece. Of course that may will happen when it is about a real robot.

V. 6. Result of the simulations

V. 6. 1. Visual result of the DSS Differential Drive and the speachRecognizerGui

In the following scene as the *Fig V*. 6 shows it, we can see a three dimensional sight of an environment which is the chosen place for the robot to move freely into the right direction, and that also by taking in account the obstacles the robot may face along the road to the supposed point we want it to reach. For that we need to use the Laser Range Finder Sensor what is indicated in the scene by the red line color. In the same way we can use the camera which shows us the whole environment containing the supposed field where the robot may run in and the all obstacles the robot may face on its way. Also we can use the integrated

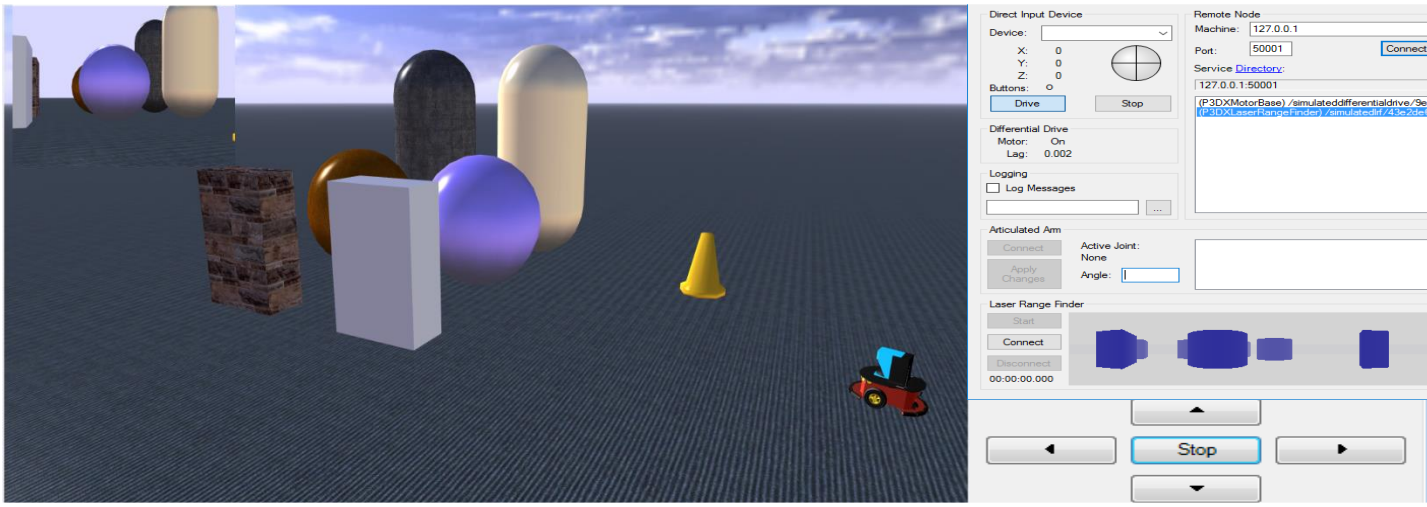


Fig V.6. Controlling pioneer DX3 by using DSS Direction Dialog or the Dashboard

-camera on the robot like the eyes which has to show us just the field the robot may see in front of its way.

V.6.2.Obstacle Avoidance

The Obstacle Avoidance Drive service is designed to simplify writing applications for a MARK robot. It performs sensor fusion on the IR, Sonar and Depth data to detect obstacles and then adjusts the robot's wheels speeds accordingly.

To use the Obstacle Avoidance Drive service with the Robot Dashboard, we simply substitute it for the existing drive service in the manifest and also partner the Obstacle Avoidance Drive service with the existing drive and proximity sensors. The resulting manifest will run the services as outlined in the *Fig V.7*:

The Obstacle Avoidance Drive service provides a Generic Differential Drive contract so that other services can use it in the same way as any other drive service. IR and Sonar sensors are optional partners for the Obstacle Avoidance Drive service, i.e. it will run even if these other services are not present. However, it does require a Kinect Depth Cam service in order to run.

The Reference Platform Simulation is already set up to use the Obstacle Avoidance Drive service. In the Start Menu, we select Microsoft Robotics Development Studio 4, then

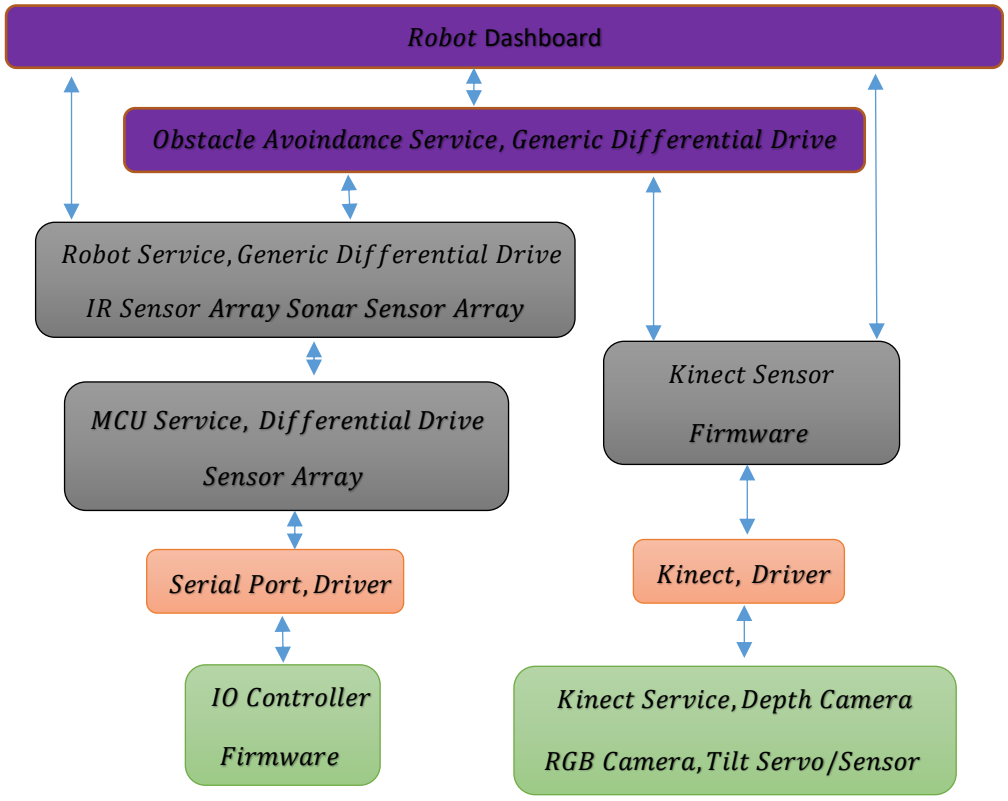


Fig V.7. using Obstacle Avoidance Service

- Visual Simulation Environment, and then Reference Platform 2011.

Obstacle Avoidance uses a PID (Proportional Integral and the Derivative) controller to steer the robot away from obstacles. We can adjust the PID parameters (K_p, K_i, and K_d) to change how quickly or slowly the robot will respond. This is a trial and error process that requires some skill as it is shown in the *Fig V*. 8

In order to understand what is happening, the window also includes a visualization often -sensor information. We can note that this window only updates while the robot is moving. Also we can see that when the simulation first starts, it will be blank.

At the top is a view of the depth data from the Kinect. The black area in front of the robot indicates that there is something too close to the Kinect to be able to determine its range.

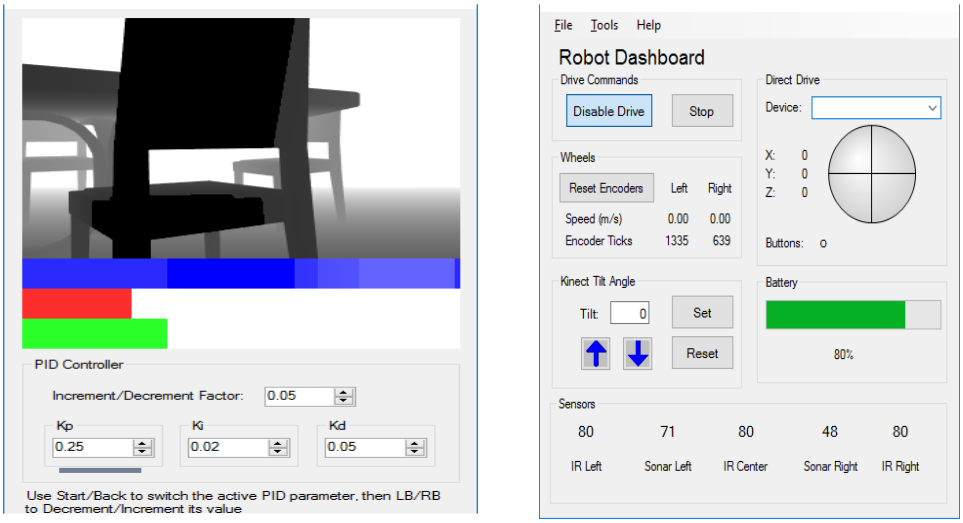


Fig V. 8. Obstacle avoidance window(left)and Robot Dashboard(right)

This is part of a chair. We can see the rest of the scene like the other chairs and a part of a table as it disappears after a certain distance and gets progressively lighter shades of gray.

The blue bar below the Kinect data shows where it is supposed the existence of any obstacles. The rise of the blue bar means the more likely that there is an obstacle. The red bar lights up when the IR sensors detect obstacles (left, center and right)

The green bar shows the sonar sensors (left and right). If these two bars are empty, then the proximity sensors are not detecting any obstacles and the robot will drive straight forward. As we have seen in this example, the robot will start to turn to the left or to the right to avoid the chair. When the simulation starts up, we have to enable the drive in the Robot Dashboard and push the thumbstick forwards on the Xbox controller (or use the on-screen trackball).

As long as we keep the thumbstick pushed forward the robot will wander around and avoid obstacles. We can give it suggestions by pushing the thumbstick left or right and then returning it to the forward position. If we hold the thumbstick left or right the robot will spin on the spot.

When the robot encounters a doorway, it might "hunt" from side to side for a little while before it drives through the door. This is most likely to happen if it was not lined up with the

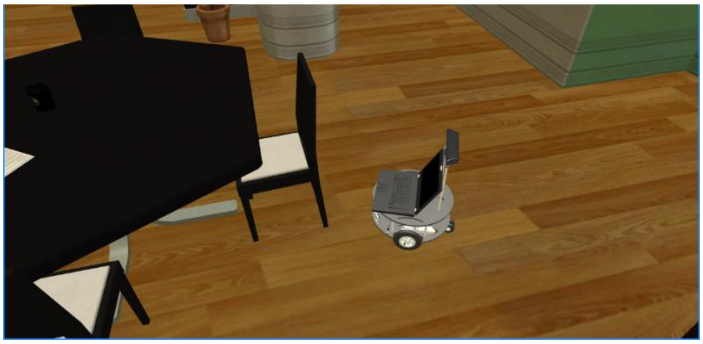


Fig V. 9. *webcam simulation using reference platform* 2011

-doorway to begin with. We can adjust this behavior using the PID parameters so that it is less sensitive to the door jambs on either side of the doorway, but this is a compromise. If we make it too insensitive it will tend to scrape against the doors and other obstacles

Conclusion:

We have seen in our work in this book that the control of the robot is not an easy task we have to accomplish, and that has to be done by using a lot of intelligence and that require a best control design. We need different type of controllers for each task the robot may have to accomplish. The P-regulator contribute to stability and medium-rate responsiveness, but it is not enough to achieve tracking.

For the control theory using a PID controller is very helpful for so many processes but it will be going more complicated to use it when it is something having relations with much more complicated systems like robotics. So there will be a lot of additional work we have to do in order to take control of our system.

This project help us to learn how to make mobile robots move in effective and safe ways using modern control theory. We also appreciate the value of systematic thinking and design.

We can enjoy and have fun to work with robots and spark further investigations. Without a model we can't say much for about how the system will behave. We need models to predict behavior forward in time, also we need models to be able to derive control laws in a

-systematic manner. The model should be rich enough to be relevant yet simple enough to be useful. Given a model, feedback control should be used to make the system behave the way we want it to (if possible). We need to reach stability, tracking and robustness. We have to state feedback and observers. We need architectures**:** we plan for simple systems, execute on the "real" system. We do the designs at different levels of three abstraction layers: we plan high level where we want to go then we generate reference paths, in this case we did use linear model and we track those using nonlinear models. And in particular we look differential drive robots. Finally we can say that we have to experiment and invent because the field is certainly not done yet.

By using Microsoft Visual Programming Language, we have seen so many opportunities the simulator has given to us to simulate any suggested experiment, and that shows us the ability to lead a visual simulation in any environment in order to come in the end for the results we want the robot to accomplish for us. For our work, as it is shown above, we have limited our experiences to control a mobile robot to a DSS Direction Dialog and the Speech Recognition. We have seen and explained as an important task the ability of the mobile robot to avoid obstacles. And all of that has been shown in a visual scene we can easily understand or interpret.

In the end I encourage students to learn and use the Microsoft Visual Programming Language (MVPL) as a graphical programming language. I find it much easier for everyone to make a simple or a complex program and having in the end a visual result what is better for any understanding or interpretation we want to reach after building such a MVPL program.

REFEENCES

[1]$C10. Magnus\ Egerstedt. Control\ of\ Autonomous\ mobile\ robots. Handbook\ of\ Networked\ and\ Embedded\ Control. D. Hristu\ and\ B. Levine, Eds., Birkhauser, Boston, MA,$ 2005. $Pages$ $526-528.$

[2]$Spyros\ G. Tzafistas. Introduction\ to\ Mobile\ Robot\ Control. National\ Technical\ University\ of\ Athens. Athens\ Greece. January$ 2015. $pages$ $36-38.$

[3] $Touzouti\ Nassima.$ 2015. $Commande\ prédictive\ visuelle\ d'un\ bras\ manipulateur. Universite\ Mouloud\ Mammeri\ Tizi-Ouzou. pages$ $27-28.$

[4] $Fabio\ MORBIDI. Localisation\ et\ navigation\ de\ robot. UFR\ des\ sciences. Département\ EEA. M2\ EEAII, parcours\ ViRob. Semestre$ $9,$ 2014|2015. $Pages$ $28\ and\ 47-48.$

[5]$M. Renaud\ BARATE. Apprentissage\ de\ fonctions\ visuelles\ pour\ un\ robot\ mobile\ par\ programmation\ génétique.$ 2008. $Pages$ $35-39.$

[6]$Prof. Alessandro\ De\ Luca. Visual\ servoing. Robotics$ $2. SAPIENZA\ Universita\ Di\ Roma. pages 21-26.$

[7] $Prof. Alessandro\ De\ Luca, Giuseppe\ Oriolo, Marillena\ Vendittelli.$ Control of Wheeled mobile robots: An Experimental Overview. Page 11.

[8]$Matthieu\ Massonneau. CONCEPTION\ D'UN\ PLANIFICATEUR\ DE\ TRAJECTOIRS\ POUR\ UN\ ROBOT\ MOBILE.$ 2003

[9]$Nicholas\ LOUANDEAU. Algorithmique\ de\ la\ planification\ de\ mouvement\ probabiliste\ pour\ un\ robot\ mobile.$ 2010

[10] $Hatem\ ALISMAIL. Exploring\ Visual\ Odometry\ for\ Mobile\ Robots..$ Carnegie Mellon Qatar. $Computer\ Science\ Department.$ 2009. Page 5.

[11] $Guillaume\ Allibert, Estelle\ Courtial, and\ François\ Chaumette.$

Visual Servoing via Nonlinear Predictive Control. Page 8.

[12] *Duran Petiteville Adrien*. Navigation Référencée multi-capteurs d'un robot mobile en environnement encombré. Université Toulouse III. janvier 2012. Page 15-16.

[13]*François Chaumette. Asservissement visuel|Visual Servoing,, and* 56. *INRIA. Rennes − Bretagne Atlantique. Novembre* 2010. *pages* 20 − 23

[14] *Mihai CRENGANIS, Octavian BOLOGA*(2015). *IMPLEMENTING PID CONTROLLER FOR MOBILE PLATFORM. pages* 144 − 146.

[15]*David A. Forsyth and Jon Ponce* (2003). *Computer vision, A Modern Approach. Prentice Hall. ISBN* 0 − 12 − 379777 − 2.

[16]*ROGER PISSARD GIBOLET. Conception et Commande par Asservissement visuel d'un robot mobile.. L'Ecole des Mine de Paris*. 1993. *Page* 54 − 56

[17] *RICHARD B. THOMSON. Global positioning system*: *the mathematics of receivers. VOL.* 71. *NO. . University of Arizona, Tucson AZ* 85721. 4, *OCTOBER* 1998. *pages* 662 − 266

[18] *PIERRE LEBRALY*. Étalonnage de caméras à *champs disjoints et reconstruction* 3*D* − *Application* à *un robot mobile.. UNIVERSIT*É *BLAISE PASCAL – CLERMONTII*, 2012. *Pages* 89

[19] *http*://*localhost*: 50000/*speechrecognizergui*. [*consulted in August* 2017].

[20] *http*://*localhost*: 50000/*texttospeech*. [*consulted in August* 2017].

[21] *Alaa DIB. Commande non linéaire et assservissement visuel de robot autonomes. Automatique/robotique supélec*, 2001. *Français*. < *tel* − 00 65 7022 > .*pages* 85 *and* 121.

[22] *Radu HORAUD, Olivier MONGA. Vision par ordinateur* : *outils fondamentaux. Pages* 140 − 144. *Editions Hermès. Traité des nouvelles technologies, Série informatique*, 978 − 2866014810. < *irina* − 00590049 >. *pp* 426. 1995.

[23] *William Puech. Vision par ordinateur. Université Montpellier II − Nimes* .pages 55-56 and 59.

[24]*Chen Xia. Intelligent Mobile Robot Learning in Autonomous Navigation. Automatic Control Engineering . Ecole centrale de lille* , 2015. *English.* < *NNT* : 2015*ECLI*0026 >. < *tel* − 01298608 >. Pages 31-32.

[25]*Simon Parsons. Introduction to Robotics. Brooklyn College. CIS* 32.5 *Fall* 2009. *page* 16

[26] *Roland Siegwart and Illah R. Nourbakhsh.* 2004. *Introduction to Autonomous Mobile Robots, Intelligent Robotics and Autonomous Agents series. The MIT Press. Massachusetts Institute of Technology Cambrige, Massachussetts* 02142 *ISBN* 0 − 262 − 19502 − *X. Pages* 186 − 188.

[27] *Rickard Nyberg.* 2014. *Development of a mobile robot platform. Master of science in Engineering Technology Mechanical , Engineering. Luel*å *University of Technology. Institutionen f*ör *teknikvetenskap och matimatik. pages* 20 − 23 .

[28] *Florent Malartre. Perception intelligente pour la navigation rapide de robots mobile en environement naturel. Autre. Universit*é *Blaise Pascal − Clermont − Ferrand II*, 2001. *Français.* < *NNT* : 2011*CLF*22132 >. < *tel* − 00673435 >. *page* 21

[29] *Alexandre krupa. Contribution* à *l'asservisssementvisuel échographique* . *Automatique*|*robtique. Universite Renne* 1, 2012. < *tel* − 00830025*v*2 >. *pages* 20 − 21 .

[30]*Geoffrey Blewit. Basics of the GPS Technique*: *Observation Equations. Departement of Geometrics, University of Newcastle. Newcastle upon Tyne, NE*1, 7*RU, United Kingdom.* 1997. *page* 15

[31] *Dr. John F. Raquet. Calculation of GPS PNT Solution. Air Force Institute of Technology. Advanced Navigation Technology* (*ANT*) *Center.* 2012. *page* 55.

[32] *Chris Clark. Autonomous Robot Navigation. COS − Lecture* 7 .2011. *pages* 5 − 7.

[33] *Margarita Chli, Paul Fulgare, Marco Hutter, Roland Siegwart. Autonomous Mobie Robots. ROBOTICS & PERCEPTION GROUP. ASL Autonomous Systems Lab. Swiss Federation Institute of Technology Zurch. pages* 47, *and* 54.

[34] *SABINA MERLO, MICHELE NORGIA and SILVANO DONATI. FIBER GYROSCOE PRINCIPLES. Electrooptics Group. University of Pavia Italy. Pages* 2 − 5.

[35]*J. Gorasia. Fiber Optic Gyroscopes. Instrumentation Spring* 2010. *pages* 46.

[36] Wolfram Burgard, Cyrill Stachniss, Maren Bennewitz, Giorgio Grisetti, Kai Arras. Introduction to Mobile Robotics . Probalistic Sensor Models. University of Freiburg. page 2 − 3.

[37] Hugh Liu, Grantham Pang. Accelerometer for Mobile Robot Positioning. Dept of Electrical and Electronic Engineering. The University of Hong kong. 0 − 7803 − 5589 − X/99/1999 IEEE. page 1735.

[38] *Mohamed Emel Qaziza − Elena Pivarčiová. Mobile robot controlling possibilities of inertial navigation system. International Conference on Manufacturing Engineering and Materials, ICMEM* 2016,6 − 10 *June* 2016, *Nový Smokovec, Slovakia. pages* 4 − 407

[39]*https://> network − cameras has been consulted in september* 24.2017.

[40]: *LMS* 200/211/221/291. *Laser Measurement System. TECHNICAL DESCRIPION. Copyright* © 2006 *SICK AG Waldrick Auto, Reute plant Nimburger Strasse* 1179276 *reute* Germany. www.sick.com

[41] *Ricardo Carona, A. Pedro Aguiar, José Gaspar. CONTROL OF UNICYCLE TYPE ROBOTS Tracking, Path Following and Point Stabilization . Instituto de Sistemas e Rob´otica, Instituto Superior Técnico, Universidade T´ecnica de Lisboa, Portugalricardo. carona@gmail. com, {pedro, jag}@isr. ist. utl. pt. pages* 2 − 6

[42] *Sandeep Kumar Malu & Jharna Majumdar. Kinematics, Localization and Control of Differential Drive Mobile Robot. Nitte Meenakshi Institute of Technology, India. Global Journal of Researches in Engineering: H Robotics & NanoTech Volume* 14 *Issue* 1 *Version* 1.0 *Year* 2014*Type*: *Double Blind Peer Reviewed International Research Journal*

Publisher: *Global Journals Inc.* (*USA*). *Online ISSN*: 2249 − 4596 & *Print ISSN*: 0975 − 5861. *pages* 4 − 6

[43]*Andrew John Davison* (*Keble College*). *Doctor of Philosophy*: *Hilary Term*, 1998. *Mobile Robot Navigation Using Active Vision. Pages* 72 − 73.

[44] *Andersen, Jens Christian*; *Ravn, Ole*; *Andersen* (2007), *Nils Axel. Mobile Robot. N avigation. Technical University of Denmark. pages* 21 − 24

[45] *Stephen Whitaker. Move Out. Adding Go − to − Goal to the Pioneer* 3*DX Explorer Program Florida Institute of Technology. pages* 8 − 16

[46]*Haoyan Li* (2014). *Microsoft Robotics Developer Studio* (*MRDS*). *Florida Institute of Technology. Pages* 9 − 12, 19

[47]*Kevin Hernacki. Little Red Riding Hood. Florida Institute of Technology. Royal Military Academy. pages* 11 − 17

[48]*Zhang Huiping. Microsoft Kinect Based Mobile Robot car. China project center E term*, 2012. *DEPUSH Technologies Ltd.*, *Wuhan, China* .pages 21-22, 33

Glossary

MRDS: *Microsoft Robotics Developer Studio is a framework for developing software to control robots. The MRDS system structure is a web of multiple systems all webbed to getherin and at many different levels* 14

DSS: *Decentralized Software Services is a lightweight .NET − based runtime environment that sits on top of CCR. DSS provides a lightweight, state − oriented service model that combines the notion of representational state transfer (REST) with a system level approach for building high performance, scalable applications* 14

CCR: *Concurrency and Coordination Runtime is a managed code library, a Dynamically Linked Library (DLL), accessible from any language targeting the .NET Common Language Runtime (CLR)* 14

VPL: *Microsoft Visual Programming Language* 14

driven independently 16

LTI: *Linear Time − Invariant* 31

Camera: *designates the physical sensor achieving some frame grabbing* 37

Coordinated homogenous: *coordinates in a projective space* 45

Coordinated Cartesian: *coordinates in the Euclidian space (sometimes named inhomgeneous coordinate)* .....

Parameters intrinsic: *Parameters modeling the process of formation of the internal picture to a camera* 44

Parameters extrinsic: *Parameters modeling the rigid transformation between the main camera and another (camera slave). These parameters are external to a camera. The notion of extrinsic*