POLITECNICO DI TORINO

Corso di Laurea Magistrale
in Ingegneria Meccanica

Tesi di Laurea Magistrale

Energy efficient motion planning for the ROPOD robot

Relatore:
prof. S.P. Pastorelli

Candidato:
A. Macheda

A.A.2017-2018

Contents

Acknowledgments

During my work on this thesis I spent six months at the Technische Universiteit Eindhoven, where I had the chance to meet and work with several persons who helped me throughout this experience. Among them I'd like to thank here a few. First of all I'd like to thank prof. dr. Bruyninckx, for his time, the precious feedback he gave me and for urging me to seek for creative ways to solve the problem I was working on. Second, my deepest gratitude goes to dr. Lopez Martinez, for his constant support and for all the discussions we had. Lastly I sincerely thank W. Houtman for his help with the setup of the experimental part.

Chapter 1

Introduction

In this chapter the objectives of this thesis are stated and the constraints to meet are listed. Besides the proposed approach is briefly presented and the organization of this work is outlined.

1.1 Objectives

Mobile robots can be used to execute a number of different tasks, that range from transporting loads in both industrial and non-industrial facilities, to search and rescue operations. In order to carry out their job, a control architecture consisting of several layers is typically used, each of these layers dealing with a particular aspect of the whole problem, in a way that the outputs of a module are the inputs of the next one. In Fig. 1.1 it's possible to see a classical cascade organization. The higher level planning is made by a task planner that will decide, given a certain task, what actions will be performed. In the case of a mobile robot most actions will need the robot to move from a certain location to a target and the motion planner will deal with this aspect giving a trajectory as output that the motion controller will track. To solve the motion planning problem, whose goal is to generate an obstacle-free trajectory for the robot to follow, over the years several approaches have been developed [1]. Since the requirements differ from one application to another, so do the characteristics of the planners: some looks for optimal paths [3], some are meant to run online to redo the planning as the surrounding environment changes [6], other explores randomly the space around the robot [4] in order to find a feasible path from which the trajectory is generated.

In the context of Horizon 2020, a European research program, the ROPOD project is under development from a consortium of universities and private societies, aiming to develop a smarter and more cost-effective robot for logistic tasks. In this scenario longer operative time without battery charge is highly desirable and so energy efficiency of the robot is a key aspect. The purpose of this thesis is the development of an motion planner capable of driving the ROPOD robot in an energy efficient way, which will pave the way to the previous mentioned target. The developed motion planner has to deal with several constraints:

- It has to be able to avoid collisions with the surrounding obstacles, both fixed and moving ones.
- It has to be able to run online, so that it will be possible to redo the planning in case unexpected events occur.

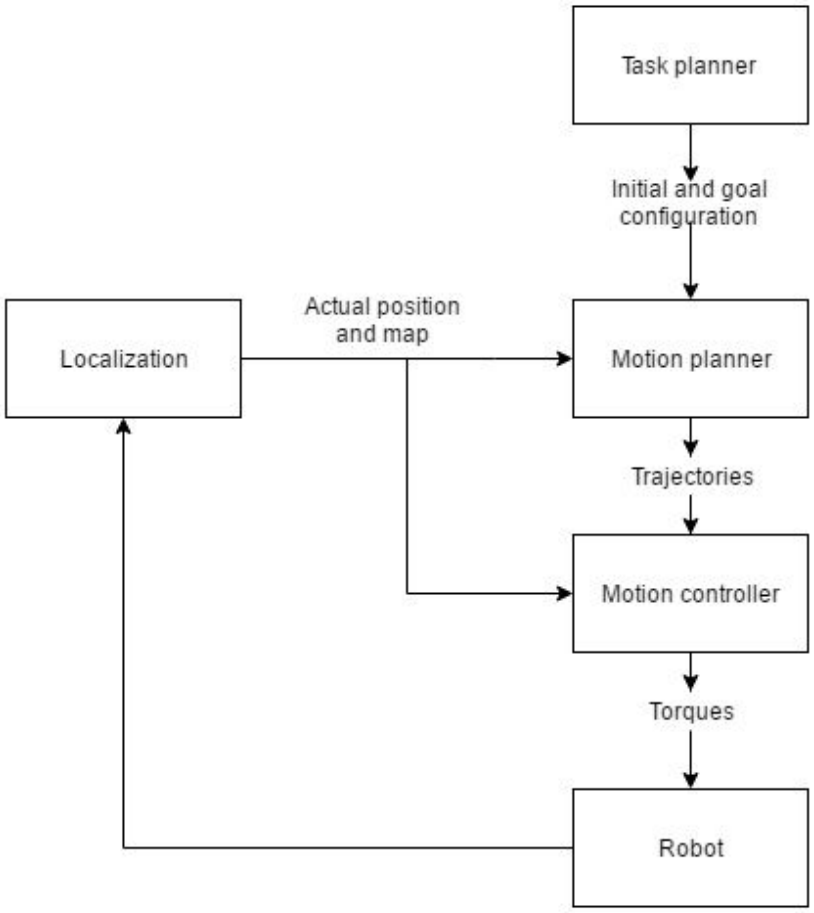


Figure 1.1: Interaction of motion planner with other components in terms of input and outputs

- It should be able to consider the kinematic constraints of the specific robotat hand, so that the generated trajectories are feasible.

1.2 Approach

The proposed approach is the use of model predictive control with a term in the cost function that summarizes the power drained from the battery of the electric-mechanical parts of the robot to achieve energy efficiency while going from a start to a goal position. Given several limitations, that will discussed in details in the next chapters, this method turned out to be prone to fall in local minima while reaching its target, thus several suggestions given for navigating complex environments, in order to ensure that the target is reached.

1.3 Organization

This thesis is organized as here outlined. In chapter 2 the ROPOD project is presented along with an overview of motion planning for mobile robots and model predictive control. In chapter 3 the 1D problem for a simple robot is analyzed, suitable models are derived, and several formulations of the problem are tried and compared. In chapter 4 the complete problem applied to the ROPOD robot is considered, its kinematic and dynamic models of the robot are derived and several simulations are carried out. After the numerical part several tests on a robotic platform were made and both the results and the robot used are presented in chapter 5. Finally chapter 6 presents conclusions and suggestions for future work and possible improvements.

Chapter 2

Background

In this chapter a brief description of the ROPOD project this thesis is contributing to is presented, alongside a review of path planning for mobile robots and an explanation of the model-predictive control.

2.1 The ROPOD project

Over the last decades robots have widespread in our society, dealing with tasks that range from search and rescue operations to space exploration. In some sectors they have proven to be very effective, like industrial manipulators that have become standard components of assembly lines, especially in the car manufacturing process. In other sectors instead, there are possibilities still to be fully exploited due to challenges that are still to overcome.

A particularly interesting field of application is the use of mobile robots in everyday life. Two examples could be vacuum cleaning robots and self driving cars. The first ones are already well-established and are present on the market with several models while the second ones are still under development. Another use of autonomatic guided vehicles(AGVs) is the transportation of goods in scenarios like factories or hospitals. Although AGVs for this purpose have been developed and are currently available, they present several shortcomings that prevent their presence in non-industrial scenarios and limit their usage. One of them is that most AGVs are guided using cables or magnets inserted in the floor, thus requiring a modification of their working environment and that they lack of flexibility, since the can only follow the wired paths. Another important limit is related to the presence, in non manufacturing facilities, of a variety of loads to deal with. In fact current available solutions are not able to carry many of the different shapes and weights of loads that are present in real world scenarios, like hospitals, but are designed to use a specific interface. Thus they can be used only if the load is a standard one.

In order to overcome these problems, in the context of Horizon 2020, a European research program, the ROPOD concept for a smarter kind of AGV has been presented, and it will developed over the next 3 years by a consortium of universities and private societies. In order to be a true step ahead to the use of AGVs in real world scenarios, there are several requirements that this robot has to meet:

- The robot must be able to deal with the great diversity in loads shape, size and weigh, thus being useful in many different situations.
- It has to be way cheaper than available solutions. In order to achieve this mass-market products have to be used which adds constraints in the mechanical design stage.

(a) *iRobot Roomba[25]*

(b) *Google self driving car[26].*

- It has to be able to properly navigate a complex and crowded environment therefore kinematic structures presenting non holonomic constraints, such as Ackermann vehicle for example, may be inappropriate.
- It should not require major adjustment to the location of its deployment.
- Long time autonomous work without battery charge or change is desirable.
- The ROPOD should be able to work autonomously as well as along with workers, e.g. acting like a force amplifier guided by persons in carrying heavy loads.
- Several robots should be able to work together in transportation of loads that are particularly heavy.

The key idea to achieve the above results is to create an ultra-flat robot. In fact such a robot will be able to go under his load and then attach itself to it and then carry it around. This idea is already present on the market as it can be seen in 2.1 but what still misses this model is the capability to adapt itself to the load to carry by collaborating with other robots, instead different sizes are available and have to be chosen in advance.

From a mechanical design point of view in order to achieve omni-directional motion the standard solution would be the use of mechanum wheels but since they are expensive and not reliable under stress another solution has been proposed. In 2.2 a scheme of twin steerable wheels can be seen. In this configuration both wheels have their own motor while the vertical axis is free to rotate and it is equivalent to a normal steerable wheel with one motor on the wheel axis and another on the steering one, but with a flatter design, since no motor on the steering axle is required. To reach holonomic motion a free to rotate offset between the vertical axis and its junction to the robot chassis has to be introduced.

Another important aspect of the ROPOD project is energy efficiency. In fact, since its working environment is indoor, nor solar panels nor thermal engines can be used, the amount of energy available is very limited and at the same time long operative time is highly desirable. This can be secured in many different ways, one important and obvious is trying to reduce friction during the mechanical design phase, choosing proper solutions for the power transmission. Also

Figure 2.1: An example of flat AGV: SIBA AKF Q-line [27]

energy drained by sensor can be a relevant component of the total energy usage. A good way of lowering it could be, for example, limiting the use of sensors that requires a lot of energy, like cameras, and relaying on odometry for estimation of position as much time as possible. A third way to improve the energy efficiency of the robot, while it is moving in its environment, is to take into account its power consumption while solving the motion planning problem and the motion control problem. In fact in the motion planning problem, where the goal is to compute a trajectory that goes from a starting position to a goal position the robot can chose several possibilities, that have to be obstacle-free, and obviously the energy consumed differs for every possible choice.

2.2 Motion planning for mobile robots

Motion planning is the process that finds a path for the robot that leads it to its destination from its current position, usually dealing with additional constraints, for example avoiding obstacles. Since navigating the environment is one of the most fundamental abilities of a mobile robot, path planning and trajectory planning have been extensively studied. As a result there is at present time a huge amount of path and trajectory planners available, e.g. [1] and [2], and the development is still ongoing. Of course each of them has his own pros and cons and it is suitable for certain applications and not for some others. Before classifying path planners however, it is important to point out that every algorithm has to relay on a representation of the world that surrounds the robot, namely a map. There are several type of maps, the most important being:

- Grid based maps
- Line maps
- Topological maps

Grid based maps represents the world using a matrix, so that a square in the real world corresponds to an element of the matrix, the dimension of the square depending on the resolution of the map. The values of the elements of the matrix tells whether that particular part of the space is free or not. Usually binary maps are used[1], so that the matrix values are either 0 or 1,

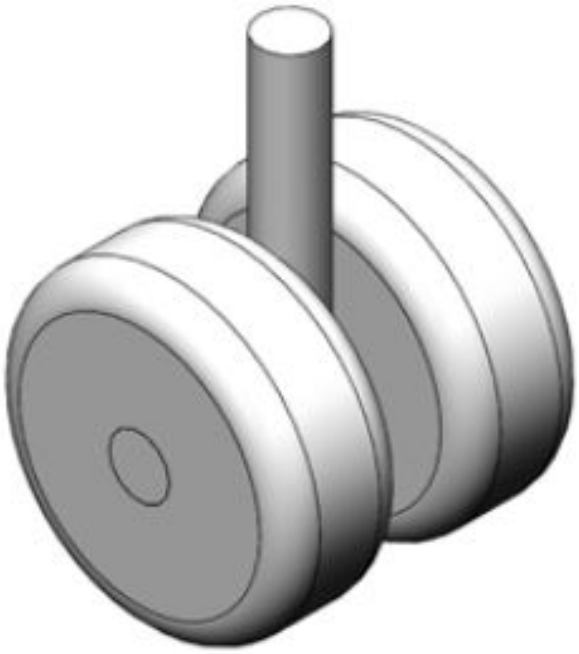


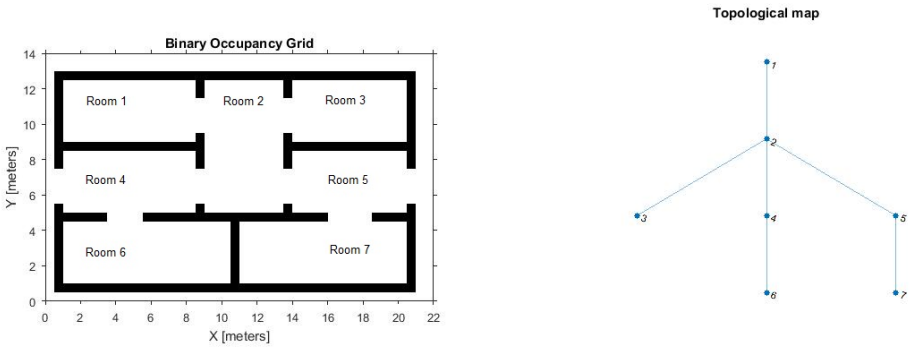
Figure 2.2: A scheme of twin steerable wheels

meaning completely free or occupied, but probabilistic grid are also used where the values range from 0 to 1, representing the probability that a square is occupied or not. Grid based maps are widely used because of their simplicity, however they present a major disadvantage, in the form of a trade-off between memory needed o store the map and precision of the map itself. In fact the higher the resolution of the grid, and so the smaller the square that each element represents, the better it will be the representation of real shapes, and so discretization errors will become less important but the dimension of the matrix will increase significantly. This problem arises especially when a large environment has to be represented with a good resolution.

Line maps are a popular alternative [1] to grid based maps and they consists of a list of lines that represents the boundary of the obstacles in the environment. They are obtained starting from the data collected by range sensor, usually a set of points, which are used to compute the best fitting lines. However, since the number of lines that have to be used to fit the data reasonably well is not known in advance, the problem of building the map from the sensor data is not trivial. On the other hand, they scale better than grid based maps for large environments, meaning that they require less memory.

Topological maps represents the space using a graph[1], where for example each node is a location and the presence of an edge between two nodes means that that is possible to go from one to the other.

Starting from one of the above representation there are many ways to solve the path planning problem. Here the most used classification for different approaches is presented along with some examples.



(a) (b) .

Figure 2.3: A grid based map and a topological representation of the same environment

2.2.1 Global planners

Global planners are algorithms that solve the path planning problem directly, so they are given the map, the starting point and the objective, and they compute the whole path. They are generally computationally much more expensive than local planners, and that's the reason why they are usually applied for off-line computations. Also they need to know in advance how the environment will change in case moving obstacles are present. On the other hand, considering the whole problem at once, they are able, for example, of reaching optimal solutions, for example in terms of time used to reached the goal or energy consumed. Another important point in their favor is that a good global planner is a complete algorithm, meaning that if a feasible path exists than the algorithm will find it.

A first important example of this kind of planner is the A^* algorithm [3]. Since this algorithm requires a topological map the first step is to compute it in case a geometrical map, line or grid, is given. For a grid map for example, every free cell can be a node, that is linked to any other free cell that shares a border with it, while for a line map several techniques are available, like cell decomposition [8]. This algorithm then explores a weighted graph, where the cost can be for example the distance between two nodes, creating a tree that starts from the starting node. At each iteration, for every node that can be reached from the current trees the cost function is computed as the sum of a first term that is the cost to reach that node, by summing the costs along the branch that leads to the node, plus one term that estimates the cost from that node to the goal. The node with minimum cost is added to the tree. It can also be shown that this algorithm is complete in the sense that if a path exists it will find it.

Another important example of global planner is the RRT^* algorithm [4]. This method, instead of computing a graph that describes the connectivity of the free space of the environment, explores randomly the space building a tree that links the starting position and the goal. At each iteration this algorithm selects a random point in the space explore and finds the node of the tree that is the closest to it. Then, if the distance between the node and the random point is over a certain parameter, a new node is created in the direction of the random point, at a fixed distance from the node belonging to the tree and if this new node belongs to the free space is added to the tree. Other variations of this algorithm exists that start from both the goal and the starting position and build two trees that will eventually merge.

This family of global planners that rely on random exploration differs from the graph-

exploring one in many key aspects. First, they are only probabilistic complete, in the sense that in a path exists they will eventually find it but there is no guarantee that it will happen in a finite time. Second, they are not optimal at all, usually the resulting path is non-smooth and a pruning phase is needed, where unnecessary nodes are got ridden of, and also a smoothing phase is required, performed by adjusting the position of the nodes. Nevertheless this class of planners is currently used in many applications as it is faster than the graph exploring methods and is very general purpose.

2.2.2 Local planners

Local planners are algorithms that guide the robot computing at each iteration only the present command to give to the robot instead of the whole path to the target. For this reason they are very fast at computing and that's the motive for their application in many real applications, where re-planning is needed to be done online. Besides, they are able to deal with moving, unpredictable obstacles that surround the robot and they do not need a map of the whole environment in advance. The price to pay for these advantages is that only a local horizon is considered, so it's possible that they get stuck at certain locations, and to get out, additional strategies have to be provided, and the resulting path is not optimal at all in any sense.

One of the most widely adopted local planner is the VHF algorithm [5]. This scheme starts from a grid map of the surrounding environment and builds a polar histogram that tells him what direction is the one with less obstacles and then steers the robot, while adjusting the speed. Another popular choice is the DWA algorithm [6]. This method starts from the current state of the robot, in terms of pose and velocities,and explores the space of velocities of the robot that can be reached in a small amount of time with a maximum acceleration. By so doing it creates several trajectories, that have to be obstacle-free, and then chooses among them the one that maximizes a certain function. In the canonical version this takes into account the distance from the obstacle, the distance from the target and the velocity of the robot, to achieve a trajectory that keeps the robot away from obstacles while driving it to the target at high speed. Also other functions can be used like in [6]. One limit of this method is that it usually considers only simple shaped path, e.g. straight lines and circle.

2.2.3 Hybrid planners

Hybrid planners are a more recent family of algorithms, compared to the previous two, and were born in order to get the advantages of both global and local planners,by making use of a combination of the them. An example of it can be the use of a global planner on a simplified map for creating sub-goals that the local planner will have to reach, breaking a big problem into several smaller and simpler ones. In addiction also methods that pre-compute the whole path and then deform it in real-time for dealing with unkown or moving obstacles are available as in [7]. Over the last year this approach has gained popularity [1].

2.3 Energy efficient trajectory planning

Thought there has been and there is a big amount of research in motion planning for mobile robots, not much has been done in the field of energy-efficient navigation for online applications, that is still an open field due to the complexity of the problem.

In [9] a first attempt to achieve energy efficient strategies is made. In this paper a generic robot is considered, time constraint on the arrival time are present as well as obstacles. From the energy point of view, friction losses are considered constant and no energy regeneration

while breaking is taken into account. The proposed scheme computes a mean velocity capable of driving the robot to the goal in the desired time. If the speed of the robot, that decreases for avoiding obstacles, is below this level an increase in speed is performed, otherwise no action is done. This approach however doesn't take into account, for example, the way to avoid obstacles.

The work presented in [10] focuses in offline computations to determine the best shape for the path of a robot whose task is to explore an area, from an energy point of view. An holonomic robot is considered and the energy consumption is computed by taking into account the energy used by the DC motors, modeled using a polynomial function of the angular velocity of the wheels that returns the energy consumed per radiant of rolling.

Looking at most recent works, in [12] a differential-drive robot is considered and a global path is computed using an A^* algorithm over a grid based map, then smoothed using Bézier curves. The dynamic behavior of the robot is analyzed as well as the motors dynamics and the energy consumption is computed taking into account kinetic energy lost due to imperfect regeneration, friction and sensors. Using energy consumption, an optimal trajectory is computed and then the results are verified implementing the proposed algorithm. The main limit if this approach is its offline nature, also no accounting for mobile unpredictable obstacles is done.

In [11] the problem of a robot turning between two straight paths is considered. The robot is differential-drive type and energy consumption is modelled taking into account the dynamic behavior of the robot and its motors. Then, using Pontryagin's maximum principle, partial differential equations to solve the problem are used for the straight path and for the turning part, and finally an optimum is searched varying the initial and final moment of the turn. In this way, combining these two geometries it's possible to move everywhere in the plane, however this method is meant only for offline computations.

A variation of the DWA algorithm was proposed in [13] for enhancing energy efficiency. The energy consumption is the sum of three terms, that represents the energy adsorbed by sensors and controllers, inertia actions and friction. Energy efficiency is achieved by introducing in the cost function a factor for the energy used. The strong point of this approach is its capability of working online but several limitations, coming from the DWA approach are still present.

2.4 Model predictive control

Model predictive control is a control scheme that over the last forty years has faced a great development from both the theoretical aspects and the real world applications and is today widely adopted for process control in the chemical sector [15]. Some of the reasons for his success are the capabilities of dealing with systems that have multiple input and outputs, his robustness with respect to unpredictable events that may occur during the process and the possibility of taking into account constraints on both states and outputs.

To understand its workflow observe the figure 2.4. The basic components of an MPC controller are a model of the plant to control, a cost function to optimize and an optimizer. The controller receives as input a reference for the controlled variables and the current state of the plant, in case of non complete access to it, a state estimator is used. Then a discrete-time optimization problem is solved over a finite time horizon of the form:

$$\begin{array}{lll} \min & J(s_0, u) & \\ \text{s.t.} & h(s_0, u) & \leq \quad 0 \\ & e(s_0, u) & = \quad 0 \\ & u \in R^N & \end{array}$$

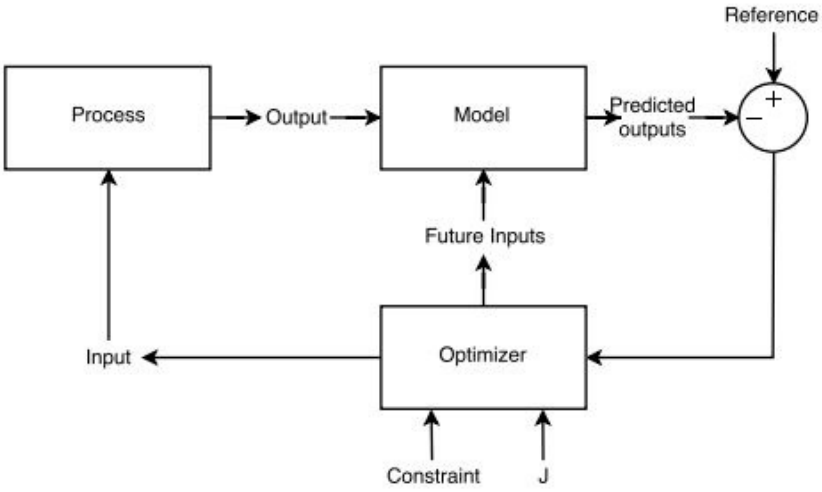


Figure 2.4: MPC workflow scheme

Where N is the length of the time horizon, u is the vector of decision variables, a series of input to the system at different times, h and e are equalites and inequalites constraints on the problem. J is a function of the inputs and of the future states of the systems s, that are predicted using the current state, the decision variable and the model of the plant to control. A typical example of J could be the sum of the squared differences between the reference and the futures states opportunely weighted and often a final cost, function of the final state of the model, is added to ensure stability. After the optimization problem is solved, only the first element of the input vector is sent to the plant, the first input in the temporal sequence, and at the next iteration an analogue problem is solved. Since only the first action is used and the problem is solved again at each time sample, this control strategy is more robust with respect to unpredicted events compared to an offline solving of an optimal control problem. Besides, even if the performance in ideal cases are inferior to the optimal, because the problem solved is a discrete time one and the time horizon is finite, they are still very good.

A serious disadvantage is however present in solving the problem at each iteration online: it requires a computational effort that prevented and still limits the application of model predictive control in many cases. This problem has been partially overcome during latest years thanks to an increase in the computational capabilities of available hardware and to the development of efficient optimization solvers for MPC-type optimization problems[17]. Another way of dealing with the problem is the use of a variation of model predictive control named explicit model predictive control. This approach computes offline the optimization problem for many initial states and references and memorizes them so than instead of solving the problem online it is possible to look for the solution in the memory. Still the memory needed for this approach limits its application to simple systems with a low-dimension state[18].

Model predictive control applied to the generation of a trajectory slightly differs from the trajectory-tracking scheme so far introduced. In 2.5 a scheme of how it works it visible. In this approach, the optimizer receives the goal location, instead of the distance from a given trajectory, and solves at each time step the same discrete-time optimization problem.

However the decision variable is no longer a set of control variable, like motor's torques, but

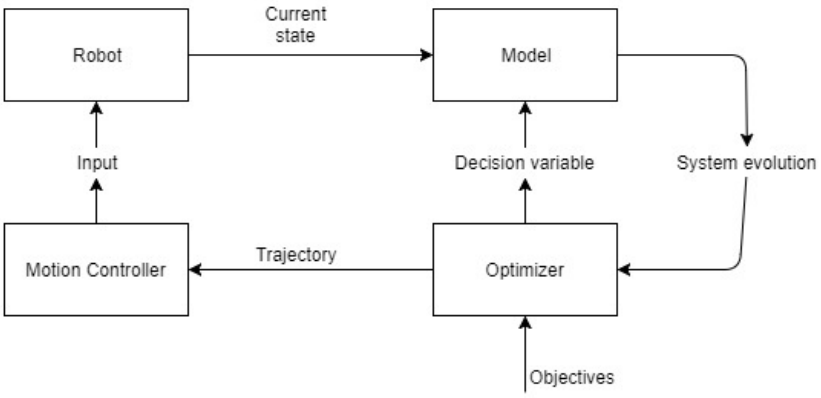


Figure 2.5: MPC for trajectories generation

it's a set of kinematic variables, again one at each time sample, of the robot chassis, like the robot acceleration or its jerk. Besides, once the optimal set is reached the output is not necessarily the first element of the set. In fact if the control variable is an acceleration and the output is a velocity trajectory, the decision variable will be integrated to generate it, however only the first step of this trajectory will be given as output. Another difference is the presence of a motion controller that will be responsible for tracking the output trajectory.

In recent years, several attempts have been made to use model predictive control for trajectory generation. In [19] a model predictive approach was used to generate trajectories for a robot arm that was trying to catch a moving ball. Another example can be found in [20]. Here a multi-stage planner is proposed that tries to balance between global and local planning. This proposal considers all non-holonomic constraints that the robot structure poses only in the planning phase closest to the present position of the robot and then it simplifies the problem for time samples that are far in the future.

Chapter 3

1-D case

In this chapter, in order to get more confident about the method and to test its viability, and also to be able to distinguish the influence of the various factors more easily, a simple robot moving along a straight line is considered. His kinematic, dynamic and energy consumption models are derived. The proposed approach is then explained and applied to this simple robot. Three formulations and cost functions will be used and simulations of its behavior will be carried out and the obtained results will be discussed.

3.1 Kinematic and dynamic model

The simple robot in exam consists of a conventional non-steerable wheel, whose geometry can be seen in 3.1, actuated by a linear DC motor acting along its axis.

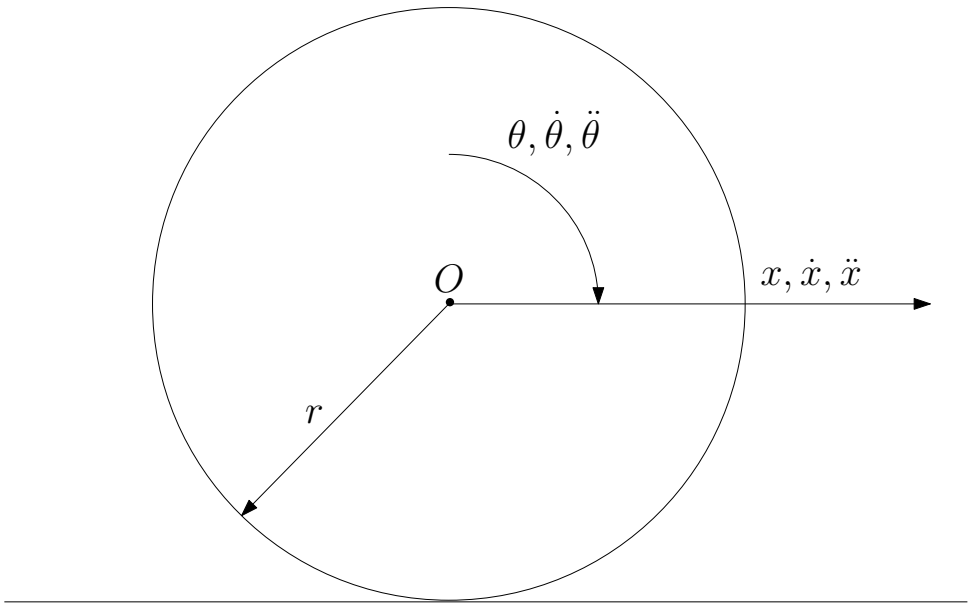


Figure 3.1: Simple robot geometry scheme

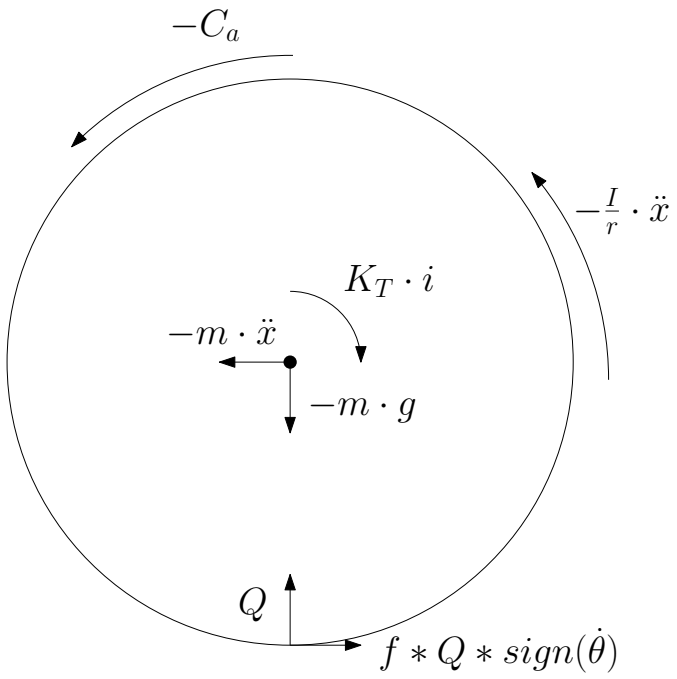


Figure 3.2: Scheme of the free body diagram of the robot

Further simplifying assumptions are made:

- The wheel is a rigid body.
- The wheel axis is always parallel to the ground.
- Pure rolling without any slip at the contact point is assumed.

Under these hypothesis, the kinematics equations derived from the above scheme, are:

$$x = r * \theta \tag{3.1}$$
$$\dot{x} = r * \dot{\theta} \tag{3.2}$$
$$\ddot{x} = r * \ddot{\theta} \tag{3.3}$$

To derive a dynamic model of the system, let's consider the schemes of Fig. 3.2 and 3.3. From the free body diagram follows:

$$Q - mg = 0$$
$$fQsign(\dot{\theta}) - m\ddot{x} = 0$$
$$K_T i - \frac{I}{r}\ddot{x} - C_a - fQsign(\dot{\theta}) = 0$$

Where Q is he reaction force from the ground, m the robot mass, g the standard acceleration due to gravity, f is the static friction coefficient, K_T is the torque constant of the DC motor and C_a is a friction torque that is assumed to be the sum of a constant term and a viscous one.

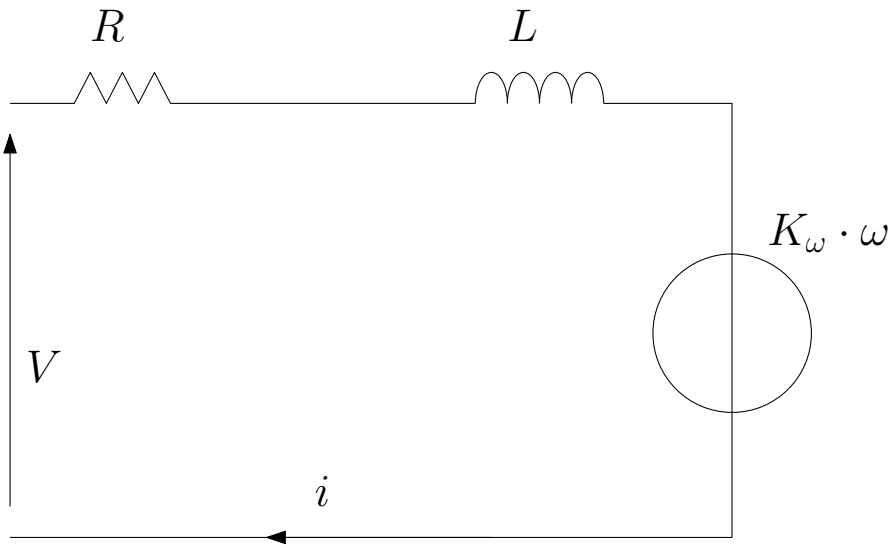


Figure 3.3: Schemes of robot's DC motor

While from the motor diagram in 3.3 , using Kirchoff's laws it follows:

$$V - Ri - L\dot{i} - K_\omega \dot{\theta} = 0$$

Where V is the voltage applied and i the current drained from the battery of the robot and K_ω is the speed constant of the DC motor. Two important remarks about the derived models should be pointed out. First, since the electric system is much faster than the mechanical one, in the following the inductance will be neglected. The second one is that these models will be used in an inverse way, in the sense that instead of computing the evolution of the system from a given applied voltage they will start from the kinematic, imposed by the motion planner, and will compute the necessary actions.

3.2 Energy consumption model

Starting from the above derived models, it is possible to compute the energy consumed by the robot when it moves along a certain trajectory. Since we're interested in the total energy drained from the batteries, the energy we have to compute is the one that enters into our system from the power circuit of the DC motor. To obtain an instantaneous power balance equation from the dynamic model above derived we multiply the DC motor equation by the current obtaining:

$$Vi - Ri^2 - K_\omega \dot{\theta} i = 0$$

The first term is the power from the battery, the second is easy to recognize as the power dissipated from the resistance while the third one is the energy given to the mechanical system. Note that given the conventions used the power will be positive entering the system and negative when being dissipated or given. Considering the mechanical part of the system we can obtain:

$$K_T i\dot{\theta} - (I + mr^2)\ddot{\theta}\dot{\theta} - C_a\dot{\theta} = 0$$

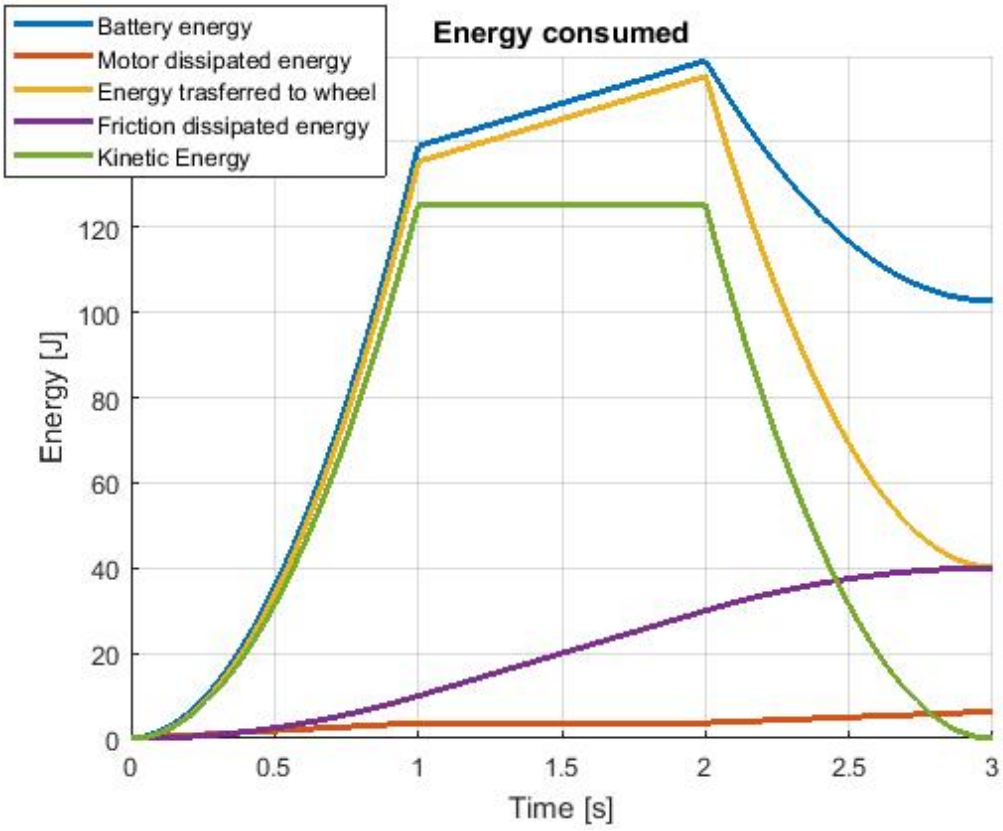


Figure 3.4: Energy consumed over time.

Where it is possible to identify the first term as the power coming from the DC motor, the second one as the power needed to win the inertia forces, and that will be converted in kinetic energy, and the third one is the power dissipated from the friction torque. Starting from these power terms the energies related to each process can be easily computed through integration.

However a remark needs to be made. The term $V * i$ would be the power from and to the battery only if the regeneration of power while breaking would have been perfect. Since this is not the case, a constant efficiency η is considered, the power of the battery P_b will become:

$$P_b = \begin{cases} Vi & \text{if } Vi > 0 \\ \eta Vi & \text{if } Vi < 0 \end{cases}$$

It is now possible to use this model to compute the energy spent while following a trajectory with the robot. An example has been computed using as input a trapezoidal velocity input and the results can be seen in Fig. 3.4.

3.3 Application to the simple robot

In this section the proposed approach, MPC for trajectories generation, will be applied to the simple robot modeled in this chapter, using different formulations. Simulations of the resulting

robot behavior were carried out and the outcomes, the pros and cons will be discussed.

3.3.1 Unconstrained formulation

In this formulation the decision variable will be the acceleration of the simple robot, the models to simulate and to predict the robot's behavior will be those derived in the sections 3.2 and 3.1 and a perfect controller is assumed, capable of tracking the given reference trajectory. The optimization problem to solve at each iteration will be:

$$\begin{array}{lll} \min & J(s_0, u) & \\ \text{s.t.} & h(s_0, u) & \leq \quad 0 \\ & u \in R^N & \end{array}$$

Where N is the length of the time horizon, u is the decision variable, in this case a vector containing an acceleration at each time sample in the time horizon and s is the state of the robot. The cost function, that will be evaluated after simulating the system evolution to a set of accelerations along the whole time horizon, will be:

$$J = \sum_{k=1}^{N} C_1 P_b(k) dt + C_2 (x(k) - x_o)^2 + C_3 \ddot{x}(k)^2$$

Where P_b is the instantaneous power consumption, estimated using the models derived in the previous sections 3.1, dt is the time step, x_o is the target goal, and C_1,C_2 and C_3 are constant that weight differently the importance of each term of the function, respectively the power consumption, the distance from the target and the acceleration of the robot. These three constants are tunable and their values affect greatly the behavior of the resulting planner. Also limits on the maximum acceleration u_{max} were set, so that h is a vector of $2N$ functions:

$$\begin{aligned} h_i &= u_i - u_{max} \quad i = 1, ..., N \\ h_k &= -u_{max} - u_{k-N} \quad k = N+1, ..., 2N \end{aligned}$$

In the following the effect of these constants and of the other parameters is pointed out.

In Fig. 3.5 one can see the results of a simulation with C_3 varying. As it is possible to see the increase of the C_3 weight brings a decrease in energy consumption. However, since no effort is put in this formulation to fix a precise arrival time it can be seen that there is an increase in the time the robot reaches its target.

The effect of a non-zero value of C_1 can be seen in Fig. 3.6. As it can be seen an increase in the C_1 value reflects in a lower energy consumption, at the price of an increase in arrival time. Another effect that can be seen is that the acceleration profile is non-smooth. This is particularly undesirable, as it differs very likely from the energy-optimal solution, as they resolve in unnecessary movements, and can induce vibrations in the robot behavior. In order to try to limit this phenomenon, a combination of the first and the third term of the cost function has been used, the first one dealing with energy efficiency and the third one trying to limit acceleration spikes. The simulation results for this approach can be seen in Fig. 3.7: the resulting acceleration profile is smoother and it also resulted in a better energy efficiency.

In the end this formulation has proven to be able to improve energy efficiency for the simple robot and to generate kinematic references profiles that are smooth. However the arrival time is not fixed and tends to increase with energy efficiency. Also the outcome depends on the relative weigths of the constants C_1,C_2 and C_3 and there is no guarantee that in a different situation,

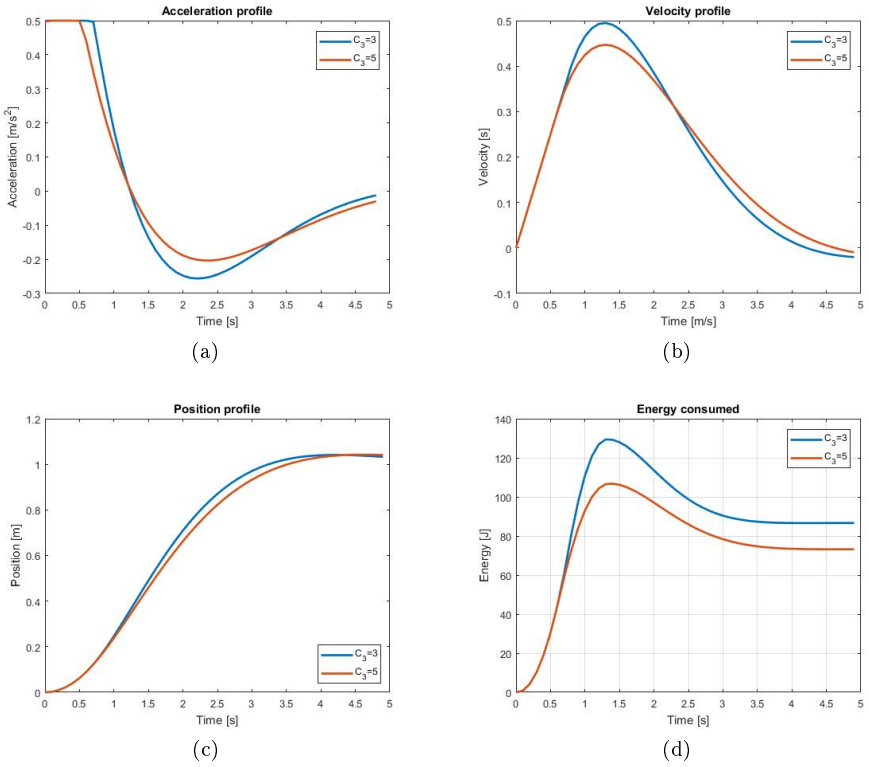


Figure 3.5: In this simulation results one can see the influence of the C_3 term. The simulation parameters were: $dt = 0.1s$, $N = 70$, $C_1 = 0$, $C_2 = 5$. The maximum number of iterations permitted was 20.

for example if the limits on the acceleration were different, the same constants would perform the same way they do in one specific case.

3.3.2 Split formulation

In order to overcome the varying arrival time the previous formulation led to, a second formulation is proposed. Also in this formulation the decision variables will be a set of acceleration over the time horizon, limited on their maximum values so that the optimization problem to solve is the same and the models used to predict and simulate the system behavior will be the same too, however the cost function used is different. In this formulation the cost function will be:

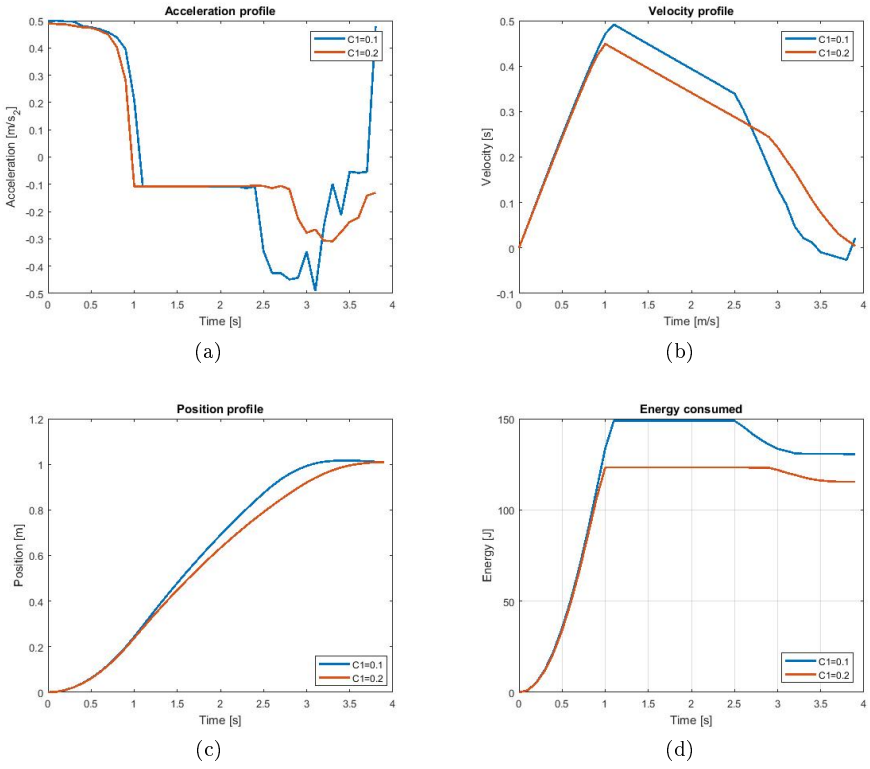


Figure 3.6: In this simulation results one can see the influence of the C_1 term. The simulation parameters were: $dt = 0.1s$, $N = 30$, $C_3 = 0$, $C_2 = 5$. The maximum number of iterations permitted was 20.

$$J = J_1 + J_2$$

$$J_1 = \sum_{k=1}^{S} [C_1 P_b(k) dt + C_2 (x(k) - x_o)^2 + C_3 \ddot{x}(k)^2]$$

$$J_2 = \sum_{c=S+1}^{N} C_4 (x(c) - x_o)^2$$

This cost function has many terms in common with the previous one, however it is very different as it splits the time horizon over which the optimization is done into two parts. The first one is the same as before, while the second one only accounts for the squared distance from the target, adding another tunable constant C_4. The difference, that is an effort to enforce the arrival time, is that the first part only applies to the part of the time horizon that is before the arrival time, being S the biggest natural number such that:

$$\epsilon * S * dt + t_0 < t_{end}$$

Where t_0 is the time at which the computation is done and ϵ a real number between 0 and 1, whose purpose will be explained later. In this way, if C_4 is bigger than the other constants,

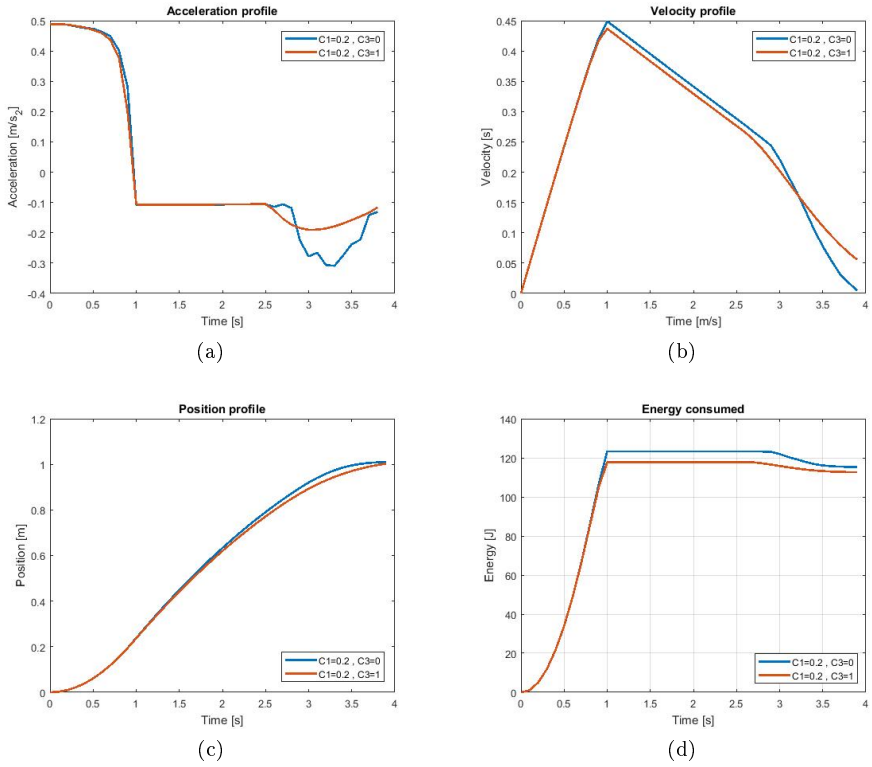


Figure 3.7: In this simulation results one can see the influence of the combination of the C_1 and C_3 terms. The other simulation parameters were: $dt = 0.1s$, $N = 30$, $C_2 = 5$. The maximum number of iterations permitted was 20

the cost function will receive a penalty if after the arrival time the robot is not yet at the target location, while keeping the energy efficiency terms during the first phase. The constant ϵ is used to enlarge or tighten the amount of time in advance to the arrival time the second condition triggers. In fact the lower its value the larger the part of time horizon during which the second part of the function will be used, and so a better match between the prescribed arrival time and the actual arrival time. In Fig. 3.8 the position profiles of simulations carried out increasing C_4 and keeping everything else stable are shown. One can see that as C_4 increases, the position at the arrival time, 3 seconds, gets closer to the target location, that is 1 meter.

Also this formulation, as well as the previous one, is capable of increasing energy efficiency given higher values to the constants dealing with acceleration and energy, and better results, both in terms of smoothness and lower consumption, are often achieved using a combination of the two, as can be seen in Fig. 3.9.

On the other hand, this formulation introduces one more shortcoming with respect to the previous one. In fact, if the arrival time is not inside the initial time horizon, it is possible, depending on the balance of the several constants, that the situation visible in Fig. 3.10 arises. In this simulation, where the time horizon is 2 seconds and the arrival time is set at 4 seconds, the robot starts to move slowly in order to be energy efficient and only after some time the part of the cost function that tries to enforce the arrival time gains enough weight to influence strongly the behavior of the robot.

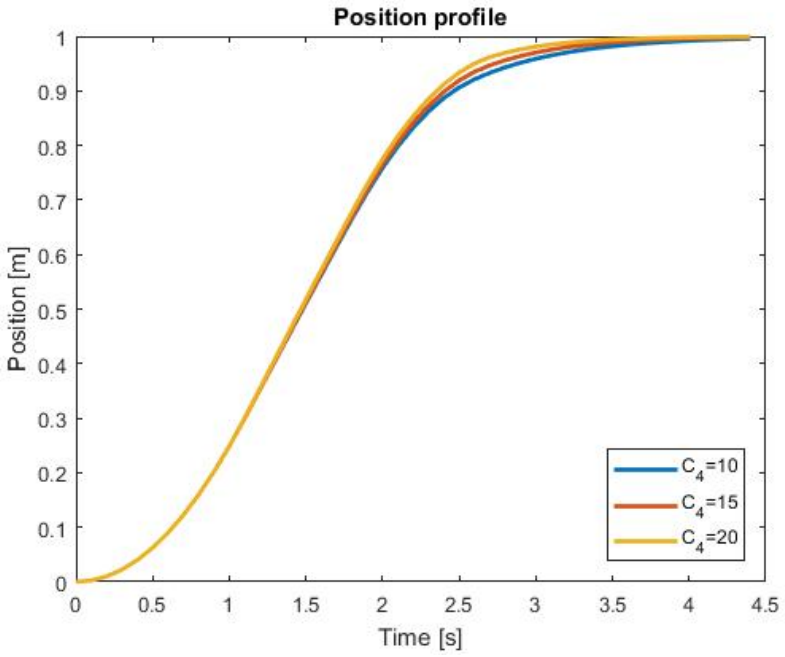


Figure 3.8: In this simulation results one can see the influence of the C_4 constant. The other simulation parameters were: $dt = 0.1s$, $N = 40$, $C_2 = 10$, $C_1 = 0.1$, $C_3 = 1$, $\epsilon = 0.8$. The maximum number of iterations permitted was 40

When this happens a sudden increase in speed is given to the robot and the overall energy consumption is increased as well.

3.3.3 Hard-constraints formulation

One of the most significant disadvantages of the previous two formulations is the fact that the optimization problem to solve at each step is a multi-objective one. The resulting performances of the robot will depend on the different weights given to each part of the cost function, in such a way that even if a specific set of values for the constants can perform well in certain situations, there is no proof that it will be the same in different scenarios, for example if the limits on the acceleration of the robot were to change. To solve this problem the following formulation of the optimization problem for the MPC for trajectories generation is used. In this case instead of using the acceleration at every time sample of the time horizon as decision variable, the whole system evolution will be the decision variable u, so that it will be a vector containing the acceleration, the speed and the position of the robot at every time step of the horizon. In this way it is more convenient to set hard constraints on the positions, as they are directly accessible, compared to the other formulations, where the positions were a complex linear combination of the starting state and of the decision variable that was varying with the length of horizon. The optimization problem to solve at each time step will be in the form:

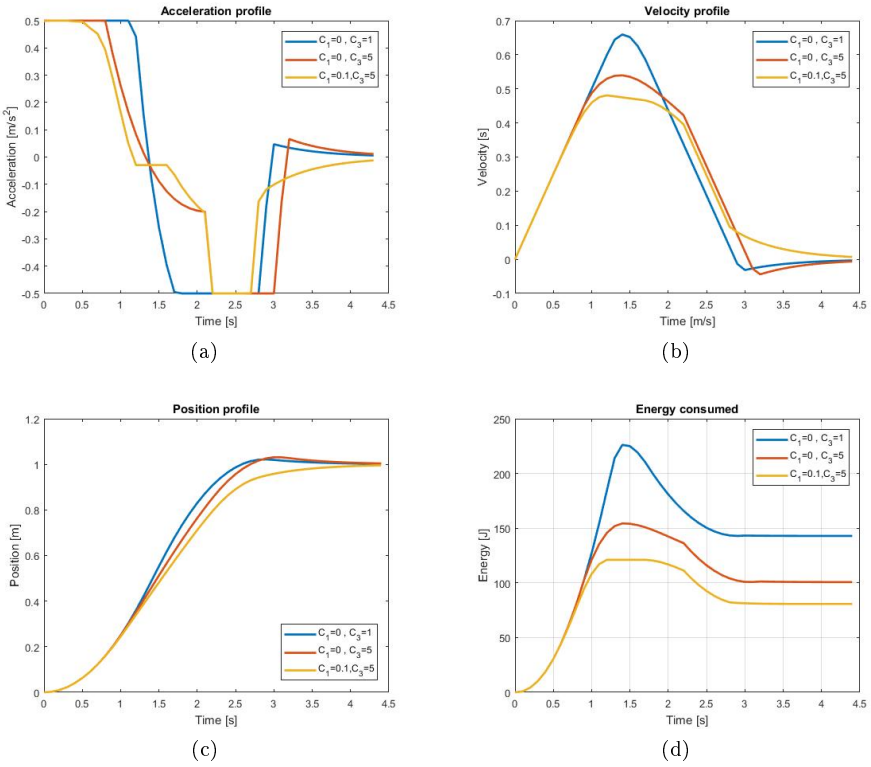


Figure 3.9: Effects of $C1$ and $C3$ on the simple robot behavior. The other simulation parameters were: $dt = 0.1s$, $N = 40$, $C_2 = 10$, $C_4 = 10$, $\epsilon = 0.8$. The maximum number of iterations permitted was 40

$$\begin{array}{lrcl} \min & J(u_0 u) & & \\ \text{s.t.} & h(u_0, u) & \leq & 0 \\ & e(u_0, i) & = & 0 \\ & u \in R^N & & \end{array}$$

Being u_0 the current state of the system at each iteration. Given the evolution of the system, the cost function to minimize will be:

$$J = \sum_{k=1}^{N} P_b(k) dt$$

As it is possible to see in this cost function there are no tunable constants, just the power consumed. In this formulation the role of the model of the system is no longer a prediction, as the whole evolution is the decision variable, but it is used in order to ensure that the selected evolution is feasible for the system. Starting from the kinematic model and using explicit Euler method to discretize with respect to time the differential equations, the following equations are used:

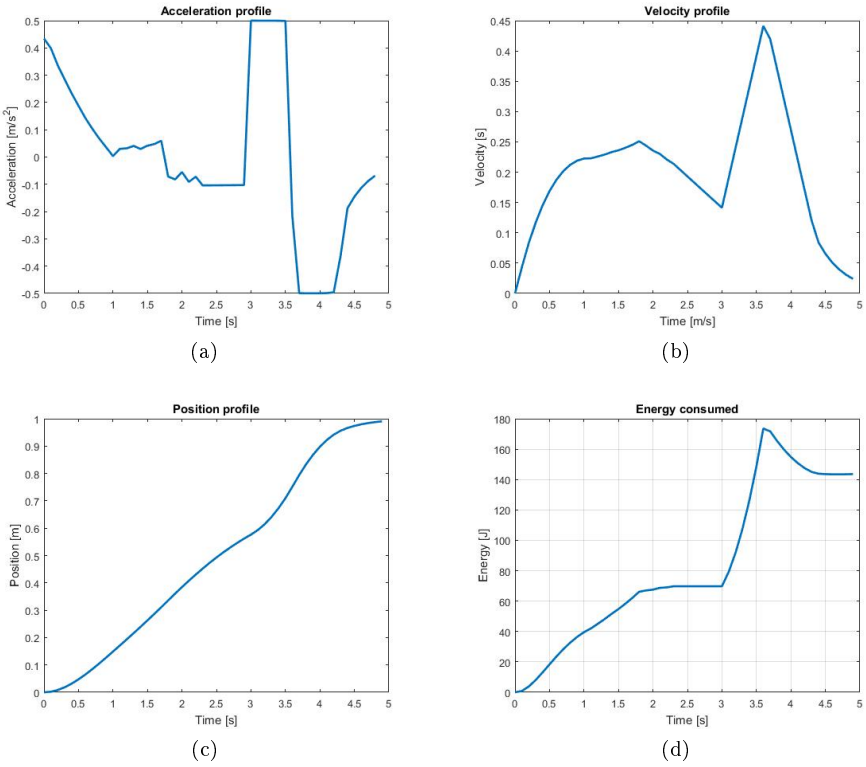


Figure 3.10: Effects of a late recognition of target: if the robot recognizes that it's too slow to reach the target in time as it enters the time horizon a sudden acceleration occurs

$$\dot{x}(k+1) = \dot{x}(k) + \ddot{x}(k) \quad k = 1, ..., N-1$$
$$x(k+1) = x(k) + \dot{x}(k) \quad k = 1, ..., N-1$$

Where the position and the velocity of the robot at the next step are expressed using the variables of the current one. These equations will become part of the equality constraints $e(u_0, u)$ of the optimization problem. Besides, all positions after the arrival time will have to be equal to the target location. Also limits on the available acceleration and speed are set, so $h(u_0, u)$ will be:

$$h_i = \ddot{x}_i - \ddot{x}_{max} \quad i = 1, ..., N$$
$$h_k = -\ddot{x}_{max} - \ddot{x}_{k-N} \quad k = N+1, ..., 2N$$
$$h_j = \dot{x}_{j-2N} - \dot{x}_{max} \quad j = 2N+1, ..., 3N$$
$$h_b = -\dot{x}_{max} - \dot{x}_{b-3N} \quad k = 3N+1, ..., 4N$$

In Fig. 3.11 an example of simulation results obtained using this formulation can be seen. It can be noticed that also in this case the outcome profiles are non-smooth.

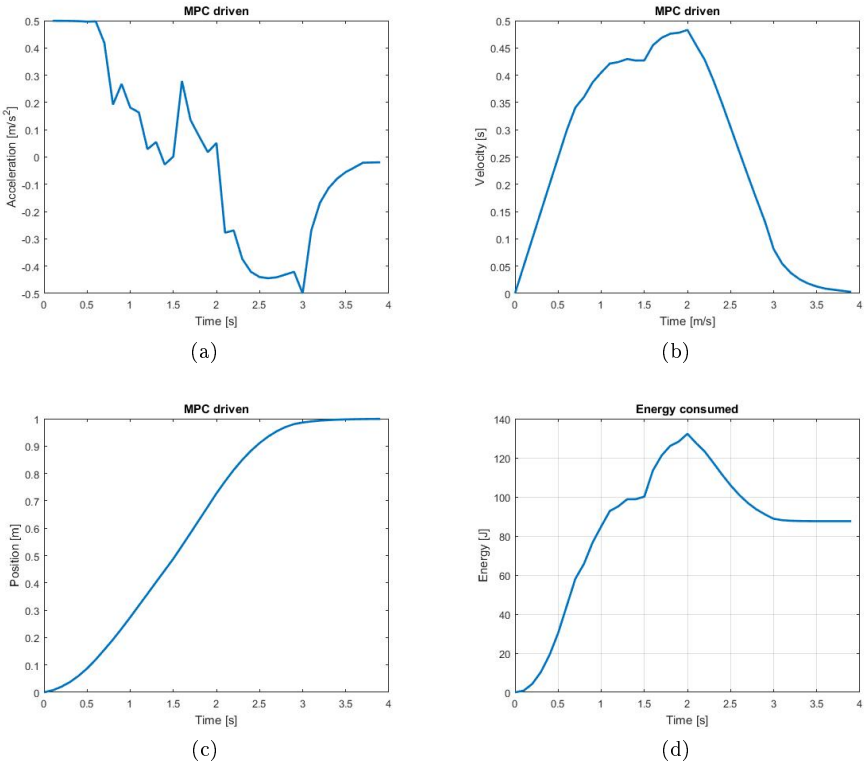


Figure 3.11: Simulation results using hard-constraint formulation. The parameters of the simulation were: $dt = 0.1s$, $N = 40$ and maximum number of iterations equal to 40.

About the disadvantages of this approach there are two main things that should be pointed out: the first one is that it requires more computing power than the other methods since the decision variable to optimize is three times bigger, and the second one is that it is possible that the optimization fails to converge to a feasible point due to the limited number of iterations. If this happens during real time execution the system can exhibit improper and also dangerous behavior. Obviously, the more computing power available the better results will be achieved, as a more dense time discretization can be used and also more iterations during optimization. In Fig.3.12 one can see the results of two simulations using the same formulation: the first one uses 20 iterations and 10 steps per second while the second uses 500 iterations and 20 steps per second. As it can be seen the second has better performance and shows that the method, given enough computing time, will converge to the optimal solution. However since every iteration required over one hundred seconds to be computed it is not suitable for online applications.

To conclude, a comparison between the second and the third formulation will be done. In order for this comparison to be fair the first formulation will not be included, even though it proved to be completely viable. This is because it doesn't fix the arrival time, a parameter that has a huge influence on the energy consumption. Another key aspect for this comparison to be significant is the computing time spent at each iteration, that has to be similar for the two formulations in order to have a meaningful confront. For this reason the required computing time were computed and are shown in Fig. 3.13 for both formulations. The first one is not present for the reasons above, however it can be said that it is way faster than both of the other two.

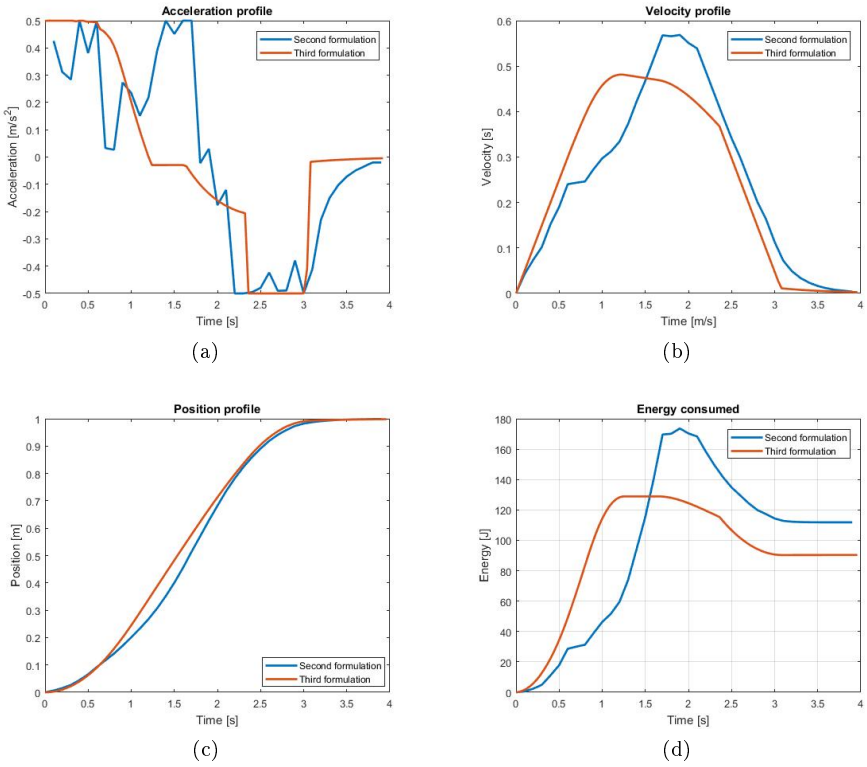
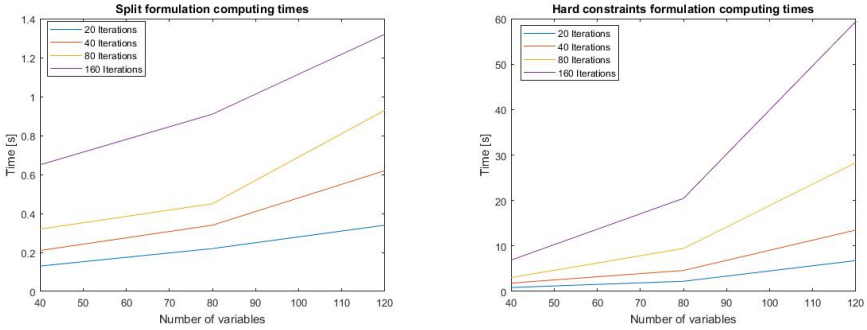


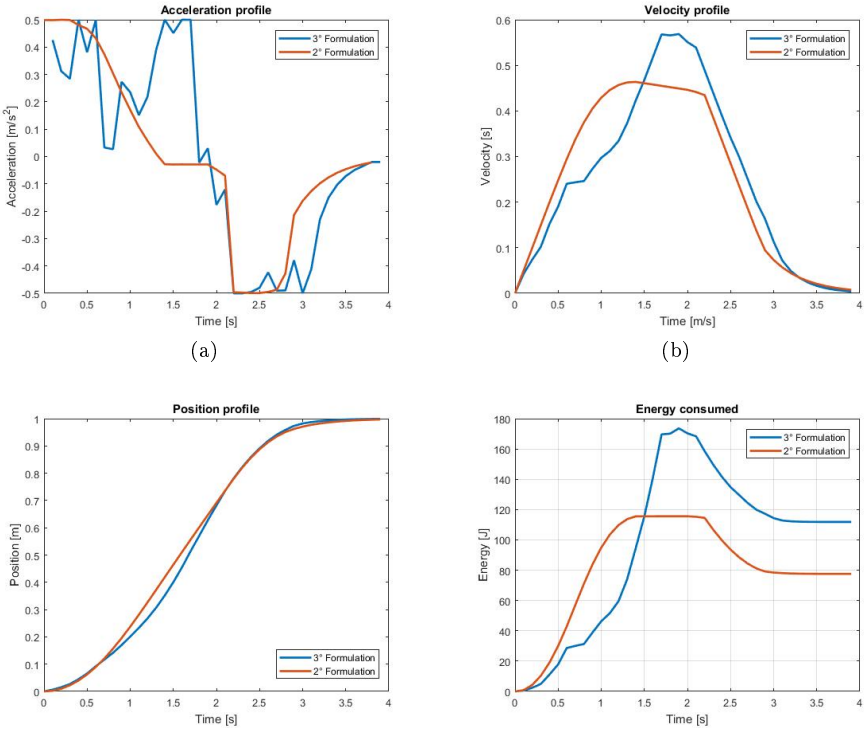
Figure 3.12: Effects of more computing power available on hard constraint formulation. The blue curves were obtained using 20 iterations and 10 steps per second, while the red one used 500 iterations and 20 steps per second.

These data were acquired running the simulations in the Matlab environment on a laptop with a CPU clock of 2.4 GHz. From the figures one can see that the third formulation is slower than the second one. This was expected because given a time horizon of N steps this formulation optimizes $3N$ variables, and as it can be seen in the figures the influence of the number of variables is important. For the comparison it was chosen for the split formulation a time horizon of 120 steps and maximum number of iterations equal to to 80 that gives a computing time roughly around 1 second. For the hard-constraints formulation 40 steps for the time horizon and 20 iterations were chosen, and the resulting computing time is around 1 second as well. From the figure one can see that the performance of the split formulation are better. Also lower computing times can be achieved lowering N and the number of iterations if the need arises, while for the hard constraints formulation the lowest computing time is already considered. For these reasons the split formulation will be the basis for the 2-D case, explored in the next chapter.



(a) (b)

Figure 3.13: Comparison between second and third formulation computing times



(a) (b)

(c) (d)

Figure 3.14: Comparison between second and third formulation performances using a computing time around 1s. The parameters for the second formulation were $N = 120$, $dt = 0.4s$, $C_1 = 0.2$, $C_2 = 5$, $C_3 = 5$, $\epsilon = to0.8$ and iterations limit equal to 80. The hard constraint formulation instead used $N = 40$ and iterations limit equal to 20.

Chapter 4

2-D case

In this chapter the proposed approach, already tested and modified during the simulations presented in the previous chapter, is applied to the ROPOD robot. Firstly suitable kinematic, dynamic and energy consumption models are derived. Then two formulations are proposed and simulations of the ROPOD behavior are carried out both in simpler and more complex environment.

4.1 Kinematic model

In this section the structure of the ROPOD robot in presented, followed by a general way of deriving kinematic models for mobile robots.

4.1.1 Geometry and configuration definition

In order to be cheap and reliable at the same time the kinematic structure of ROPOD is composed of four pairs of twin wheels, a couple of which can be seen in figure 4.1.

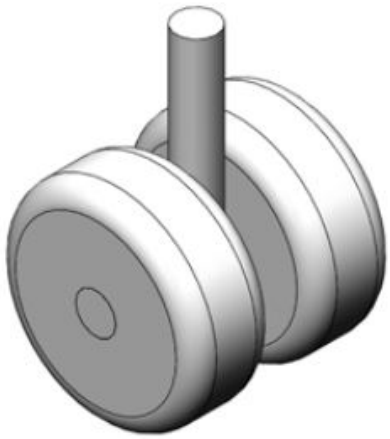


Figure 4.1: Twin steerable wheels

These kind of wheels are equivalent to a normal steerable wheel but have a simpler structure. In fact when, intuitively, when both wheels are spinning in the same direction we've got movement along that axis, while if they spin in opposite directions the wheel turns around the shaft. In order to be holonomic the four wheel pairs are arranged as can be seen in figure 4.2.

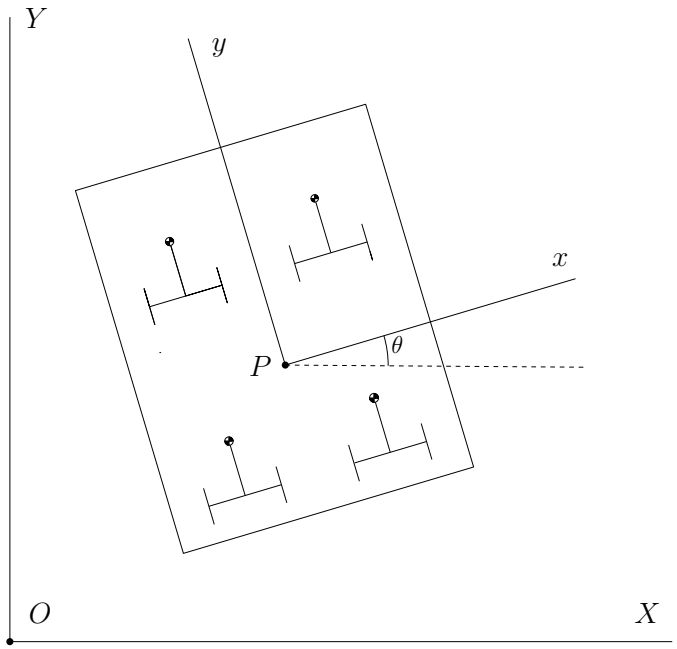


Figure 4.2: ROPOD kinematic structure

In order to describe the robot configuration at each time instant, a proper set of variables has to be used, sufficient to fully describe the state of each part of the robot. First step in choosing these variables is the choice of reference frames, one fixed or global and one attached to the robot or local, in order to be able to describe the robot position and orientation in the space as a rigid body. We'll denote from here on $O - XY$ as the global reference i and $P - xy$ the one attached to the chassis of the robot, also referred to as r, where P is a reference point on the robot chassis we can chose as we please. Their relative orientation can be expressed thanks to a rotation matrix around the common vertical axis. Being θ the angle between the X and x axes of the frames, positive in the counterclockwise direction and starting from X the matrix relating the global frame to the robot's frame is:

$$R(\theta) = \begin{bmatrix} cos(\theta) & sin(\theta) \\ -sin(\theta) & cos(\theta) \end{bmatrix}$$

To fully describe the robot from a kinematic point of view we'll also need to refer to single wheels to account the motion constraints they pose. In figure 4.3 the geometry of a generic wheel is shown.

We can see that for its geometry we have five constant parameters, namely α, l , d , b and the wheel radius r, and two variables, β and the angle ϕ that accounts for the wheel rotation around his axis. It's now clear that for fully describe the robot's configuration we need:

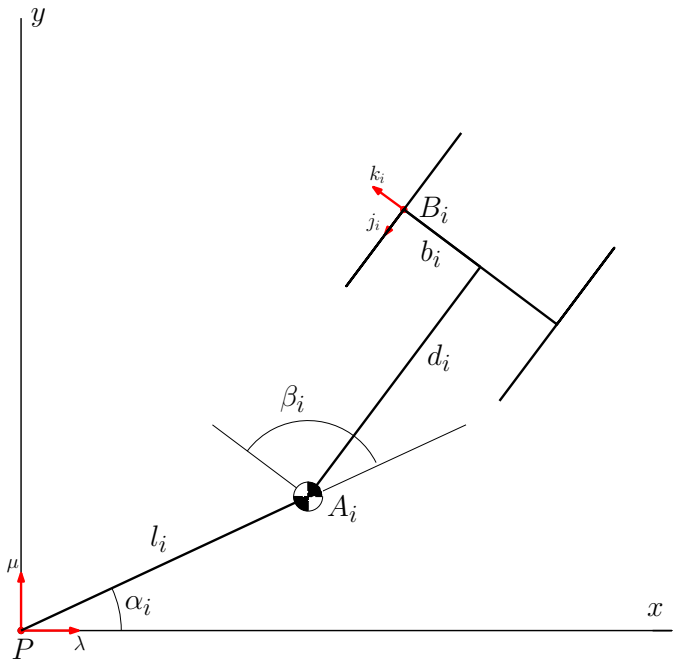


Figure 4.3: Geometric scheme for a wheel of ROPOD

- Three variables to define the position of P with respect to O and the orientation if the robot, namely X, Y, θ
- Four angles β as each wheel pair has the same angle and they are parallel to each other.
- Eight angles ϕ, one for each wheel

For a total configuration variables number of 15. We can thus define the state vector of our kinematic model as:

$$\mathbf{q} = \begin{bmatrix} X \\ Y \\ \theta \\ \beta_1 \\ \vdots \\ \beta_4 \\ \phi_1 \\ \vdots \\ \phi_8 \end{bmatrix}$$

4.1.2 Hypothesis

In the following development of ROPOD robot kinematic models the following assumptions were made:

- The robot is moving in a plane environment, no irregularities are taken into account and so all wheels are in contact with the ground at each time instant.
- The axle of every wheel of the robot is parallel to the ground and so it remains. Also wheels are perfectly round.
- Only perfect rolling motion is accounted, so no slip at the contact point is admitted. This means, from a physical point of view, that a wheel is allowed to move only in the wheel's plane or to rotate around the contact point as there is no lateral slip. Also since the rolling is perfect there is proportionality between distance covered and rotation made by the wheel.
- Velocities are considered as input for our robot configuration. This choice is the one most widely adopted in mobile robots field and leads to simpler models. This is acceptable for many applications thought it's very important reminding that the robot in real world cannot stop instantaneously, for example for avoiding obstacles, but only a finite acceleration can be provided. Moreover high values of acceleration can lead to slip, thus creating errors in the model predictions, as it doesn't account for that.

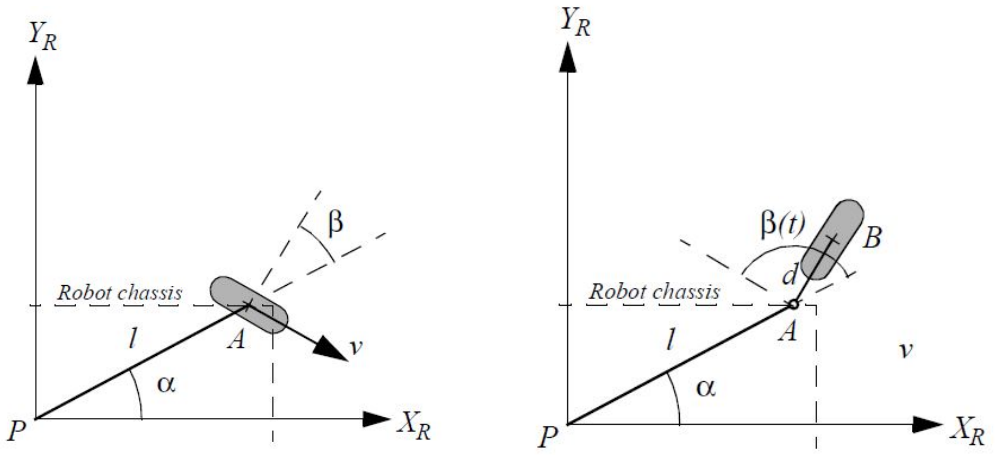
Under the previous assumptions there are two ways to derive the kinematic model: the first one is to build a direct model giving as an input the velocities at the wheel level and then computing the resultant movement of the main frame. Another one is to write the constraints that each wheel poses to the motion of the chassis and then, starting from the movement of the main body derive what should be the resultant for each component so that the constraints are satisfied. Even though this second approach may seem less intuitive, it has gained large popularity in the mobile robots modelling due to the fact that is better suited to deal with complex kinematic structures. Besides from the resulting model is easier to analyze if the motion of the robot is constrained or not. Given the complexity of the kinematic structure of the ROPOD the second path has been here chosen.

4.1.3 Single wheel constraints

The two conditions that each wheel has to satisfy are namely the non-slip condition, no motion along the wheel axis is allowed, and the rolling condition, that states that since rolling is perfect there is proportionality between distance covered from the wheel and its ϕ. angle. Since the equations that enforce these conditions depends only on the geometry of the wheels itself and how it is attached to the chassis, it is possible to find in the literature the expressions already written for the most common type of wheels and geometries. In **??** it is possible to see a conventional wheel and a caster one. For the first scheme the two conditions can be written as:

$$[sin(\alpha+\beta), -cos(\alpha+\beta), -lcos(\beta)]R(\theta)[\dot{X},\dot{Y},\dot{\theta}]^T - r\dot{\phi} = 0$$
$$[cos(\alpha+\beta), sin(\alpha+\beta), lsin(\beta)]R(\theta)[\dot{X},\dot{Y},\dot{\theta}]^T = 0$$

The second equation states that not every $[\dot{X},\dot{Y},\dot{\theta}]$ is acceptable but only those that allow the wheel to move accordingly to the non-slip condition. The constraint that this equation poses to the mobility of the robot is non-holonomic, meaning that, in an obstacle-free environment,



(a) *Standard fixed wheel* (b) *Caster wheel.*

every $[X, Y, \theta]$ is reachable but there are constraints on the way they can be reached, that is on the admissible velocities of the robot chassis. For the second kind of wheel instead it holds:

$$[sin(\alpha+\beta), -cos(\alpha+\beta), -lcos(\beta)]R(\theta)[\dot{X}, \dot{Y}, \dot{\theta}]^T - r\dot{\phi} = 0$$
$$[cos(\alpha+\beta), sin(\alpha+\beta), (l+d)sin(\beta)]R(\theta)[\dot{X}, \dot{Y}, \dot{\theta}]^T + d\dot{\beta} = 0$$

Considering again the second equation it's possible to see that now every $[\dot{X}, \dot{Y}, \dot{\theta}]$ is acceptable as there is one more extra variable, $\dot{\beta}$. This means that the caster wheel doesn't limit the overall mobility as a standard wheel does.

Since the kind of wheel the ROPOD is using was not found in the literature, the proper conditions had to be derived as follows. Considering a wheel of the ROPOD n out of the eight we have the velocity of his center B_n is defined as:

$$\frac{d\mathbf{OB_n}}{dt} = \frac{d\mathbf{OP}}{dt} + \frac{d\mathbf{PA_n}}{dt} + \frac{d\mathbf{A_nB_n}}{dt}$$

Considering figure 4.3, let's compute each term in the robot frame using the configuration variables. The first one is the velocity of the robot, that in the robot's reference frame can be written, starting from its velocity in global frame and using the R matrix defined above, as:

$$\frac{d\mathbf{OP}}{dt} = (cos(\theta)\cdot\dot{X} + sen(\theta)\cdot\dot{Y})\cdot\boldsymbol{\lambda} + (-sen(\theta)\cdot\dot{X} + cos(\theta)\cdot\dot{Y})\cdot\boldsymbol{\mu}$$

The second term, being l and α constants, varies in time just because of the rotation of the robot, so it rotates around the point P :

$$\frac{d\mathbf{PA_n}}{dt} = [-l_n\cdot sin(\alpha_n)\cdot\dot{\theta}]\cdot\boldsymbol{\lambda} + [l_n\cdot cos(\alpha_n)\cdot\dot{\theta}]\cdot\boldsymbol{\mu}$$

To write the last term, we can see that in the robot reference frame it holds:

$${}^r\mathbf{A_nB_n} = R(\alpha_n+\beta_n-\frac{\pi}{2})\cdot\begin{bmatrix}d_n\\b_n\end{bmatrix}$$

And after its differentiation we get:

$$ {}^{r}\mathbf{A_nB_n} = \begin{bmatrix} -d_n * cos(\alpha_n + \beta_n) - b_n * sin(\alpha_n + \beta_n) \\ -d_n * sen(\alpha_n + \beta_n) + b_n * cos(\alpha_n + \beta_n) \end{bmatrix} * (\dot{\theta} + \dot{\beta_n}) $$

Next step is to project the above velocity ${}^{r}\mathbf{O\dot{B}_n}$ along the direction of the wheel plane and of the wheel axis. From geometric considerations we can see the unit vector of the wheel plane is:

$$ {}^{r}\mathbf{k_n} = \begin{bmatrix} \cos(\alpha_n + \beta_n - \frac{\pi}{2}) \\ sin(\alpha_n + \beta_n - \frac{\pi}{2}) \end{bmatrix} $$

While the unit vector of the wheel axis is:

$$ {}^{r}\mathbf{j_n} = \begin{bmatrix} -sin(\alpha_n + \beta_n - \frac{\pi}{2}) \\ cos(\alpha_n + \beta_n - \frac{\pi}{2}) \end{bmatrix} $$

And finally, after some manipulation, we can write the conditions each wheels imposes to the motion, namely the non-slip condition:

$$ \begin{bmatrix} cos(\alpha_n + \beta_n) & sin(\alpha_n + \beta_n) & d_n + l_n \cdot sin(\beta_n) \end{bmatrix} \cdot R(\theta) \cdot \begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{\theta} \end{bmatrix} + d_n \cdot \dot{\beta_n} = 0 $$

And the pure rolling condition:

$$ \begin{bmatrix} -sin(\alpha_n + \beta_n) & cos(\alpha_n + \beta_n) & b_n + l_n \cdot cos(\beta_n) \end{bmatrix} \cdot R(\theta) \cdot \begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{\theta} \end{bmatrix} + b_n \cdot \dot{\beta_n} + r_n \cdot \dot{\phi_n} = 0 $$

Examining the non-slip conditions it's possible to see that our robot will be holonomic, even if non-holonomic constraints are present that link the configuration variables to each other.

4.1.4 Whole model assembly

Using the above equations it's now possible to build to whole robot's model. In fact we can write 4 non-slip equations and 8 pure rolling conditions that are independent from each other. In addition we can define an input vector $\boldsymbol{\eta}$ for our robot, containing the velocities in the fixed frame of the point P and $\dot{\theta}$ namely:

$$ \boldsymbol{\eta} = \begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{\theta} \end{bmatrix} $$

This way we can rewrite all equations using the input vector components. Using the state vector above defined it leads to the following compact formulation for our kinematic model:

$$ \dot{\mathbf{q}} = S(\mathbf{q}) \cdot \boldsymbol{\eta} $$

Where S is a 15x3 matrix that can be written using the above mentioned equations. In case direct kinematic model is needed the present equation can be inverted computing Moore-Penrose pseudoinverse of S($\boldsymbol{q}$). From the derived model it is possible to obtain a model that uses accelerations at the inertial level as input simply deriving with respect to time:

$$ \ddot{\mathbf{q}} = S(\mathbf{q}) \cdot \dot{\boldsymbol{\eta}} + \dot{S}(\mathbf{q}) \cdot \boldsymbol{\eta} $$

4.2 Dynamic model

In this section the dynamic model of the ROPOD will be presented. Using the model, one can compute the evolution of the system given the wheels' torques inputs. To derive dynamic models for mobile robots two approaches are possible: the first one is the use of Newton-Euler equilibrium equations on the several bodies that constitute the robot, while the second involves the Lagrange equations, that derive the equations of motion from energy considerations. Here the second approach is followed. By definition the Lagrangian is :

$$L = T - U$$

Where T is the kinetic energy of the system and U its potential energy. Since our robot is moving in a plane environment, U is constant and so neglected. The kinetic energy of the robot can be written using the state vector defined in the previous section as:

$$T = \frac{1}{2}\dot{q}^T Q(q)\dot{q}$$

Where $Q(q)$ is the mass matrix of the system. Since non-holonomic constraints are present a generalization of Lagrange equations has to be used[22]. The Lagrange equations are then:

$$\frac{d}{dt}(\frac{\partial L}{\partial \dot{q}})^T - (\frac{\partial L}{\partial q})^T = C\tau + G(q)\iota$$

From which derive:

$$Q(q)\ddot{q} + f(q,\dot{q}) = C\tau + G(q)\lambda$$

Where C is a matrix that relates the input of the system, in this case each wheel's motor, to the generalized work on q degrees of freedom, $G(q)$ is the transpose of the kinematic constraints matrix, and ι is a vector of unknown lagrangian multipliers. The physical meaning of the term $G(q)\iota$ is a set of generalized forces that enforces the non-holonomic constraints. To delete the unknown lagrangian multipliers from the equations it is possible to transform the equations using the kinematic model previously derived, and pre-multiplying both side for the transpose of the kinematic model it follows:

$$S(q)^T Q(q)S(q)\dot{\eta} + S(q)^T Q(q)\dot{S(q)}\eta + S(q)^T f(q,\dot{q}) = S^T C\tau + S^T G(q)\iota$$

Noticing that for the nonslip conditions to be verified it holds, for every $\dot{q}$:

$$S(q)^T\dot{q} = 0$$

And using the kinematic model we get, for every η:

$$G(q)^T S(q)\eta = 0$$

Then the lagrangian multipliers term can be erased and the dynamic model becomes:

$$M(q)\dot{\eta} + o(q,\dot{q}) = S(q)^T C\tau$$

4.3 Energy consumption model

In order to evaluate energy consumption and also to be energy efficient using MPC, a model of the energy consumption of the ROPOD will be here derived.

A first approach to derive such model could have been the use of dynamic and kinematic models above obtained to compute the velocity and the torque of each actuated wheel starting from the trajectory of the chassis, and then apply the same model used for the 1D case, however this turned out to be not viable. In fact, considering those two models one can observe that, while it is possible to solve the inverse kinematics, since $S(q)$ is already an inverse kinematic model that relates the input generalized velocities to the state vector q, the same it's not possible for the dynamic model. Examining the model terms dimension it turns out that $S(q)^T C$ is a 15x3 matrix, meaning that the reduced dynamical model is over-actuated. Being so, there exists infinite possible pseudo-inverse matrices, and there is no reason one of them should be the one gives us the torques that are really applied to the system. A way to deal with this problem would be to model also the control system of the robot: then starting from the chassis trajectory it would be possible to compute the velocity references for the control system, whose output are torques that act on the system, but since our motion planner is meant to run online such model is believed to become too complex and to require too much computing time.

Another way is to change method completely and to use, instead of a first principles model, an identified one. These models are usually much simpler and they're easier to derive too but it is mandatory to identify what are the key phenomena in the process to model, thus requiring experience in that particular field of application. This is a popular approach in electric cars field [23] [24], since cars are systems that would be way too complex to model otherwise. Using this approach the energy consumed is equal to:

$$E = \int F v dt$$

Being F the force appleid to the center of mass of the car and v its speed. While the speed is easy to measure, the force acting on the center of mass is estimated as the sum of several terms, usually the inertia forces, the rolling resistance, the aerodynamic resistance and the force due to the slope of the ground are considered. Since the robot is supposed to be moving in a plane and velocities are very low compared to what is normal for a car, both aerodynamic resistance and gravity force can be neglected. The remaining terms can be modeled as follows:

$$F_r = \zeta m$$
$$F_i = \gamma m a$$

Where m is the overall mass of the robot, a its acceleration and ζ, and γ are parameters to tune through comparison with experimental data. The power consumed at each time instant will then be:

$$P_b = \begin{cases} (F_r + F_i)v & \text{if } (F_r + F_i)v > 0 \\ \eta(F_r + F_i)v & \text{if } (F_r + F_i)v < 0 \end{cases}$$

Given its simplicity this kind of model is well suited for online applications and so this was the chosen approach for the simulations.

4.4 2D environment navigation

In this section, starting from the formulations for the 1D problem, approaches for moving in a 2-D environment with obstacles are proposed. Also capabilities that are needed for real wolrd applications, such as obstacle avoidance, are included and their capability of navigating complex worlds are investigated.

4.4.1 Obstacle avoidance

Obstacle avoidance is the capability of a robot of avoiding obstacles while reaching its target and it is a key aspect of every navigation scheme that has to be applied in a real scenario.

There are several ways obstacle avoidance can be performed, for example the VHF motion planner [5] computes a polar histogram that tells the density of obstacles surrounding it and then steers the robot towards the direction less obstacles.

The DWA [6] motion planner uses a different approach. In this method a factor that accounts for the distance between the robot and its closest obstacle is added to the cost function that will be minimized. This way proved to be effective, however several shortcomings are present. For example, since the distance from the obstacle is maximized, in the surroundings of an obstacle the robot is pushed away from it. This can result in unnecessary oscillations and can prevent the robot from been able to go through paths between obstacles that are close to the robot dimensions. Also computing the distance between the robot and an obstacle can be very difficult for complex geometries. For these reasons they are usually simplified, but simplified shapes, like circles, can result in bigger space occupied than the real shape would.

For these reasons a different approach is tried in this thesis. This approach consists of using MPC predictions of the future to check, when evaluating a trajectory, whether a collision will happen in the future and to discard the trajectory under evaluation in case. To check if a collision occurred the robot real shape is discretized using a polygonal mesh. Then, using a grid map that in real applications will be estimated using the robot sensors, the coordinates of every node of the mesh are checked as part of an obstacle or not, if any positive match is found a collision with an obstacle is detected. To discard a trajectory, a penalty is given to its cost function, high enough to be sure it is not chosen. One advantage of this method is that complex shaped robots can be treated simply without requiring simplifying hypothesis, like considering a circle that includes the robot as its shape for collision check. Also, since no repulsion from obstacles is present, also very narrow paths should be available. One obvious disadvantage is its high computational cost, that increases the finer is the mesh to approximate the robot.

4.4.2 Direct formulation

In this formulation the decision variable to optimize at each step will be the acceleration at each time sample along the x and y axis of the robot. Since the robot is holonomic the orientation in the plane is not as important as it would be for a differential drive robot or another non-holonomic platform. In fact for these two the directions of motion available depend on the current orientation of the robot, while for an holonomic one every direction is available in every configuration.For this reason from now on the robot orientation is considered to remain constant, however formulations that include also commands on the orientation can be easily derived from the ones presented here.

The cost function that will be optimized is structured as follows:

$$J = J_1 + J_2$$

$$J_1 = \sum_{k=1}^{S} [C_1 P_b(k) dt + C_2 ||(P(k) - P_o)|| + C_3 \ddot{P}(k)^2 + \sigma_{obs} + \sigma_{speed}]$$

$$J_2 = \sum_{k=S+1}^{N} [C_4 (P(k) - P_o)^2 + \sigma_{obs} + \sigma_{speed}]$$

As it can be seen J_1 is the sum of five terms. The first is the instantaneus power consumed, or restored if breaking, at the k-th time sample times an arbitrary weight C_1, the second term is the euclidian distance between P,the actual robot position, and $P_o$the goal position weighted using the C_2 constant, the third one is the squared acceleration of the robot with his weight C_3, similarly to the second formulation of the 1D case. The last two terms enhance obstacle avoidance using the approach described in the previous section and set a soft constraint on the maximum speed of the robot. The second part of the cost function, J_2, is used to try to enforce the arrival time, giving up on the energy efficiency terms and having a driving term with a different weight, C_4 and the terms dealing with maximum speed and the obstacles.

Examining the navigation capabilities of this formulation in avoiding obstacles, it can be noticed that they are very limited. In fact looking at the simulation results in Fig. 4.4 , it can be seen that the robot moves only along the shortest path that connects the starting and goal positions. For this reason, if an obstacle is in front of it it doesn't hit it but is not able to circumvent it in a reasonable amount of time. Instead the robot moves towards the goal and stops as it path is blocked. Since such a limitation to the planner capabilities is reputed inadmissible, as it would required targets reachable with straight paths free of obstacles to be computed in advance to work well, another formulation is proposed to try to overcome these limitations.

4.4.3 Two-directions formulation

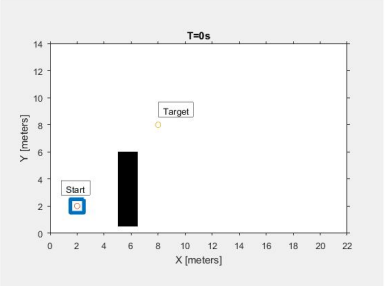
This formulation differs from all previous ones due to the fact that the cost function will no longer be one but two and at each time step only one of them will be optimized. The two cost functions will be:

$$J_x = J_{1x} + J_{2x}$$

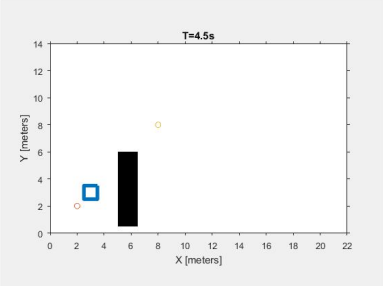
$$J_{1x} = \sum_{k=1}^{S} [C_{1x} P(k) dt + C_{2x} (x(k) - x_o)^2 + C_{3x} \ddot{x}(k)^2 + \sigma_{obs}]$$

$$J_{2x} = \sum_{m=S+1}^{N} [C_{4x} (x(k) - x_o)^2 + \sigma_{obs}]$$

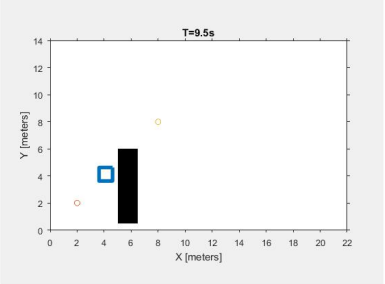
And:



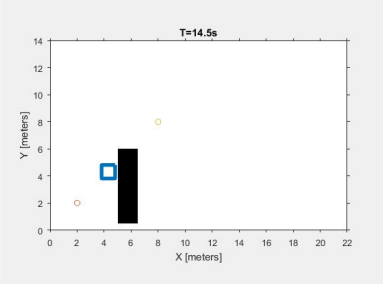
(a)



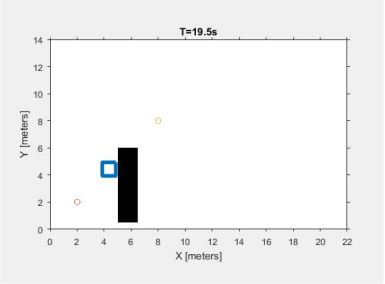
(b)



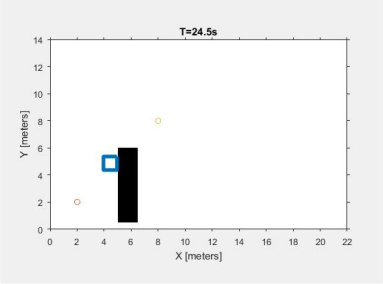
(c)



(d)



(e)



(f)

Figure 4.4: In this figures the navigation capabilities of formulation n.1 are shown, every picture is a frame at a different time of the simulation. The robot starts from its position but it's not able to overcome the obstacle in front of it.

$$J_y = J_{1y} + J_{2y}$$

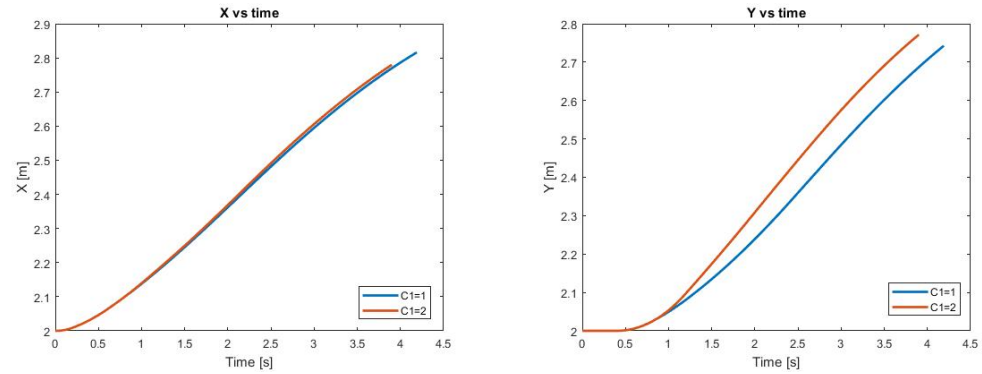
$$J_{1y} = \sum_{k=1}^{S} [C_{1y}P(k)dt + C_{2y}(y(k) - y_o)^2 + C_{3y}\ddot{y}(k)^2 + \sigma_{obs}]$$

$$J_{2y} = \sum_{m=S+1}^{N} [C_{4y}(y(k) - y_o)^2 + \sigma_{obs}]$$

Looking at the two cost functions they have similar structure, and they are similar to the formulation proposed in section 3.3.2. Each of the two only tries to minimize the distance between the robot and the target in one direction, instead of using the euclidean distance and they use the acceleration in the direction they are minimizing the distance as decision variable, because at the end of section 3.3.3 it was found that this kind of formulation is more suited for online applications. At each time step one of the two is optimized, using the output of the previous optimization to know the acceleration in the other direction, to compute the evolution of the system, and the output to the system is the composition of the first term of each sequence. At the next time step the function to optimize is switched, so that one time is optimized along the x direction of the robot and the next one along the y direction of the robot, these directions being chosen while deriving the kinematic model of the ROPOD. The effect of each term of the cost functions is similar to the 1D case and also in this case an increase in the C_1 coefficients can lead to more energy-saving trajectories as can be seen in Fig 4.5.

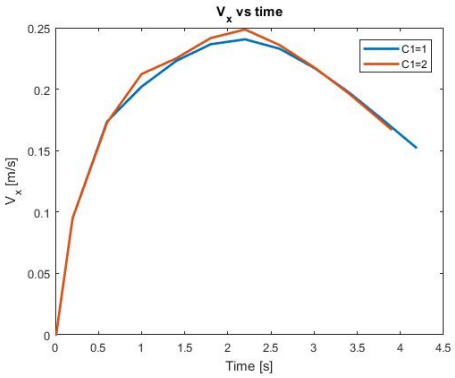
Repeating a simulation similar to the one done for the previous formulation it can be seen in Fig 4.6 that the obstacle is successfully overcome.

Considering that the cost functions that are used, further considerations on the ability of the motion planner of circumventing obstacles can be made. In fact, since the driving term tries to minimize the distance from the target along one specific direction, it's easy to see that the algorithm will easily overcome all those obstacles that doesn't require that distance to increase. For example squares will be always overtook if the target projection, along the directions in which the optimization is done, is outside the square's sides. Also circles will be surpassed if the target is outside the circumscribed square and the same condition as before is matched. Another limit to the navigation capabilities of this formulation is that the directions over which the optimization takes places are arbitrary, since they are referred to the robot and not to the environment surrounding the robot. Another possible way of fixing them could make use of the position of the target, equally spacing the angles between these directions and the direction that points towards the target, however it doesn't guarantee better performances. In the end, considering that the planner takes into account only a short time horizon in the future and the possibility for the robot to end up stuck somewhere, it is clear that the proposed motion planner is a local one. For general shapes and complex environments there is no guarantee of reaching the goal so additional components are required to find the path in case the robot is stuck, so that an hybrid planner could be made, composed of the local MPC planner and an higher-level one. One possible addition is a probabilistic planner, from the RRT [4] family, that finds a sequence of sub-goals capable of leading the robot to its target. In case the computing cost and the resulting random path are not acceptable other ways are available, for example a sub-goal can be computed directly looking at the shape of the obstacle in front of the robot, at the price of sacrificing the completeness of the addition. To conclude the presence of a moving obstacle near the robot was considered. In Fig. 4.7 one can see that the robot, instead of going straight to the target deviates to avoid a moving obstacle. This result was obtained using the same strategy used for fixed obstacles with only one difference. The obstacle avoidance formulation

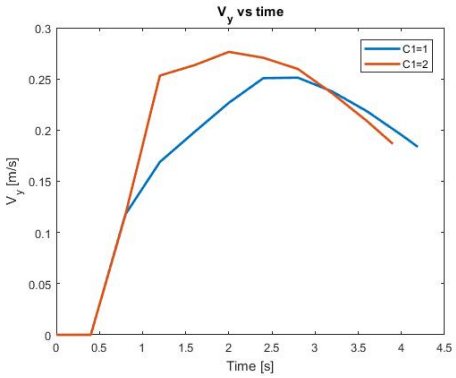


(a)

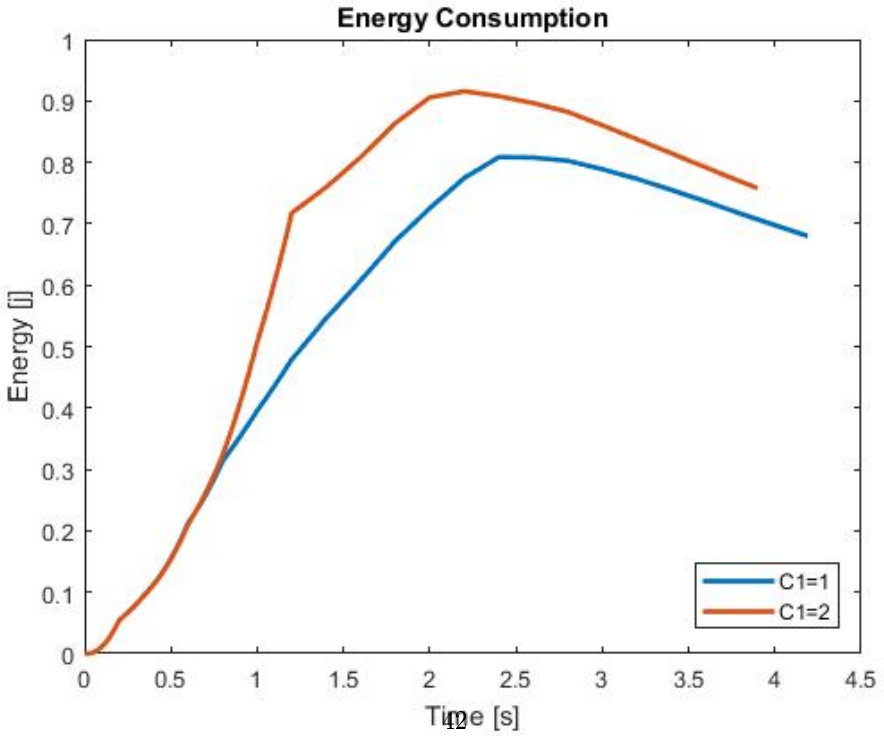
(b)



(c)

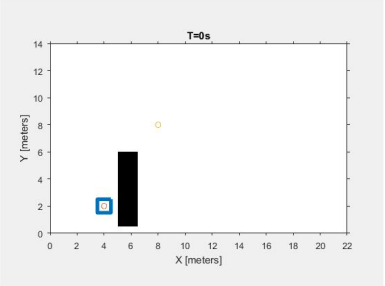


(d)

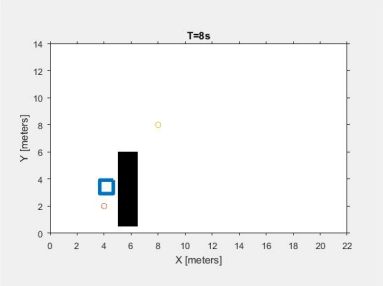


(e)

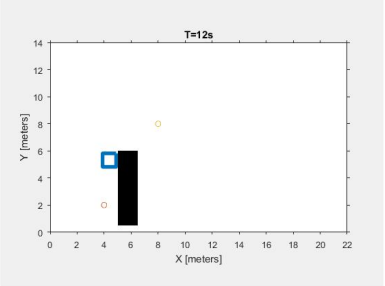
Figure 4.5: Effect of increasing the C_1 value in the two directions formulation. The other parameters of the simulation were: C_1=10, C_3=0,C_4=100, ϵ=0.8,dt=0.2 and N=20



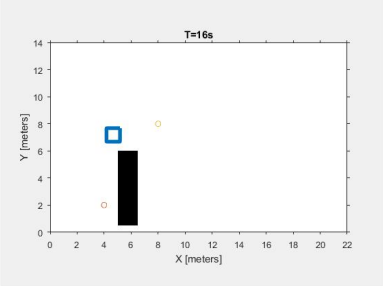
(a)



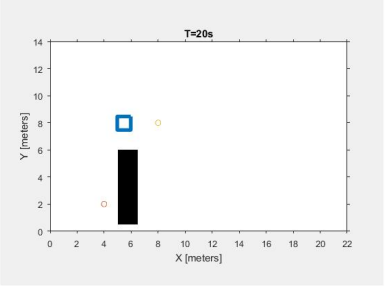
(b)



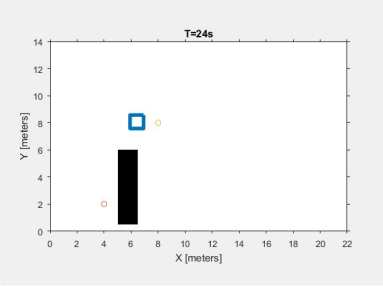
(c)



(d)



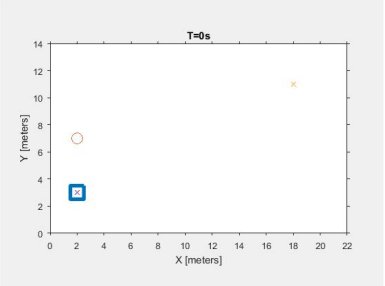
(e)



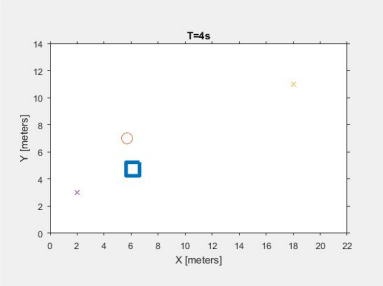
(f)

Figure 4.6: In this figures the navigation capabilities of formulation n.2 are shown, every picture is a frame at a different time of the simulation. The robot starts from its position and it's able to overcome the obstacle in front of it.

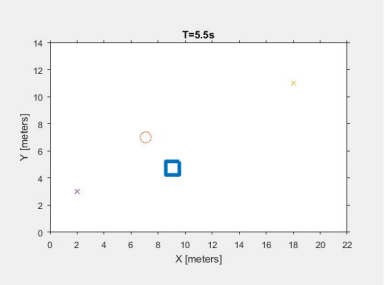
that was used in this work is based on the capability of the MPC of predicting where the robot will be in the future. The position of the robot can be evaluated using the models derived and the fixed obstacles are fixed in the space but the moving obstacles change their pose. In order to be more effective the planner has to know where these obstacles will be, otherwise it may discard trajectories that are actually safe and choose others that in the future will prove to have worst performances. To have a good estimation of the future of the moving obstacles proper models should be used. In this case the obstacle was supposed to keep its velocity constant. Another important factor is that, in order to be able to react to the moving obstacles in case their movements differs from the prevision, a reasonably high sampling frequency for the motion planning has to be guaranteed. The threshold depends on the application, particularly on the velocities of the robot and the obstacles and on the degree of uncertainty of the model used to predict the obstacles trajectories.



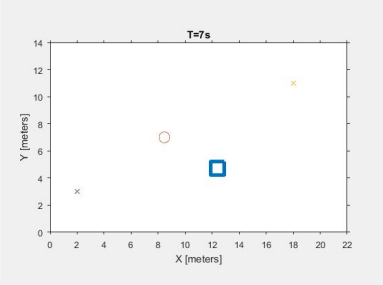
(a)



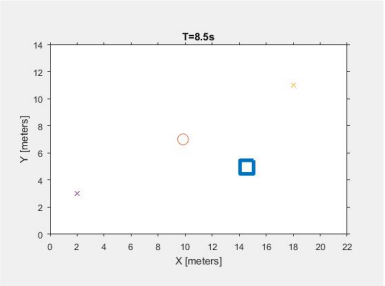
(b)



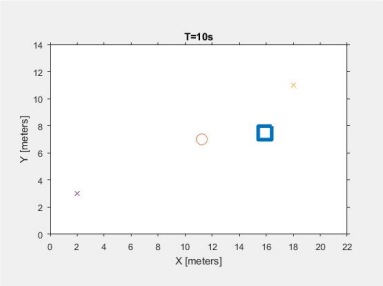
(c)



(d)



(e)



(f)

Figure 4.7: In this figures one can see the robot deviating from the straight path that links it to the target to avoid a moving obstacle.

Chapter 5

Experimental results

In this chapter the experiments carried out to prove the effectiveness of the 1D motion planning scheme proposed in section 3.3.2 are described and the outcomes are commented.

Firstly a description of the robotic platform used, the PICO robot, is presented from both a kinematic and a control point of view. Then energy consumption model proposed in the 2D case will be adapted to the robot, the parameters of the model will be tuned so that they can be used in the motion planner. Lastly the obtained experimental results will be shown and commented.

5.1 The PICO robot

Since by the time of the conclusion of the present work the ROPOD robot was not ready for testing, another omni-directional robotic platform, the PICO robot, was used during the experimental tests. A picture of the robot can be seen in Fig. 5.1.

Figure 5.1: The PICO robotr[28]

The kinematic structure of the robot is composed of 3 actuated Omni wheels equally spaced along the perimeter, an example of which can be seen in fig. 5.2. This type of wheel is made of a wheel at the perimeter of which a large number or small rollers is disposed with their rotation axes perpendicular to the axis of the wheel. This structure, although less robust than a conventional

Figure 5.2: An omni wheel, by Cbteam234 at English Wikipedia

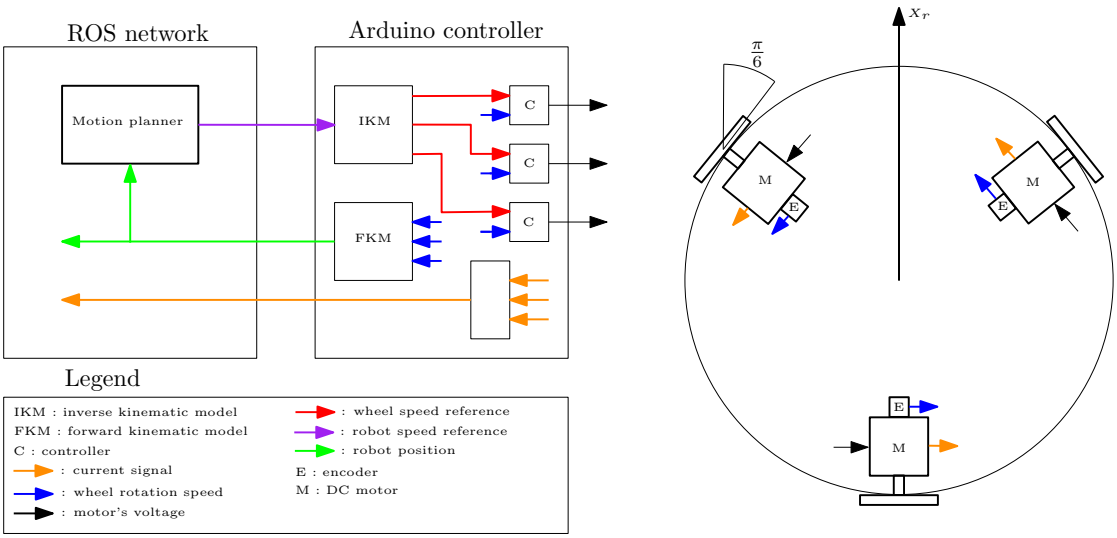


Figure 5.3: A scheme of the PICO robot control architecture

wheel, is allowed to move along the wheel axis, thanks to the rollers.As a result they do not create any non-holonomic constraint on the robotic platform, so that it is omni-directional, just as the ROPOD platform was meant to be.

From the control point of view there are several layers. In order to better understand the composition of the control part of the PICO robot, let's refer to fig. 5.3, where a scheme of the control architecture of the PICO robot is presented.

At the lower level we can see that each wheel is actuated by an electric motor, that is controlled at low-level by an Arduino micro-controller. This controller receives velocity commands for the whole robot from the higher level of control listening to the proper ROS topic, solves the inverse kinematic problem so that the required speed of the robot is converted into references for each wheel and then uses a close loop controller for each motor, closing the loop using the encoders that are present on the wheels.

At a higher level there is the motion planner, that acquires information using the sensors and sends velocity commands to the robot. During the testing the motion planner will be the motion planner proposed in section 3.3.2, with the two main differences. The first is the energy consumption model used to predict the energy consumed to go along a trajectory, that will be presented in the next section, as i this case this model has to relay on the sensors' information. The second difference is that in this case the command send to the robot will no longer be the first acceleration of the optimized trajectory but instead the velocity at the next time instant, as this control architecture is so that only velocity commands are available. In order to function

properly the motion planner will need to acquire information to use for the planning,so it will need to communicate with the robot's sensors and the Arduino controller.

The linking between the lower level controller and the higher level sensors and modules, such as the motion planner, is done via the robot operating system (ROS). This software, that runs on a UNIX system, provides a unified protocol for communication between the parts of the robot and it is structured as follows: every component is seen as a node of a network, at the top level of which there is the ROS master node that coordinates them all, and can exchange information with the other modules. The communication is done using messages that are published on specific topics. Each topic has one module that is the publisher, the node that sends the information, and can have several subscribers, that are the nodes that receive the information. A list of all valid topics of a ROS network, each with a unique name to identify it, is kept by the ROS master node, along with the structure of the message of each topic. The advantages of such a solution is that being an open-source platform is free to use and that a vast number of packages for different purposes are already available and ready-to-use. Also since the only communication is done via messages the different models can be written in different languages.

In the PICO robot ROS network several topics were present, for example those where the high level sensors, like the camera on top of the robot and the laser sensors mounted in the front part of the robot published their information. Only a few of these topics were actually used during the testing and they link the top level of the control architecture, the motion planner, to the Arduino controller. These topics can be seen the scheme of fig. 5.3. We have two topics published from the Arduino controller, that are the topic for the measurement of the currents in the motors and the topic that contains the actual position of the robot, computed solving the direct kinematic problem using the encoders' readings. Then there is the topic that is published from the motion planning node, that is listened from the Arduino controller, containing the speed required of the robot at every instant.

Given this control structure, that relays on a ROS network, the motion planner proposed in section 3.3.2 had to be converted into a ROS node to function properly and this was done using the MATLAB robotics toolbox.

5.2 Energy consumtion models

In order to try to minimize the energy consumed along a trajectory and also to compute the effective energy consumption of the robot while not having access to the battery,proper energy consumption models are needed. In this experimental part two models were used.

The first one was used for estimating the energy consumption during the trajectory optimization from the MPC motion planner and it is analogous to the one proposed in section 4.3:

$$E = \int (F_r + F_i) v dt$$
$$F_r = \beta$$
$$F_i = ma$$

where E is the energy consumed, m is the mass of the robot, v is its speed, a its acceleration and β a parameter that needs to be tuned.

The second one was used to tune the parameters of the first model and to compute the effective energy consumption of the robot while post-processing the acquired data. In this model we use the data acquired from the current sensors of the motors. Given the current i in a DC motor in fact the torque T exerted can be written as :

Table 5.1: Least square regression results for β estimation

Robot constant speed [m/s]	Slope [J/s]	β [N]
0.1	1.42	14.2
0.15	2.08	13.9
0.2	2.94	14.7

$$T = \tau K_t i$$

where τ is the reduction gear ratio and K_t is the torque constant of the motor. Both these parameters can be found in the motor's datasheet. Knowing the torque exerted from each wheel, the wheel's radius and their orientation with respect to the direction of motion it is possible to compute the total force actuating the robot in the direction of motion, that is the X_r direction in fig. 5.3, as:

$$F_t = 2sin(\frac{\pi}{6})\frac{T}{r}$$

That is the double of the force exerted on the ground from one wheel as there are two wheels that contribute to move the robot in the X_r direction, and they are symmetric with respect to the X_r axis.

The consumed energy for this second model can be written as:

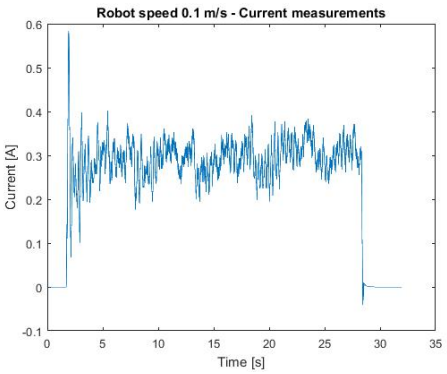
$$E = \int F_t v dt$$

In order to tune the parameter β of the first model several constant speed tests were conducted so that the inertial forces could be neglected. The readings of the current sensors, that were published from the Arduino controller on a specific topic, and the consumed energy computed using the second energy consumption model for each test can be seen in 5.4.

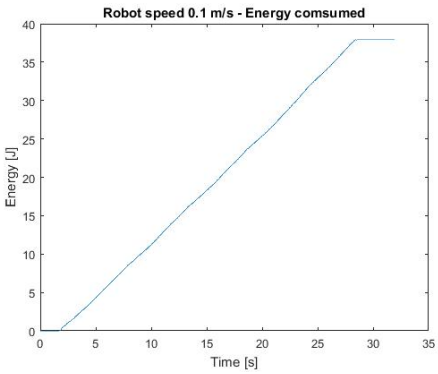
Using these measurements it is possible to estimate the parameter β of the first model. Due to the fact that the noise in the current measures appears to be relevant the fitting was done using the energy values in the constant speed part of the of the tests. For each test a least-square regression was made using a linear model to fit the data. The results of the regressions can be seen in table . Out of the three values a medium of β equal to 14.3 N will be used in the motion planner.

5.3 Results

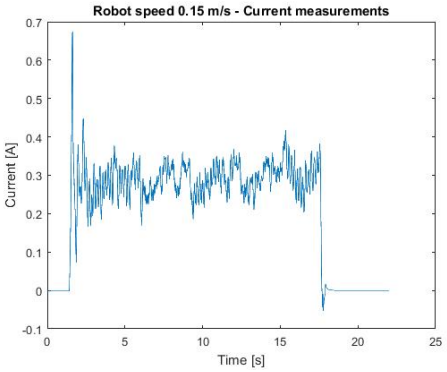
Using the motion planner described in section 3.3.2 and the first energy consumption model presented in the previous section to estimate online the energy consumption along a trajectory, several one-dimensional tests on the PICO robot were carried out, where the PICO was asked to reach the target starting from a standing status. Among the tests the weight of the different parts of the cost function was changed, so that the effects of each term could be seen. During each test the outcomes of the currents topic and of the position topic were recorded, listening to



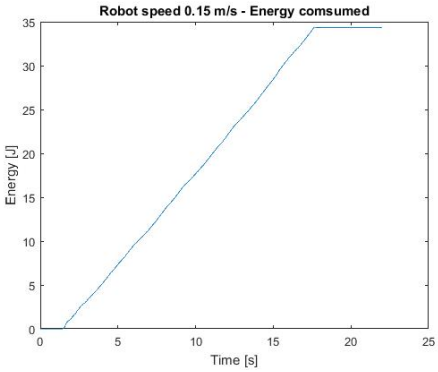
(a)



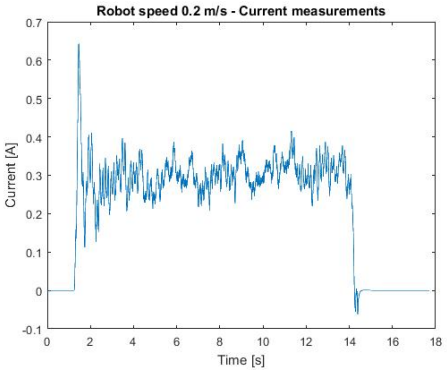
(b)



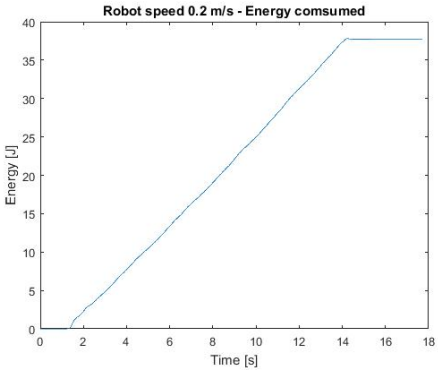
(c)



(d)



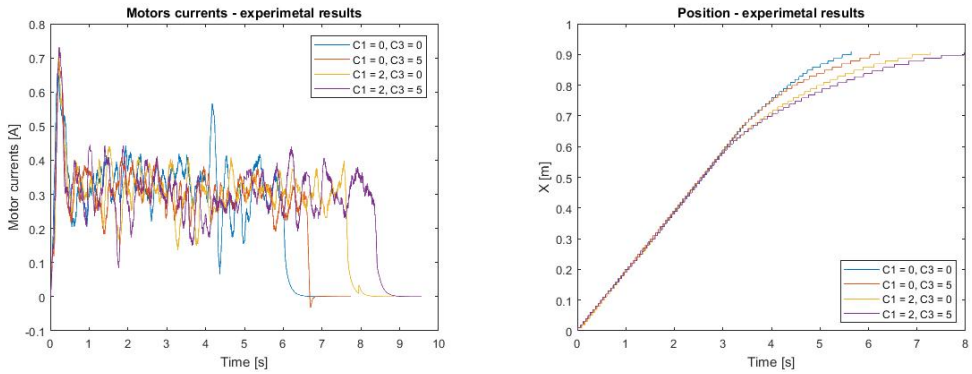
(e)



(f)

Figure 5.4: In these figures the results of the tests for tuning the β parameter of the first model are shown

.



(a) (b)

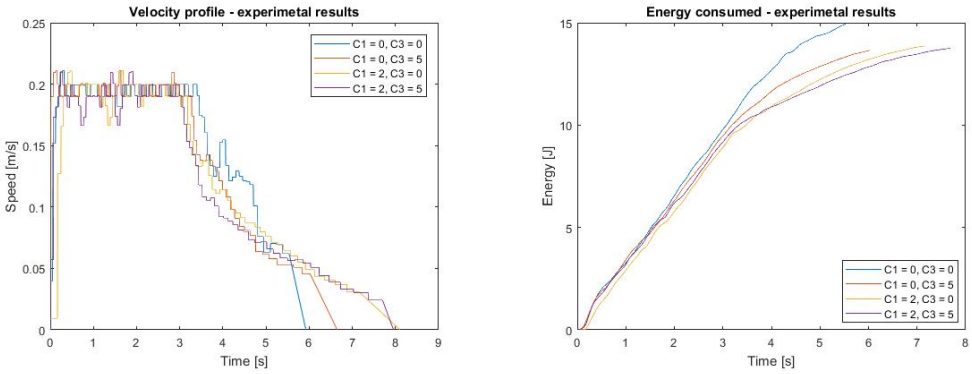
Figure 5.5: Currents and position of the PICO robot vs time during different tests

the proper ROS topics, in different ROS bags, from which the data could be then retrieved and post-processed.

In fig 5.5 a summary of the recorded data can be seen. As it can be seen four tests are present, all of them having the same time step and length of the prediction horizon, as well as the same weight for the term driving the robot towards its goal. Also the goal is obviously the same and so it is the set end time. What changes among the tests shown is the weight given to the parts of the cost function dealing with the energy consumption, weighted using the constant C1, and with the acceleration of the robot, weighed using the constant C3. An higher value of the constants means a greater importance of that parts in the total cost function. Looking at the position vs time plots it is possible to see that at the beginning of the test the behavior is almost the same for the different cases, while close to the end the evolution in time changes for the different cases. This can be explained if we consider that the total cost function to minimize is the sum of different terms, one of them being proportional to the squared distance from the target. It appears clear then, that at the beginning this part of the cost function is so much more important of the others two that the optimal trajectory minimizes this term first, while when the robot is closer to the final position this term is lower so that the difference in the other two terms can lead to different evolutions of the system. It can also be seen that an increase in the parts of the cost function accounting for the energy consumption gives as a result an increase in the arrival time of the robot, which looks reasonable and it is expected to lead to lower energy consumption.

In order to use the collected data and the second energy consumption model of the previous section to evaluate the energy consumed from the robot during the different tests the velocity profile of the robot, that couldn't be measured during the tests, needs to be estimated. In order to do so a direct numerical derivation of the position profile is not possible, because the position profile is made of several constant parts arranged as a stair, so a numerical differentiation would lead to a zero valued velocity profile with a spike every time the position value is updated. For this reason the position profile is first transformed into an linear interpolated one, and in order not to over-estimate or under-estimate the position the medium point of each constant step are chosen as interpolation points, and then, since the update rate in the position data is at a different step from the current profile the obtained linear interpolated position profile is sampled at the correct time step. This sampled profile is then numerically derived to obtain the velocity profile, that will be composed of constant steps since the interpolation of the position data was

linear. Using this velocity profile the energy consumption for each test can be computed, and the results of these operations can be seen in fig 5.6, where the obtained velocity and energy consumption profiles of the different tests are shown.



(a) (b)

Figure 5.6: Velocity and energy consumed of the PICO robot vs time during different tests

As it can be seen the behavior of the energy consumption is somewhat similar to the one of the position, in the sense that at the beginning the effect of the terms of the cost function dealing with energy consumption is negligible and the evolution is very similar for the different cases, while towards the end their presence can be clearly seen and as expected the test where no energy consumption at all was considered, so both C1 and C3 equal to zero, consumes more energy than the other cases.

Chapter 6

Conclusions and future work

In this thesis, starting from the need of the ROPOD robot for energy efficiency, a motion planner capable of satisfying this requisite has been developed, while dealing on several constraints, that varies from the arrival time on the target to the presence of obstacles surrounding the robot.

The chosen approach was the use of model predictive control for trajectories generation, as this method allows to optimize a cost function to derive the present action of the robot, and so , given an estimation of the energy consumption computed using a suitable model, it is possible to reduce it.

Firstly, a simple robot in a one-dimensional problem was considered to prove, through simulation, the viability of the method and to better understand the influence of the different factors, such as length of the time horizon, time step, numbers of iterations for the optimization. Several formulation of the problem were considered and compared.

Then the 2D problem of moving in a planar environment was considered. Kinematics and dynamics models of the ROPOD were derived to compute the evolution of the system. These models however turned out to be not well suited for the creation of an energy consumption model due to the fact that the system is over-actuated. For this reason an identified model was used to estimate energy consumption. During navigation in the environment, both fixed and moving obstacles surrounding the robot were taken into account, and the capability of the proposed formulations, that were based on the most suitable formulation of the 1D case, of the motion planner to reach the target were explored.

Finally, the proposed approach was tested using a real robot, the PICO robot, and the adaptation of the proposed approach to this robot as well as the obtained results were described.

As a result, it has been seen that the proposed approaches in the 1D case were able to drive the target at the desired location in a energy efficient way, at the price of a non-smooth profile as output of the planning phase. The reasons for this are several, the more important being not enough iterations in the optimization and the size of the time step. Also careful balancing between the different costs of the cost functions was needed, and there is no guarantee about good performances in every situation, given a set of parameters for the cost function.

In the 2D case, the second proposed formulation showed it can increase energy efficiency, however it inherits the disadvantages that were already present in the 1D case. On the front of the obstacle avoidance the proposed approach proved to be able to overcome some obstacles but to end up stuck if to circumvent them it was needed to increase the distance between the robot and the obstacle. Several ways to avoid it were proposed, and it become clear that to navigate more complex environment a coupling with higher level algorithms is required using the cost function formulations proposed.

Lastly, using the results of the tests on the PICO robot it was shown that an increase in the energy-accounting terms of the cost function can result in a decrease of the energy consumed.

One key aspect that hasn't been sufficiently explored and that should be part of the future work to improve what was done here is to lower the computing time needed to run the optimization. This is very important because to run online in the real world the frequency of the motion planner should be high enough to avoid moving obstacles, and also a lower time step allows better results. One way to significantly reduce the computing time required is the translation of the motion planner from the Matlab language used in this work to a faster one, like C.

Another aspect that would required further work is the experimental part, as only the one dimensional part was tested, so the 2D case both without and with obstacles around the robot is yet to be tested. Also it would be important to conduct these tests on the actual ROPOD for which the motion planning schemes proposed in this thesis were developed.

Bibliography

[1] B. Siciliano, O. Khatib, *Springer handbook of robotics* , Springer, 2016

[2] M. Hoy, A.S. Maatveev, A.V. Savkin, *Algorithms for collision-free navigation of mobile robots in complex cluttered environments: a survey* , Cambridge University Press , 2014

[3] P. Hart, N. Nilsson , B. Raphael *A Formal Basis for the Heuristic Determination of Minimum Cost Paths* , 1968

[4] M. LaValle, *Rapidly-exploring random trees: A new tool for path planning* , 1998

[5] J. Borenstein, Y.Koren, *The Vector Field Histogram - Fast Obstacle Avoidance for Mobile Robots* , 1991

[6] D. Fox, W. Burgard, S. Thrun , *The dynamic window approach to collision avoidance*, IEEE , 1997

[7] S. Quinlan, O. Khatib, *Elastic Bands: Connecting Path Planning and Control*

[8] S.G. Tzafestas, *Introduction to Mobile Robots Control*, Elsevier Insights

[9] A.Barili, M.Ceresa, C.Parisi, *Energy-Saving Motion Control For an Autonomous Mobile Robot*

[10] Y.Mei, Y.Lu, Y.Charlie Hu, C.S. George Lee, *Energy Efficient Motion Planning for Mobile Robots*, IEEE , 2004

[11] C.H. Kim, B.K. Kim , *Minimum-Energy Motion Planning for Differential-Driven Wheeled Mobile Robots* In: *Mobile Robots Motion Planning, New Challenges*, pp. 193-226, In-Tech

[12] S.Liu, D. Sun, *Minimizing Energy Consumption of Wheeled Mobile Robots via Optimal Motion Planning*, IEEE, 2014

[13] C. Henkel, A.Bubeck, W.Xu, *Energy Efficient Dynamic Window Approach for Local Path Planning in Mobile Service Robots* , 2016

[14] J.B. Rawlings, D.Q. Mayne, *Model Predictive Control: Theory and Design*, Nob Hill

[15] L. Grune, J. Pannek, *Nonlinear Model Predictive Control Theory and Algorithms*, Springer, 2011

[16] Fitri Yakub, *Model Predictive Control for Car Vehicle Dynamics System - Comparative Study*, 2013l

[17] A.Domahidi, A.U. Zgraggen, M.N. Zeilinger, M. Morari, C.N. Jones, *Efficient Interior Point Methods for Multistage Problems Arising in Receding Horizon Control*

[18] M.N. Zeilinger , *Real-time Model Predictive Control*

[19] M.M.G. Ardakani, B. Olofsson, A. Robertsson, R. Johansson, *Real-Time Trajectory Generation using Model Predictive Control*, IEEE , 2015

[20] L.I. Galindez Olascoaga, *Model Predictive Motion Planning for Robots*, 2015

[21] L. Pantieri, T. Gordini, *L'arte di scrivere in Latex*, 2012

[22] B. Siciliano, L. Sciavicco, L. Villani, G. Oriolo, *Robotics Modeling, Planning and Control*, Springer

[23] X. Wu, A. Cabrera, D. Freese, W.A. Kitch, X. Jia , *Electric Vehicles' Energy Consumption Measurement and Estimation*, 2015

[24] N. Chang, D. Baek, J. Hong, *Power Consumption Characterization, Modeling and Estimation of Electric Vehicles*, IEEE, 2014

[25] www.irobot.it

[26] www.fortune.com

[27] www.siba.at

[28] www.cstwiki.wtb.tue.nl