OBSTACLE AVOIDANCE WITH ARTIFICIAL POTENTIAL FIELD FOR A CAR-LIKE ROBOT

Federico Rollo
Department of Control Engineering
Sapienza University of Rome
rollo.1851121@studenti.uniroma1.it

Adelina Selimi
Department of Control Engineering
Sapienza University of Rome
selimi.1843827@studenti.uniroma1.it

ABSTRACT

The control of autonomous mobile systems is increasingly catching on in the automation industries. Due to the importance that this systems are obtain in the modern world a great number of companies is financing projects for optimizing cost and performances of such robots. In these years, several solution have been presented for the motion control of these devices both for industry purposes and for research ones. The objective of this paper is to simulate and implement an Autonomous Differential Drive robot which follow automatically an online computed reference in a known site with known fixed obstacles by only receiving as external input the desired position in terms of x and y coordinates.

1 Introduction

Motion planning of mobile robot could seem a simple task to perform for simple systems. But when we want to optimize the time, the cost and the reliability to perform this simple task into a known space where some obstacle are present, the difficulties are no more the same. To perform a movement in a non-empty space there are lots of details to consider for the safety of the environment and the robot itself.

1.1 Problem formulation

The problem presented here is the online motion planning of a Car-like robot (see subsection 1.2 for model presentation) in a known environment with known obstacles. This problem has been treated by means of different control strategies. In particular, in section 2 is presented the Artificial Potential Field technique which have been used to reach the desired position considering the presence of the obstacles and the initial position of the robot. In section 3 is described the localization module used to understand how the car is moving in the environment. Section 4 present the creation of a Simulink model used to simulate the robot before starting the real implementation. Section 5 concerns the implementation of the mobile robot using a micro-controller and in section 6 are presented all the results about the trials and simulations of the robot plus some possible future improvements. To start with, the system model and the tools used are presented in this section.

1.2 From Car-like to Differential Drive model

The real car used in this project is in-between a Car-like and a Differential drive robot. A Car-like robot is characterized by 4 wheels whose two of them has the possibility of steering. While a Differential Drive model refers to a mobile

robot with two actuated wheels.
The robot used has 4 actuated motor as in the Car-like but there is no possibility to turn the wheels to obtain a steering angle and this fact has characterized the choice to use the simplified model of the Differential Drive mobile robot.
This model is strictly related to the unicycle model and so for simplicity and for clarity this model is introduced in advance and then the connection between the two will be expressed.
The unicycle model is characterized by three states: x,y and θ. With x and y are indicated the coordinates of the position and with θ the angle between the robot movement direction and the x-axis with reference to the fixed reference frame.
The inputs of the model are the desired linear and angular velocities, respectively v and ω.
Some characteristics of the unicycle model are:

- it has a non-drifting behaviour, so the drifting actions are not introduced (this may introduce errors);
- it describes the model of the robot with only one wheel and so with only one non-holonomic constraint (there is no problem with differential drive robot because the two wheels share the same non-holonomic constraint);
- this kind of robot is not simple to be constructed because of equilibrium problems (this is why the Differential Drives or other more stable robots are used instead of unicycles).

Despite of this, the unicycle model can be used as starting model for different kinds of robots, as in our case.
The model of the robot is:

$$\dot{x} = cos(\theta)v \tag{1}$$
$$\dot{y} = sin(\theta)v \tag{2}$$
$$\dot{\theta} = w \tag{3}$$

What we really have as input are the velocities of the right and left wheels, ω_r and ω_l and not the linear and angular velocities but the angular wheel velocities can be converted into the car ones using:

$$v = r\,\frac{\omega_r + \omega_l}{2} \tag{4}$$
$$w = r\,\frac{\omega_r - \omega_l}{2\,d} \tag{5}$$
$$\tag{6}$$

Where r is the radius of the wheels and d is the distance between the center of the wheels.
This model can be applied to our mobile robot and can be used both for the simulation and implementation of the project. To conclude with, the wheel velocities read by the encoders have been used as model outputs. Due to the model structure utilized, the four wheels sensed velocities have been merged into two real velocities (left and right) by doing a mean of the front and rear wheels both for the right and left side. As a result, two measurement have been used which are the real angular velocities of the Differential Drive model wheels.

1.3 Tools

This subsection has been introduced to exhibit the software and hardware used in the project. Starting with the software, the main programs used are Matlab & Simulink for the computations of the Artificial Potential Fields, the Localization and the simulations of the robot while the Arduino IDE has been used to upload the micro controller.
Regarding the hardware, apart for the PCs obviously, have been used:

- An Arduino Uno card as micro controller.
- An Ada-fruit Motor Shield to control the motors directly through a velocity reference without considering input voltages and PWM.
- A five pack batteries to obtain a 6v voltage to supply the motors.

- Four motors each of which has an incremental encoder and a wheel to be connected with.
- An HC-06 Arduino Bluetooth module for communications.
- Various other utilities such as connection wires, two resistors, an input port to charge the batteries and a button to turn on/off the car.
- A case to contain all the objects specified above.

This project can be suited for every different kind of mobile robot with some slight changes regarding the model states, inputs and outputs implementations into the system.

2 Motion planning

The aim of this project is to make the car able to decide online the path to follow, having some information about the workspace. In what follows it will be considered that the space in which the car has to move is known, so also the obstacles present in the environment are known.
The method which will be described and used is the so called *artificial potential field*, which is suitable for online motion planning in the case of known workspace.

2.1 Artificial Potential Field

This method is based on the concept of potential field. The main idea is to create an attractive field which makes the car go to the desired position, and multiple repulsive potential fields corresponding to the obstacles, used to make the car avoid them. So the total potential is the sum of attractive and repulsive fields, and the negative gradient of that potential indicates the direction of motion.
For the attractive potential filed a combination of two fields is used: paraboloidal field and conical. The paraboloid is good to be used in proximity of the desired final position. So we have that the attractive potential is given by:

$$U_a(q) = \frac{1}{2} k_a ||e(q)||^2 \qquad \text{if} \quad ||e(q)|| \leq \rho \tag{7}$$

$$U_a(q) = k_b ||e(q)|| \qquad \text{if} \quad ||e(q)|| > \rho \tag{8}$$

Where $e(q) = q_g - q$ is the position error, the difference between the current configuration and the desired configuration (q_g). To achieve continuity in the forces transition it is required that $k_b = \rho k_a$.
For the obstacles instead, as we said, a repulsive potential is required in order to keep the car far from the obstacles. For each obstacle a convex potential field is created:

$$U_{r,i}(q) = \frac{k_{r,i}}{\gamma} \left(\frac{1}{\eta_i(q)} - \frac{1}{\eta_{0,i}} \right)^{\gamma} \qquad \text{if} \quad \eta_i(q) \leq \eta_{0,i} \tag{9}$$

$$U_{r,i}(q) = 0 \qquad \text{if} \quad \eta_i(q) > \eta_{0,i} \tag{10}$$

Where η_i is the clearance, the minimum distance of the car from the obstacle. And $\eta_{0,i}$ indicates the range of influence, indeed for a clearance bigger than it the potential is equal zero. It was chosen in this case $\gamma = 2$.
By the superposition principle, we have that the total force field is given by:

$$f_t(q) = -\nabla U_t(q) = -\nabla (U_a(q) + U_r(q)) = f_a(q) + \Sigma_i f_{r,i}(q) \tag{11}$$

To avoid local minima, vortex forces [1] were used as repulsive forces. This means that the repulsive forces are given by:

$$f_r = \begin{pmatrix} \frac{\partial U_{r,i}}{\partial y} \\ -\frac{\partial U_{r,i}}{\partial x} \end{pmatrix} \tag{12}$$

Moreover the obstacles that are going to be considered are cylinders, so that the clearance can be calculated by:

$$\eta_i = \begin{bmatrix} x_{0,i} \\ y_{0,i} \end{bmatrix} + \frac{q - \begin{bmatrix} x_{0,i} \\ y_{0,i} \end{bmatrix}}{||q - \begin{bmatrix} x_{0,i} \\ y_{0,i} \end{bmatrix}||} \cdot r_{obst,i} \tag{13}$$

Where $x_{0,i}, y_{0,i}$ is the center of the i-th obstacle and $r_{obst,i}$ its radius.

2.2 Control law

From these forces different techniques can be used to control the car. In the following the forces are used as generalized velocities. So the control law is:

$$\dot{q} = f_t(q) \tag{14}$$

This control law guarantees asymptotic stability of q_g in absence of local minima. Applying this control law to the kinematic model of a unicycle, the least-squares solution is given by:

$$v = f_{t,x} cos\theta + f_{t,y} sin\theta \tag{15}$$

$$\omega = f_{t,\theta} \tag{16}$$

That for the differential drive is:

$$\omega_R = \frac{v}{r} + \frac{d}{2r}\omega \tag{17}$$

$$\omega_L = \frac{v}{r} - \frac{d}{2r}\omega \tag{18}$$

Where r is the radius of the wheels and d is the distance between the axes of the two wheels.
Assuming that the orientation is not relevant we can define the force $f_{t,theta}$ in such a way that the car will be aligned with the total field:

$$\omega = f_{t,theta} = k_\theta(atan2(f_{t,y}, f_{t,x}) - \theta) \tag{19}$$

3 Localization

The localization module is essential for the state reconstruction of the system and essentially that's the block which compose the feedback path. This block is composed by an odometric localization algorithm which reconstruct the state of the robot by integrating the robot model using the mean wheels velocities within the sampling time interval. In practice, the motion of the car is computed by using the ideal model of the Differential Drive and the velocities of the wheels. These latter are available in the real system through incremental encoders. This is done in order to predict the position of the car after some time ΔT (in our case that's the sampling time T_s).
Instead of the angular velocities (angle derivatives within time $\frac{\delta\theta}{\delta t}$) the mean velocities (difference of angle over a sampling time interval $\frac{\Delta\theta}{\Delta T}$) have been used by considering the pulses of the encoders within a sampling interval. These pulses are strictly related to the angle of the wheel shaft and the mean velocities are computed by simply analyzing the pulses obtained by the incremental encoder within the sampling interval and applying the following formula $\Delta\Phi = \frac{Pulses\ T_s}{1920} 2\pi$.

3.1 Runge-Kutta integration

The odometric localization integration module used is the 2^{nd} order Runge-Kutta one. This module takes as input the differential angle $\Delta\Phi$ within the sampling time interval an use it to integrate the robot model. Due to the digital

behavior of our system (Arduino & Matlab) also the localization module is supposed to work in the discrete time. In fact, the integration module utilized is the following:

$$x_{k+1} = x_k + v_k\, T_s\, cos(\theta_k + \frac{\omega_k\, T_s}{2}) \tag{20}$$

$$y_{k+1} = y_k + v_k\, T_s\, sin(\theta_k + \frac{\omega_k\, T_s}{2}) \tag{21}$$

$$\theta_{k+1} = \theta_k + \omega_k\, T_s \tag{22}$$

Instead of the velocities, two other variables are used, this is due to the fact that we have encoders and not tachometers. This two variables are Δs which is the traveled length and $\Delta\theta$ which is the total orientation change. Once this two variable have been introduced can be seen that they are related to the velocities by the simple equations.

$$v_k\, T_s = \Delta s \tag{23}$$

$$\omega_k\, T_s = \Delta\theta \tag{24}$$

Also for a Differential Drive robot the velocities can be completely replaced by the traveled length Δs and the total angle displacement $\Delta\theta$ which can be computed by using the differential angles founded above with the encoder pulses. The equations are the following:

$$\Delta s = \frac{r}{2}(\Delta\Phi_R + \Delta\Phi_L) \tag{25}$$

$$\Delta\theta = \frac{r}{d}(\Delta\Phi_R - \Delta\Phi_L) \tag{26}$$

where *r* is the radius of the wheel, *d* is the distance between the two wheels and *R* and *L* are the differential angle indices which correspond respectively to the right and left wheel.

4 Simulation

A simulation of the overall system has been implemented using Simulink software. Through the simulation a tuning of the gains of the potential fields has been obtained. For the model of the car a differential drive has been used, as already mentioned in paragraph 1. The position of the robot is calculated by the Runge-Kutta localization.

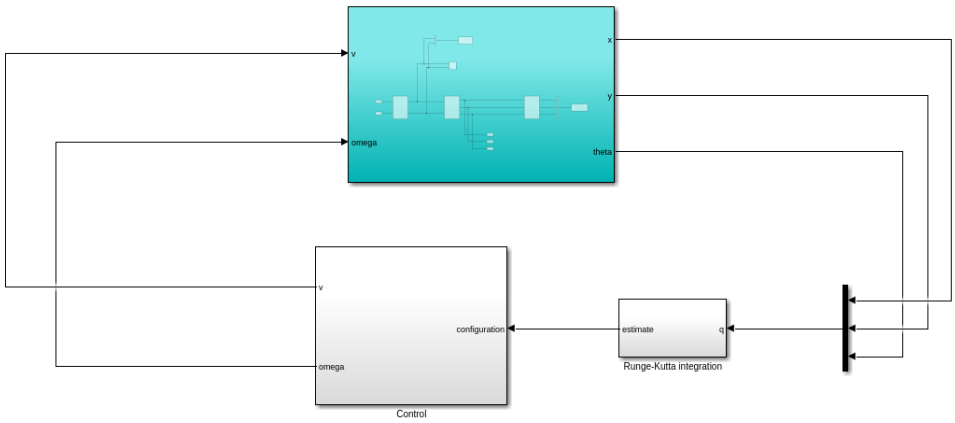
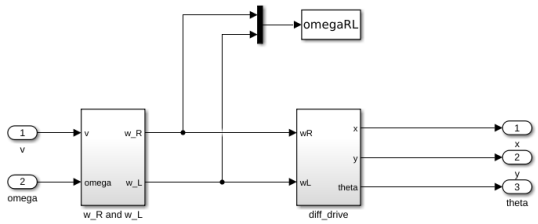
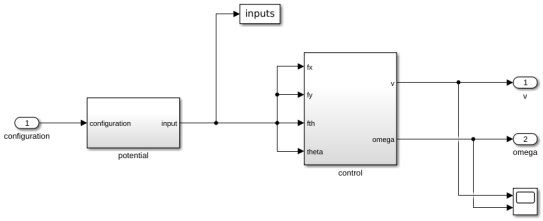


Figure 1: Simulation of the system

The controller calculates the inputs depending on the configuration of the car, such configuration is obtained from the model of the differential drive. The inputs are v and ω that inside the model are used to calculate the real inputs for the car that are ω_R and ω_L. These subsystems are shown below:



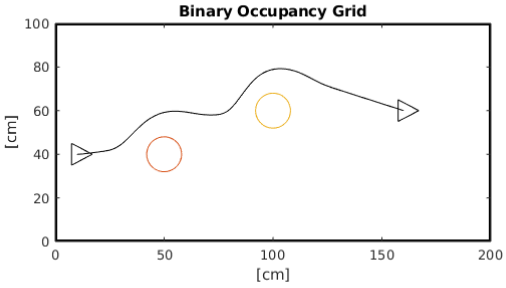
(a) Differential drive model



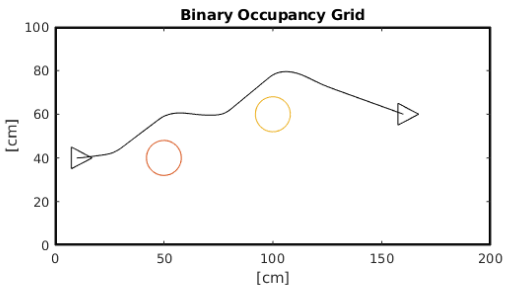
(b) Controller, artificial potential fields

4.1 Simulation results

In the simulation a motion of the car has been simulated in a workspace with some obstacles, in what follows two obstacles are considered (similar behaviour for more or less obstacles). The first gain to be chosen is the one of the attractive force. Simulating the motion, the gain has been chosen such that the angular velocities necessary to reach the final position are not too large. In particular, the real car has a maximum velocity of about 6 rad/sec so it is better to do not exceed this value. With a gain $k_a = 10$, in the general case without obstacles, the velocity required has a maximum of about 3 rad/sec. Now considering the presence of obstacles, the gain of the repulsive forces has to be chosen. In the formula of the repulsive forces, the gain k_r is multiplied by values which are of the order of 10^{-4} (because of the presence of the square of the inverse of the clearance) so an high value is needed for k_r in order to make the repulsive force of the same order of the attractive one. The value $k_r = 60000$ has been chosen. The last gain to be chosen is the gain k_θ, with a very low value for this gain the car would go around the first obstacle without reaching the final position. That is why a gain close to k_a has been chosen (lower than it to avoid too much curvature in the path) and it is $k_\theta = 5$.



(a) path without considering limitation of the real car



(b) path considering limitation of the real car

In the figure above the path planned is shown, the car is represented by a triangle and the obstacles are circles. In the first figure the limitation of the real car maximum velocity has not be considered. In the second figure it is considered by adding a saturation on the input angular velocity of 6 and -6 rad/sec. The behaviour is still good and close to the first one because the velocity are not exceeding too much the maximum value of the velocity. A different approach could be used, like lowering the gains or making the controller react to the saturation and try to do not go in saturation. This has not been done because the behaviour remains pretty much the same, but it can be an improvement to be done for future applications.

5 Implementation

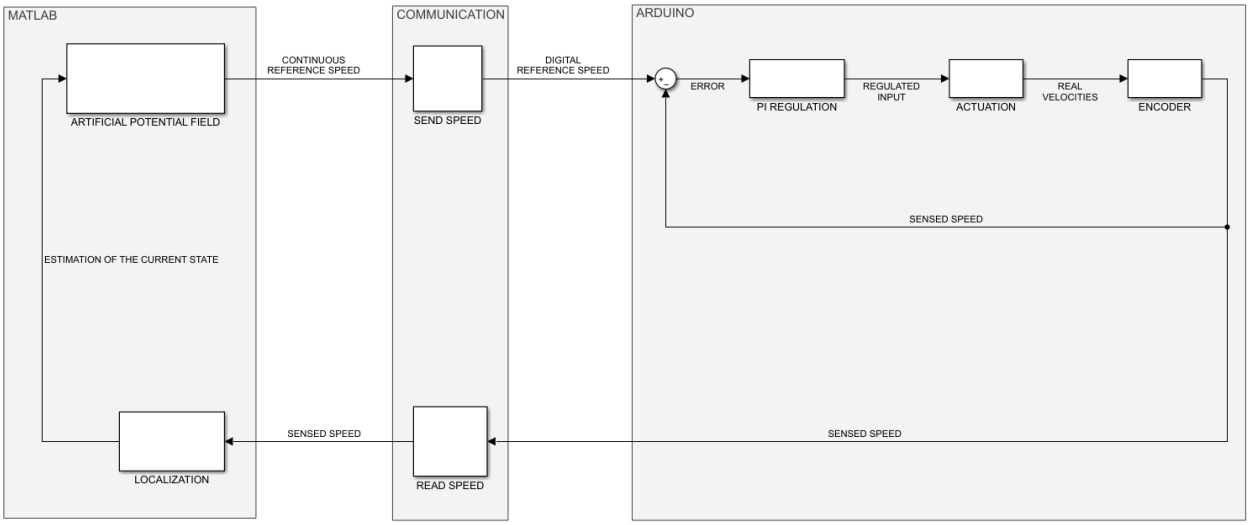


Figure 4: Implementation scheme

The whole implementation system is characterized by different blocks. As can be seen in the figure above,the main subsystem are three:

- Matlab subsystem
- Communication subsystem
- Arduino subsystem

The first subsystem is composed by two blocks:

- Artificial potential field, needed for the trajectory computation
- Localization, used to obtain the current position and orientation of the robot using the 2^{nd} order Runge-Kutta model integration

Apart from the signal conditioning used for the communication with Arduino, the Matlab block works as in the simulation. Indeed, trajectory creation and localization are computed in the same way both for the robot model and the real car robot.
The second subsystem, the communication one, is composed by a single module which perform two different tasks:

- send the reference speed of the wheels to the car
- read the real speed of the wheels form the encoders and send them to Matlab

As explained in next *Bluetooth Communication* subsection in reality an approximation of the velocities has been used and not the real ones.
Finally,into the Arduino Subsystem there are three important function blocks:

- PI regulation of the four wheels speed (one regulator for each wheel)
- Actuation of the regulated reference computed by the PI

- Collection of the approximated velocity from the encoder

The Matlab block characteristics have already been treated in the previous sections 2 and 3 so in next subsections there will be the only explanation of the Communication and Arduino subsystems.

5.1 Bluetooth communication

Due to the serial Bluetooth communication between Arduino and the PC opened in the initialization, the data can be written and read at a very high speed (communication rate: 115200 BAUD). The Write task aim to send the desired references, the right and left wheel encoder pulses, into Arduino subsystem. These pulses are strictly related to the velocity of the wheels; they are proportional to the angular position changes of the wheel within the sampling time and this values are a good approximation of the real velocity. After having written these data to Arduino, Matlab subsystem will wait for a response of the micro-controller that once has collected the pulses of the wheels with the encoders will send always on the Serial Bluetooth channel those values. Once Matlab has received the new values it performs it's control loop composed by localization and artificial potential field and so on.
An important aspect of the communication block is also the synchronization between Matlab and Arduino. Due to the fact that Arduino works autonomously once have received a reference, because it maintains that level until a new one is received (works similarly to a ZOH), while Matlab wait each time after its computations, we need that the Matlab loop has to be faster then the Arduino internal loop. For this reason, Arduino has a double sampling time (explained in *Arduino code* subsection) which each 0.4 seconds will send the velocities to Matlab.

5.2 PI tuning

Let's now see how the PID has been tuned. The chosen regulator is composed by only the proportional and integrative action so there was the need to tune only Kp and Ki terms. First of all, the behaviour of the process has been approximated as a first order system by using some step inputs of different amplitudes and collecting the encoder data about the motors shaft rotations. The different trials made with different step amplitudes had allowed to find a mean of the process behavior and this has been used to tune the regulator. Due to the encoder digital operations the processes founded was simply composed by samples.Thanks to a simple interpolation point-to-point, we were able to find the time constant τ at the 63% of each sensed process. This is an approximate known technique which allows to find the time constant of a first order process and the previous assumption consider the behavior of the motor transfer function just like that. All the interpolated sensed processes had a similar time constant, $\tau = 0.32$ while the gain of the process was almost proportional to the input one. Finally, the process used is:

$$P(s) = \frac{1.2}{1 + 0.32s} \tag{27}$$

The characteristics of this process are:

- settling time = 1.25 seconds
- rise time = 0.703 seconds
- peak = 0%

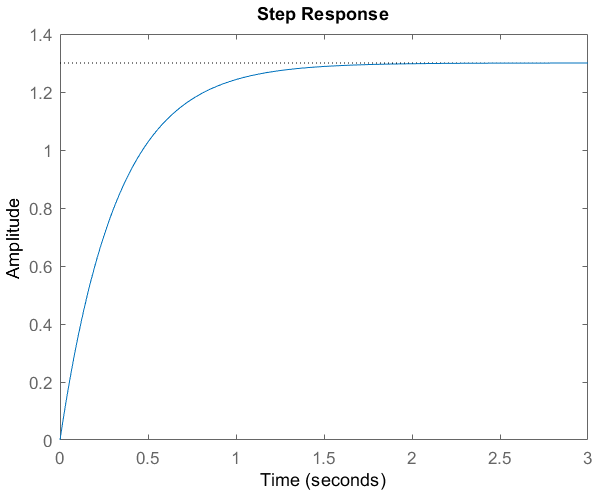


Figure 5: step response of the process transfer function in (27)

Once the mean Process has been found we were able to tune the PI regulator by simply using the Matlab PID tuner which allows to observe how the closed -loop behavior changes using different Kp and Ki terms. The tuning has been performed as to be faster enough to satisfy the sampling time requirements but also robust to variations in order to avoid oscillations and peaks in the response. According to the software, the chosen parameter values were $Kp = 0.6$ and $Ki = 5$ and so the regulator was:

$$PI(s) = 0.6 + \frac{5}{s} \tag{28}$$

Once these values had been used for the first time, it was clear that this kind of approach for the PI tuning contained uncertainties and not modeled dynamics for the motor models which lead to a different behavior of the robot, introducing a significant overshoot and considerably reducing the closed-loop system settling time. There was some improvements but due to the approximated process used in Matlab simulations the expected performances was not achieved. By using an empirical approach some others PI terms have been tested both on Matlab and directly on the car to find a good compromise between settling time and control effort. After some trials, the found PI terms have been set to $Kp = 1$ and $Ki = 4$ and the regulator transfer function is:

$$PI(s) = 1 + \frac{4}{s} \tag{29}$$

Now that the process is know and the PID has been tuned the last thing to do is to see how really are the performance of our system. The final feedback transfer function is:

$$\frac{PI(s) * P(s)}{1 + PI(s) * P(s)} = \frac{1.3s + 5.2}{0.32s^2 + 2.3s + 5.2} \tag{30}$$

The characteristics of this closed-loop transfer functions are:

- settling time = 0.623 seconds
- rise time = 0.421 seconds
- peak = 1.38%

These values are perfectly fitted into our implementation scheme because they assure that the robot can reach the true speed in a fast time without slowing down the synchronization of the whole scheme and without introducing any peak. From the step response of the real process can be seen that the motor behave faster then the simulated one because as

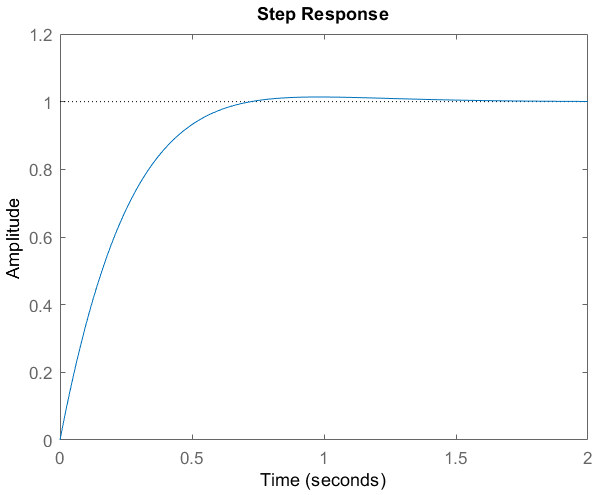
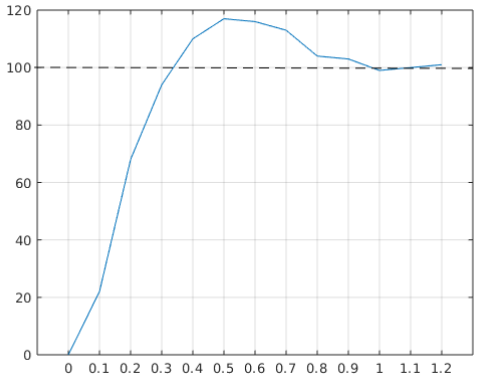
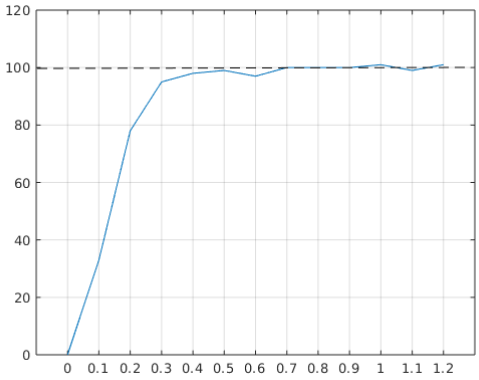


Figure 6: Feedback transfer function in (30)

previously said the model contains some approximations. In the following pictures can be seen how the real motors behaves using the different PI terms found by simulation ($Kp = 0.6$ and $Ki = 5$) and the once found by trials ($Kp = 1$ and $Ki = 4$)



(a) Behavior of the motor using $Kp = 0.6$ and $Ki = 5$



(b) Behavior of the motor using $Kp = 1$ and $Ki = 4$

5.3 Arduino code

The Arduino code is a simple liner code which,thanks to the some delays and synchronizations, works as three parallel programs. This parallelism is needed because the sampling time of the PI regulator and the encoder are faster than the one of the feedback loop. In Practice, at each computation Arduino do the following operations:

- count the pulses from the encoder and add them into a counter used for the feedback loop
- compute and use the output value of the PI controller using as input the error between the pulses reference (held with a ZOH) and the real pulses (gained with the encoder)
- write the encoder pulses obtained from the previous sampling each 0.5 seconds

To be more clear, the temporal sequence of a single iteration is: read the references if available from the Bluetooth, use the new reference or the older one as one input of the PI, read and store the encoder values, compute the PI output with

the error between reference and sensed values and apply it, update the feedback pulses counter and zero the encoder values, finally, each 0.5 seconds write the feedback pulses counter to the Bluetooth.
One important thing to notice is that the robot has four wheels and consequently four encoders but the input and output of the encoders are always two because of the model considered in this project is a Differential Drive mobile robot which works with only two wheels. For these reason, the two output are the left and right pulses but each output has been averaged over the front and rear wheel pulses for each side.

6 Conclusion

At the end some tests on the real car has been done. The path followed by the car without obstacles and without making curves is correct, also the localization works correctly. There are errors arising in the localization when the car has to change orientation and make a curve, because the Runge-Kutta localization accumulates error.
For example if the initial position of the car is in (0;0) aligned with the x axis ($\theta = 0$) and the final position is (0;y) where y is different than zero, the real final position of the car has a significant value of x different than 0. Instead the final position calculated by the Runge-Kutta localization of Matlab has an x close to 0. This is because of accumulation of errors in the calculation of the the angle θ, which is made worse also because of the presence of drifting events.
In the case of a workspace with obstacles the result of the tests are still affected by localization errors. This implies that in the case of not too long paths and with not too much change in orientation the car is moving in an acceptable way otherwise the error in the localization makes the car not able to avoid obstacles and reach final position.

6.1 Future improvements

To conclude with the scheme of control proposed seems to work fine for the problem of obstacles avoidance, the *artificial potential fields* are a good tool for motion planning. To make this project applicable for real system a better localization system is necessary. Moreover for an Arduino project an Arduino Uno micro-controller board has some limitations. The interrupt inputs available for this board are two, one of which is used by the motor shield so just one could be used. To use the four encoders, four different interrupts are necessary. Other digital and analog inputs were used as interrupts, making the code heavier and also limiting the performance of the encoders. An Arduino Mega board would overcome these limitations. Also it would allow the use of an additional sensor as an ultrasonic sensor, useful and necessary for unknown obstacles.

References

[1] Oriolo G. De Medio C., Nicolò F. *Robot Motion Planning Using Vortex Fields. In: New Trends in Systems Theory. Progress in Systems and Control Theory, vol 7.* 1991 edition.