UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Receding Horizon Control of Multiagents Systems with Competitive Dynamics

Candidate: Andrea Carron

Advisor: Prof. Luca Schenato
Advisor: Prof. Elisa Franco

Master in Control Engineering
Department of Information Engineering
2012

To my parents

"*Questo controllo è un pò nervoso...*"

Abstract

We consider the problem of controlling two agents with competitive objectives. Agents are modelled as linear discrete time systems, and collect each other's state information without delays. The competitive problem is formulated in a classical receding horizon framework, where each agent's controllers are computed by minimizing a linear, quadratic cost function which depends on both agents' states. The two agents specify their state tracking objective in a coordinated or competitive manner. We do not consider state constraints. The simplicity of our framework allows us to provide the following results *analytically*: 1) When agents compete, their states converge to an equilibrium trajectory where the steady state tracking error is finite. 2) Limit-cycles cannot occur. Numerical simulations and experiments done with a LEGO mindstorm multiagent platform match our analytical results.

Contents

1 Introduction

Multiagent systems are systems composed of multiple interactive elements that are called agents. The agents are capable of autonomous actions in order to achieve their objectives and are able to interact with the other agents. The goal of the communication is to allow the cooperation or the coordination between the agents.
The cooperation of multiagent systems has token a central role in the control community in the last twenty years. The complexity of the systems has required to develop advanced algorithms in order to solve that kind of problems, for example the coordination of multirobot platform where is impossible use a central unit, or in the electric networks that are so wide that cannot be controlled in a centralized way.
Another important aspect is the competition, where several agents try to get what just some of them can get. This thesis analyses the competition in a multiagent system where the single agent use a Receding Horizon Control technique.
Receding Horizon Control (RHC) was developed during the 1960s, when its application was limited by the reduced computational capabilities of the period. It attracted the interests of the control community twenty years later, with theory generalizations [10], [9] and application of RHC in industrial proccesses [19]. RHC is used especially for MIMO systems with slow dynamics, where it is more easily tunable than a PID controller. For this specific reason the application area of the RHC is really wide, such as: chemical plants [22], supply chain management [12], control of hybrid vehicles [2], automotive [4] and aerospace

application [14].
Modern applications of RHC include control of human crowds [21] (where the model is based on Mixed Logical Dynamical (MLD) systems) and motion control of biped robots [1]. These applications are inspired by the algorithmic prediction similarity between RHC and the human mind, where, loosely speaking, control optimization/evaluation over a finite-time horizon occurs iteratively including updated information about the environment.

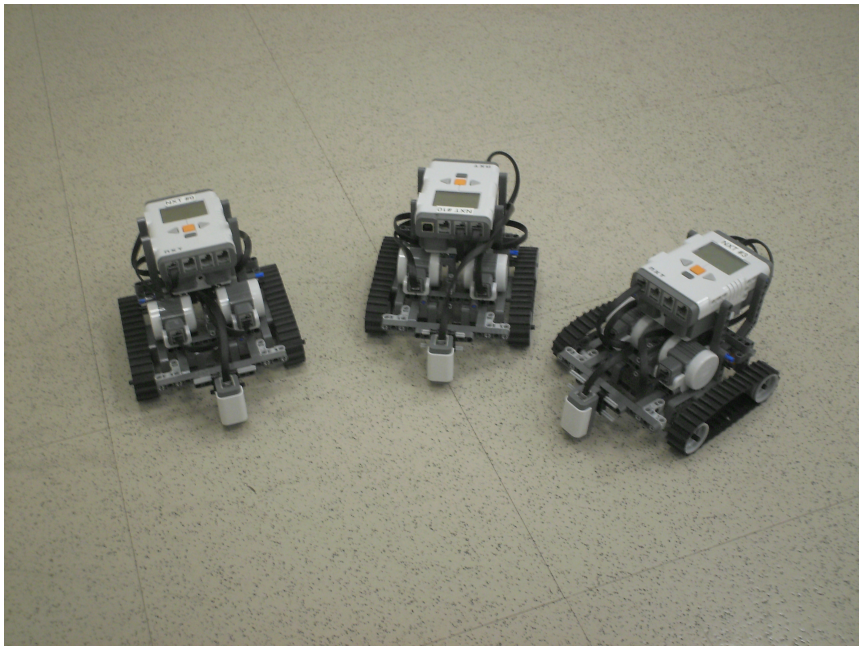

Figure 1.1: The LEGO robots used in the experiments.

Many distributed versions of RHC have been proposed in the context of multiagent, cooperative systems, where agents include local and team information to perform tasks and maintain stability. Some of the most important works in this ares are [7], where the goal is to achieve coordination among agents that are solving model predictive control (MPC) problems with locally relevant variables, costs and constraints; and [11, 13], which considers stabilization of multi-vehicle formations. To our knowledge, RHC has not been applied to competitive multiagent systems, where agents have conflicting objectives. Competitive multiagent systems have been studied in the past using probabilistic approaches based on game theory. Many interesting results are available for pursuit-evasion games: for example, [15] analyzes a greedy policy to control a swarm of agents in the pursuit of one or more evaders, demonstrating that this policy guarantees to find evaders in a finite time. Another important reference on pursuit-evasion games is [25], where two different greedy polices are considered, and all agents concurrently build a map of the unknown environment. Finally, a

"hide and seek" game was studied in [5], again in a probabilistic framework.
We believe that RHC might provide a useful "algorithmic framework" to study competitive multiagent systems. We are inspired by a simple problem with two adversary agents: one agent is a pursuer, seeking to reach a given neighborhood of the second agent's state; the second agent is an evader, seeking to maintain a safety distance from the pursuer. Suppose each agent uses an RHC-like algorithm to reach its objective, repeatedly updating its future decisions (within a certain time horizon) based on the moves of its adversary. What is the class of dynamics that can arise?
We provide a simple analytical treatment of this problem: we assume that the two agents are discrete time linear systems (LTI) without internal dynamics, and each agent optimizes a linear, quadratic cost function in a RHC fashion. The cost function depends on the state of each agent and its adversary, and competitive objectives are simply defined as a conflicting distance tracking offset between the two agents. In the absence of state or input constraints, we analytically show that if their objectives are conflicting, the two agents reach an equilibrium trajectory along a line, with a finite tracking error at steady state. We also show that limit cycles are not possible.
We verified our results with numerical simulations, and we used a LEGO Mindstorm robot kit [8] to create a two-agent testbed implementing our simple case study. Our experimental results match extremely well our predictions. Thus, this thisis contributes a) Novel analytical results in a simple RHC competitive/cooperative system, b) An RHC implementation benchmark, based on a commercial robot kit, which we believe is valuable for educational purposes.

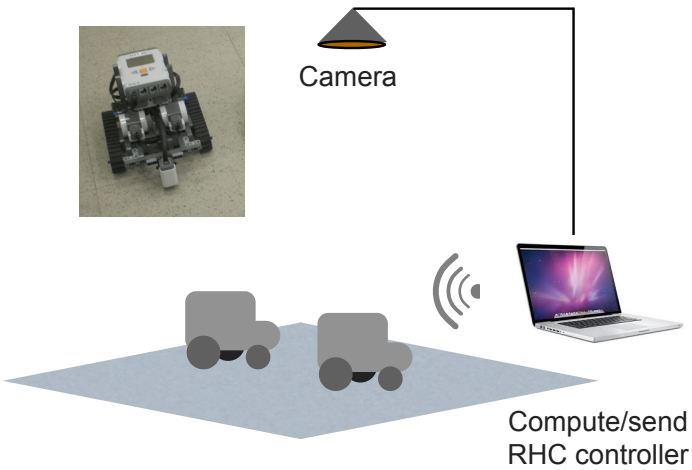

Figure 1.2: Scheme of our experimental LEGO robotic platform.

This thesis is divided in four chapters, in the first is studied the tracking of a trajectory. To complete the task the agent predicts the trajectory with a simple autoregressive model, based on the old and known samples, and use the data obtained to compute the next input. To solve the problem at the beginning we assume to know the actual trajectory value without delays and in a second step we suppose to know it with one step delay. The simulations show that the average performances become worse when then knowledge of the actual position of the object to follow is delayed. The problem developed in a stochastic framework is brought back to the standard deterministic problem of the receding horizon.
In the third chapter we consider that both the agents have an "intelligence" and both are controlled with a receding horizon control technique. We assume that the states vectors and the inputs vectors don't have any kind of constraints. The main result proofs that, in case of competitive targets (for example the first agent has to reach the other and the second has to keep a safety threshold from the pursuer), the two agents converge over a line and just in case of cooperative targets they are able to complete the tasks and the steady state position is fixed. After a short section about the problem formulation, there are the main theoretical results and the validation given by the simulations.
In the fourth chapter are explained the experimental results, there is also an overview of the experimental setup and of the software architecture, here there is also a section that shows how the main software issues encountered were solved. Then the results are shown and analysed.

2

Tracking with Receding Horizon

In this section is explained how track a trajectory using the receding horizon control. The idea is to predict the trajectory using a simple autoregressive model (with the past trajectory samples) in order to have a rough estimation of the future positions. The framework is stochastic and is shown how bring it back to a deterministic problem, it permits to use the well known theory of the RHC to guarantee the stability of the system. Moreover we try to understand how the delays influence the tracking performances.

2.1 Problem Formulation

In this section, the system dynamic is defined. We make use of the following notation, for any vector $x \in \mathbb{R}^n$, $\|x\|_P^2$ denotes the P-weighted norm, given by $\|x\|_P^2 = x^T P x$, and P is any positive definite real symmetric matrix. We call with the vector s_t the state of the pursuer agent at the time t, and with r_t the state of the trajectory to follow always at time t. We assume that all the states vectors don't have any kind of constraints and also the control vector of the agent is not subject to any constraint.
The discrite-time invariant linear dynamics of the agent to control is given by:

$$\begin{aligned} s_{t+1} &= F_s s_t + G_s u_t \\ y_t^s &= H_s s_t + v_t \end{aligned} \tag{2.1}$$

Where F_s, G_s, H_s are the state space matrices, where $H_s = \mathbb{I}$, y_t^s is the known state vector, that is equal to the real state vector plus a Gaussian noise given by the signal $v_t \sim N(0, \sigma^2)$, where σ^2 is the variance of the noise.

Algorithm 2.1.1. *The steps to follow at any time instant in order to complete the task are:*

1. *From the past data of the trajectory to pursue the parameters of an autoregressive model are estimated.*

2. *Given the model, the values of the trajectory to follow form 1 to k steps ahead is predicted.*

3. *A quadratic cost function, that depends by the agent actual position, the control and the trajectory to reach, is minimized in respect of the control.*

4. *Just the first control step is applied and the sequence start again.*

Now all the steps are analysed more in depth.

2.2 Autoregressive Model Estimation

The problem of estimate the autoregressive model parameters can be brought back to the least square problem. Let consider an autoregressive model of order n:

$$\begin{aligned} &y_{t+1} + a_0 y_t + a_1 y_{t-1} + \ldots + a_{n-1} y_{t-n-1} = e_{t+1} \\ &A(z^{-1}) y_{t+1} = e_{t+1} \end{aligned}$$

Where y_t are the measurements, e_t is a Gaussian noise with finite variance and zero mean that drives the model and a_i are the coefficients to estimate. Rewriting the equation:

$$\begin{aligned} &\varphi_t^T = [-y_t \quad -y_t \quad \ldots \quad -y_{t-n-1}] \\ &\theta = [a_0 \quad a_1 \quad \ldots \quad a_{n-1}] \\ &y_{t+1} = \varphi_t^T \theta + e_{t+1} \end{aligned}$$

The one step predictor is trivial and is given by the last equation without the unpredictable term that is the Gaussian noise:

$$\hat{y}_{t+1|t}^{\theta} = \varphi_t^T \theta$$

In order to find the best approximation of the parameters the following quadratic prediction error function has to be minimized:

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}} \frac{1}{N} \sum_{t=0}^{N} (y_t - \hat{y}^{\theta}_{t+1|t})^2 \tag{2.2}$$

Where N is the number of collected data and, obviously, the estimation is more accurate if N is big. The problem is a typical least square problem, and can be solved in closed form using the following equation:

$$\hat{\theta} = \frac{\sum_{t=0}^{N} \varphi_t^T y_t}{\sum_{t=0}^{N} \varphi_t \varphi_t^T}$$

The last equation admits solution if and only if the matrix at the denominator is invertible and this propriety is related at the model identification, and is supposed to have always it.

2.3 Trajectory Prediction

Recalling the autoregressive model, and writing it in a more compact form:

$$y_{t+1} = \sum_{i=0}^{N-1} a_i y_{t-i} + e_{t+1}$$

where N is the model order, e_{t+1} is a Gaussian noise with zero mean and variance σ_p^2. The equation of the one step ahead predictor, as already said, is:

$$\hat{y}_{t+1|t} = \sum_{i=0}^{N-1} a_i y_{t-i}$$

The two steps ahead predictor is similar and is obtained sliding the temporal window in this way:

$$\hat{y}_{t+2|t} = a_0 \hat{y}_{t+1|t} + \sum_{i=0}^{N-2} a_{i+1} y_{t-i}$$

The k-steps ahead predictor is also trivial:

$$\hat{y}_{t+k|t} = a_0 \hat{y}_{t+k-1|t} + a_1 \hat{y}_{t+k-2|t} + \ldots + a_{k-2} \hat{y}_{t+1|t} + \sum_{i=0}^{N-k} a_{i+k-1} y_{t-i}$$

It is interesting compute the prediction error, knowing that the uncertainness is bigger if the prediction horizon is longer. Using the Z-transform the equation becomes simpler, the

autoregressive model can be rewritten like the product between the Z-transform of a white noise and a transfer function dependent by the autoregressive coefficients.

$$Y(z) = H(z)E(z) = \frac{1}{1 - a_0 z^{-1} - \ldots - a_{N-1} z^{-N}} E(z)$$

Now if we came back in the time domain the representation is given by a convolution between the impulse response of H(z) and the white noise.

$$y_t = \sum_{i=0}^{\infty} h_i e_{k-i}$$

The mean of y_t is zero because the system is driven by a random signal with zero mean. Where we can compute the h_i terms with the inverse Z transform or could also be convenient use the method of the long division.
The variance of k step prediction error is equal to:

$$var(x_{t+k} - \hat{x}_{t+k|t}) = \sum_{i=0}^{k-1} h_i^2 var[e] \tag{2.3}$$

Just to give an example we compute the two steps prediction error.

$$\hat{y}_{t+2|t} = a_0 \hat{y}_{t+1|t} + \sum_{i=0}^{N-2} a_{i+1} y_{t-i} = a_0 \sum_{i=0}^{N-1} a_i y_{t-i} + \sum_{i=0}^{N-2} a_{i+1} y_{t-i}$$

So the prediction error became:

$$\begin{aligned} y_{t+2} - \hat{y}_{t+2|t} &= \sum_{i=0}^{N-1} a_i y_{t-i+1} + e_{t+2} - \hat{y}_{t+2|t} \\ &= a_0 y_{t+1} + \sum_{i=1}^{N-1} a_i t_{t-i+1} + e_{t+2} - \hat{y}_{t+2|t} \\ &= a_0 [\sum_{i=0}^{N-1} a_i y_{t-i} + e_{t+1}] + \sum_{i=1}^{N-1} a_i y_{t-i+1} + e_{t+2} - \hat{y}_{t+2|t} \\ &= a_0 e_{t+1} + e_{t+2} \end{aligned} \tag{2.4}$$

In this example the mean of the error is equal to zero and the variance of the error is equal to $a_0^2 \sigma_p^2 + \sigma_p^2$.
Using the formula (2.3) and the method of the long division to compute the h_i coefficients is possible make a comparison between the two steps prediction error computed above. The

result is the same because the first two coefficient of the impulse response are 1 and a_0.

2.4 Trajectory Representation

The reference trajectory to follow can be represented in state space form.

$$\begin{cases} r_{t+1} = F_r r_t + G_r n_t \\ y_t^r = H_r r_t + z_t \end{cases} \tag{2.5}$$

Where r_t containts the real values of the trajectory from t to $t+N$, but just an approximation of the first sample is known and the others are predicted using the AR model. n_t and z_t are zero mean and finite variance white noise where the first represent the uncertainness of the unpredicted state N+1 and the second the measurement and prediction errors. In particular the variances assumed over the time by z_t are the following

$$var(z_t) = \begin{cases} \sigma_m^2 & \text{at t} \\ \sigma_{p_1}^2 & \text{at t+1} \\ \sigma_{p_2}^2 & \text{at t+2} \\ \vdots & \\ \sigma_{p_k}^2 & \text{at t+k} \\ \vdots & \\ \sigma_{p_N}^2 & \text{at t+N} \end{cases} \tag{2.6}$$

Where σ_m^2 is the measurement error variance and $\sigma_{p_i}^2$ is the prediction error variance at time $t+i$. Moreover the model matrix are:

$$F_r = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \\ 0 & 0 & 0 & \dots & 0 \end{bmatrix}$$

$$G_r = \begin{bmatrix} 0 & 0 & 0 & \dots & 1 \end{bmatrix}^T$$

$$H_r = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \end{bmatrix}$$

The matrix F_r is a nilpotent matrix of dimension N so at every step selects a different value of the predicted trajectory.

2.5 Cost Function

The cost function to minimize is the following and is based on the noisy measures.

$$J(N, y_t^s, y^r) = \|(y_{t+N}^s - y_{t+N}^r)\|_P^2 + \sum_{j=0}^{N-1} \|(y_{t+j}^s - y_{t+j}^r)\|_{Q_{N-j-1}}^2 + \|u_{t+j}\|_{R_{N-j-1}}^2 \tag{2.7}$$

where the norm is the euclidean norm, P,Q_i are a set of $2N$ semi positive symmetric matrices, and R_i are definite positive symmetric matrices.

2.6 Tracking stability

Theorem 2.6.1. *Given the system* (2.1) *and a trajectory, predicted by past measurement, represented with the model* (2.5) *the stochastic receding horizon problem based on the cost function* (2.7) *is solved under the same assumptions of the deterministic one.*

Proof. Considering the following augmented state:

$$x_t = \begin{bmatrix} s_t \\ r_t \end{bmatrix}$$

$$x_{t+1} = Fx_t + Gu_t = \begin{bmatrix} F_s & 0 \\ 0 & F_r \end{bmatrix} x_t + \begin{bmatrix} G_s & 0 \\ 0 & G_r \end{bmatrix} \begin{bmatrix} u_t \\ n_t \end{bmatrix}$$

$$e_t = y_t^s - y_t^r = \begin{bmatrix} H_s & -H_r \end{bmatrix} x_t + v_t - z_t = err_t + m_t$$

where err(t) is the real distance between the state of the two agents and m_t is the difference of the Gaussian noise so is always a Gaussian noise with zero mean and variance the sum of

the two variances. In our particular case the variance of m_t is equal to:

$$var(m_t) = \begin{cases} 2\sigma_m^2 & \text{at t} \\ \sigma_{p_1}^2 + \sigma_m^2 & \text{at t+1} \\ \sigma_{p_2}^2 + \sigma_m^2 & \text{at t+2} \\ \vdots & \\ \sigma_{p_k}^2 + \sigma_m^2 & \text{at t+k} \\ \vdots & \\ \sigma_{p_N}^2 + \sigma_m^2 & \text{at t+N} \end{cases}$$

Rewriting the cost function in this way:

$$\begin{aligned} J(N, x_t) &= \|err_{t+N} + m_{t+N}\|_P^2 + \sum_{j=0}^{N-1} \|err_{t+j} + m_{t+j}\|_{Q_{N-j-1}}^2 + \|u_{t+j}\|_{R_{N-j-1}}^2 \\ &= \|err_{t+N}\|_P^2 + \|m_{t+N}\|_P^2 + 2 < err_{t+N}, m_{t+N} >_P + \sum_{j=0}^{N-1} \|err_{t+j}\|_{Q_{N-j-1}}^2 \\ &+ \|m_{t+j}\|_{Q_{N-j-1}}^2 + 2 < err_{t+j}, m_{t+j} >_{Q_{N-j-1}} + \|u_{t+j}\|_{R_{N-j-1}}^2 \end{aligned}$$

It is known that the mean of m_t is equal to zero and the distribution of a square Normalized Gaussian is a Chi Squared where the mean is given by the degree of freedom. Recalling also that the variance is equal to the raw moment if the mean is zero because holds $Var[m] = E[m^2] - E[m]^2$. Considering the mean of the cost function and the relations just written:

$$\begin{aligned} E[J(N, x_t)] =& \|err_{t+N}\|_P^2 + var\{m_{t+N}\}P_N + \sum_{j=0}^{N-1} [\|err_{t+j}\|_{Q_{N-j-1}}^2 + var\{m_{t+j}\}Q_{N-j-1} + \|u_{t+j}\|_{R_{t+N-j-1}}^2] \\ =& \|err_{t+N}\|_P^2 + \sum_{j=0}^{N-1} [\|err_{t+j}\|_{Q_{N-j-1}}^2 + \|u_{t+j}\|_{R_{t+N-j-1}}^2] + \sum_{j=0}^{N} P_j var\{m_{t+j}\} \end{aligned} \tag{2.8}$$

where in the last equations P_i stans for the Q_i. The receding horizon problem is based on the

minimization of this cost index, but observing that:

$$U_C^O(t) = Arg\min_u E(J(N,x_t)) = Arg\min_u E(\hat{J}(N,x_t))$$

$$where$$

$$E[\hat{J}(N,x_t)] = \|err_{t+N}\|_P^2 + \sum_{j=0}^{N-1} \|err_{t+j}\|_{Q_{N-j-1}}^2 + \|u_{t+j}\|_{R_{t+N-j-1}}^2$$

This is a standard Linear Quadratic problem and it is possible get a closed form solution for the receding horizon control.

$$u_t = -(G^T P_{N-1} G + R_{N-1})^{-1} G^T P_{N-1} F x_t = K_{N-1} x_t$$

To discuss about the stability of the tracking two important theorems are recalled. [18]

Theorem 2.6.2. *Consider the ARE associated with an infinite horizon LQ control problem*
$P = F^T PF - F^T PG(G^T PG + R)^{-1} G^T PF + Q$
where

- *(F,G) is stabilizable.*
- *(F,$Q^{1/2}$) has no unobservable modes on the unit circle.*
- *$Q \geq 0$ and $R > 0$.*

Then

- *there exists a unique, maximal, non negative definite symmetric solution $\bar{P}$.*
- *$\bar{P}$ is a unique stabilizing solution.*

The theorem above is fundamental for the stability of the infinite horizion LQ control, the next is fundamental for the RHC asymptotic stability.

Theorem 2.6.3. *Consider the ARE and its stabilizing solution $\bar{P}$, and consider the RDE*
$P_{t+1} = F^T P_t F - F^T P_t G(G^T P_t G + R)^{-1} G^T P_t F + Q$
Then, provided (F,G) is stabilizable. (F,$Q^1/2$) is detectable and P(0) $\geq$0, $P_t \to \bar{P}$ as $t \to \infty$

These theorems provide us a sufficient condition for the stability of the receding horizion control, to consider it valid is important have big values of N. ■

2.7 Tracking Performances with unit delay

Here is discussed the stability of the controller using the target trajectory known with a step delay. In this case the mean input is exactly the same input computed for the control without delay, but the difference is given by the variance that is bigger. Looking at the prediction of the target trajectory, an extra step prediction is necessary and the variance of all the steps is bigger than a quantity equal to σ_p(the variance of the one step prediction). It is shown that the central momentum of the cost function increase if the target trajectory is known with a delay. Considering this simplified cost function:

$$J(y_t^s, y_t^r, N) = \sum_{j=0}^{N-1} \|y_{t+j}^s - y_{t+j}^r\|_{Q_j}^2$$

The central momentum is given by $E[J(y_t^s, y_t^r, N)^2]$, so rewriting it in this way:

$$E[J(y_t^s, y_t^r, N)^2] = E[\sum_{j=0}^{N-1} \|err_{t+j} + m_{t+j}\|_{Q_j^2}^4]$$

Here is computed the square of the square:

$$\begin{aligned}\|err_{t+j} + m_{t+j}\|_{Q_j^2}^4 =& \|err_{t+j}\|_{Q_j}^4 + \|m_{t+j}\|_{Q_j}^4 + 4\|m_{t+j} err_{t+j}^3\|_{Q_j}^2 + \\ & 4\|m_{t+j}^2 err_{t+j}\|_{Q_j}^2 + 6\|m_{t+j}^2 err_{t+j}^2\|_{Q_j}^2\end{aligned}$$

Moreover all the odd raw moment of a Gaussian distribution are equal to zero if the mean is zero.

$$\begin{aligned}E[\|err_{t+j} + m_{t+j}\|_{Q_j^2}^4] &= \|err_{t+j}\|_{Q_j}^4 + E[\|m_{t+j}\|_{Q_j}^4] + 6E[\|m_{t+j}^2 err_{t+j}^2\|_{Q_j}^2] \\ &= \|err_{t+j}\|_{Q_j}^4 + 3\sigma_{p_j}^2 + 6\sigma_{p_j}^2 err^2(t+j)\end{aligned}$$

And then:

$$E[J(x_t, N)^2] = \sum_{j=0}^{N-1} Q_j^2 (err(t+j)^4 + 3\sigma_{p_j}^2 + 6\sigma_{p_j}^2 err^2(t+j))$$

When there is a delay of one or more steps the variance of the prediction grows and also the statistic power grows.

2.8 Simulation Results

In this section all the results obtained are verified via simulations. All the simulations are made using Matlab and same extra toolboxes. In the first subsection is tested the parameters estimation using the least square method. In the second subsection the prediction performances are tested. In the third and fourth subsections is tested the tracking with and without delays of a circular trajectory with a white noise added. In the last section is tested a multiagent system, the first agent tracks a circular trajectory and the second tracks the first with a distance of about 40cm along both the axis.
The model considered is the following:

$$\begin{bmatrix} x_{t+1} \\ y_{t+1} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_t \\ y_t \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} u_t^x \\ u_t^y \end{bmatrix} \tag{2.9}$$

And the trajectory to follow is a circular trajectory with white noise added.

Parameter Estimation

To estimate the model parameters is used the free matlab toolobox $arfit$ [23], that allows to solve the least squares problem.The code implementation first of all estimates the parameters and after this the prediction is made. The prediction is analysed in the next section.

```
1  function [pred] = prediction(qr,k)
2  % PREDICTION This function recognize the model from the data
3  % and gives as output the predicted step from t+1 to t+k
4
5      n = 3;
6      % Estimation of the parameters
7      [wX,paramX] = arfit(qr(1,:)',n,n);
8      [wY,paramY] = arfit(qr(2,:)',n,n);
9
10     pred = [qr(:,end-n+1:end),zeros(2,k)];
11
12     % Prediction
13     for j=1:k
14        pred(1,j+n) = paramX(end:-1:1)*pred(1,j:j+n-1)' + wX;
15        pred(2,j+n) = paramY(end:-1:1)*pred(2,j:j+n-1)' + wY;
16     end
17
18     pred = pred(:,end-k+1:end);
19
```

```
20 end
```

In this code snippet the predicted value of the trajectory is computed and the parameters are estimated for all the equations of the linear model.
In the simulations the model is driven with samples obtained from a circular trajectory with white noise added. From the plot is shown that after a transient the parameters converge to a steady state value and the poles of the model are stable.

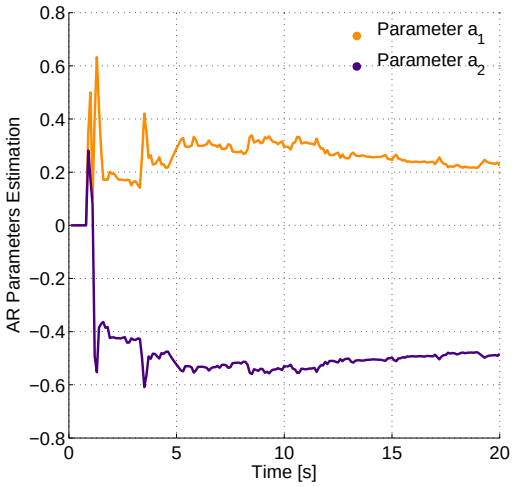

Figure 2.1: Autoregressive model parameters estimation over the time.

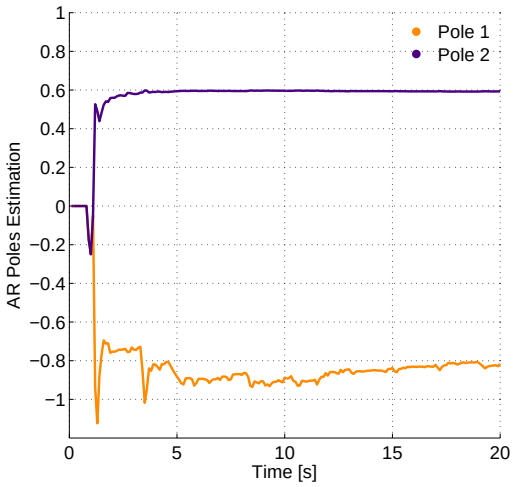

Figure 2.2: Autoregressive model poles estimation over the time.

Prediction Analysis

After a transient used to collect enough data in order to made invertible the matrix at the denominator on the least square formula (2.2) the prediction starts. The trajectory used in the simulations is always circular with center $(1.12, -1.5)[m]$ and radius $0.75[m]$. The prediction horizon is equal to 3. In this example is added a Gaussian noise with zero mean and variance $\sigma^2 = 0.1$.

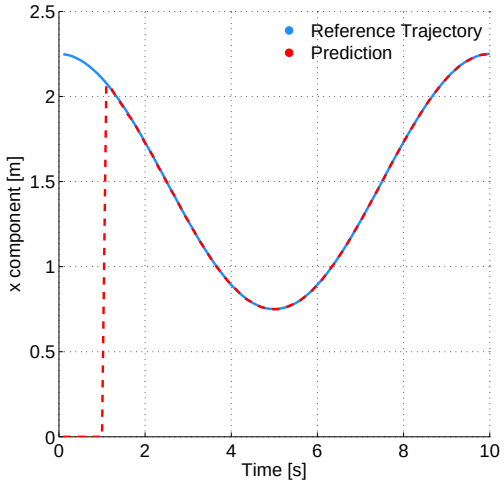

Figure 2.3: Prediction of the x state component.

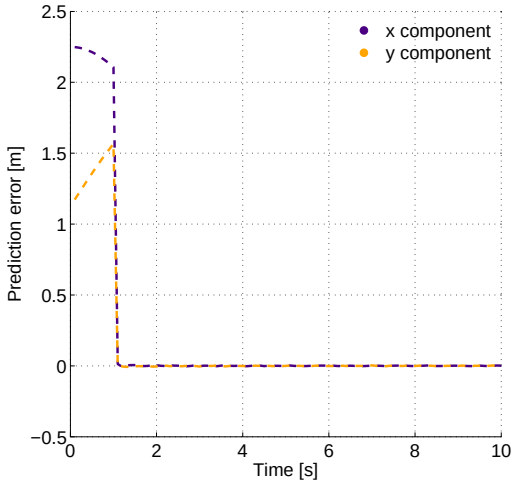

Figure 2.4: Prediction errors along each axes.

Tracking without delays

In the simulations the target is moving on a circle of center $(1.12, -1.50)[m]$ and radius $0.75[m]$, the target actual position is supposed to be known without delays also if in the reality the actual position is known with at least one step delay, moreover the noise variance is equal to 0.1. After a short transient the tracking is almost perfect and the agent follows the trajectory with a really small delay that can be reduced changing the weights of the control in the cost function. The initial conditions of the agent are (0,0)[m] and the length of the control horizon is 3. The parameter of the RHC are $R = 0.1\mathbb{I}_2$ and $P = \mathbb{I}_2$.

Tracking with unit delay

The simulations carry on again the same task but the target trajectory is known with a step delay and the circle to track has center in $(1.12, -1.50)[m]$ with radius $0.75[m]$. The initial conditions are $(2.70, -1.25)[m]$ and the parameters of the receding horizon control are $R = 0.1\mathbb{I}_2$, $P = \mathbb{I}_2$ and $N = 3$. The white noise added has a variance of $\sigma^2 = 0.1$. The results of the simulation are really good also with this limitation.

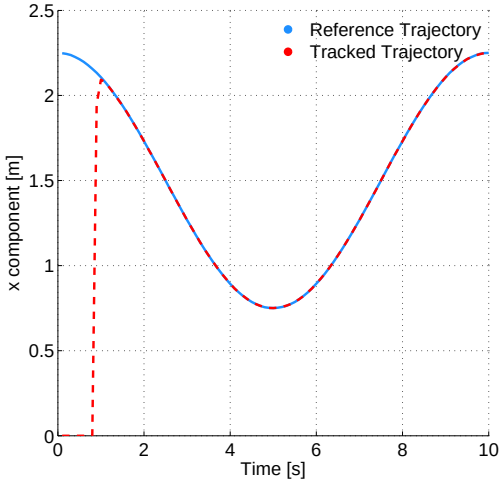

Figure 2.5: Tracking of the x state component using RHC with parameters $R = 0.1\mathbb{I}_2$ and $P = \mathbb{I}_2$.

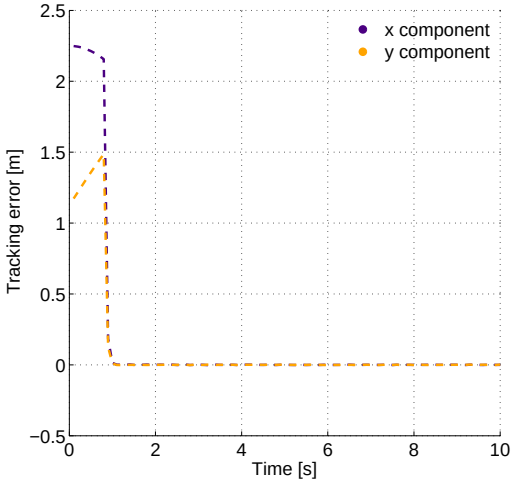

Figure 2.6: Tracking error along each axes.

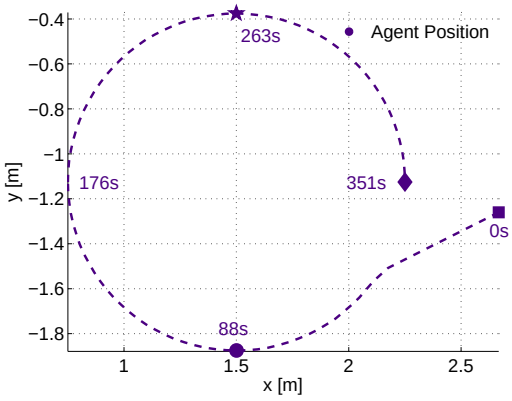

Figure 2.7: Tracking of a circular trajectory known with a one step delay using RHC with parameters $R = 0.1\mathbb{I}_2$ and $P = \mathbb{I}_2$.

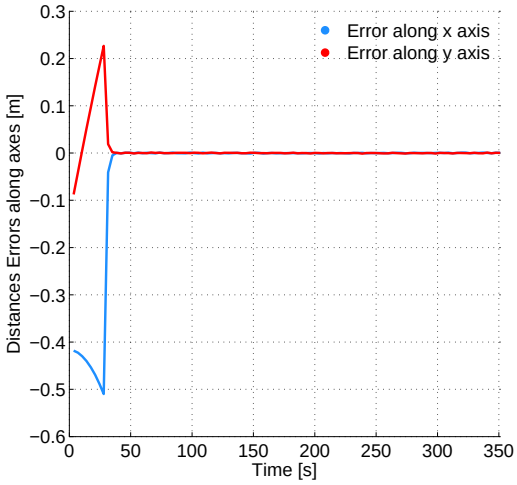

Figure 2.8: Tracking error of a circular trajectory known with a one step delay.

Tracking with Two Agents

In this simulation the first agent tracks a circular trajectory and the second tracks the first keeping a distance of 37.5 [cm] on both the axes. The result is that the second agents tracks a circular trajectory with a different center, moved exactly of 37.5 [cm] on both the axes. The reported results show good performances, considering also that the circular trajectory is know with a white noise of zero mean and variance $\sigma^2 = 0.1$. The parameters used are $P = 0.5\mathbb{I}_2$ and $R = 40\mathbb{I}_2$ for both the agents. The initial conditions are $(1.49, -1.05)[m]$ for the first and $(1.94, -1.18)[m]$ for the second.

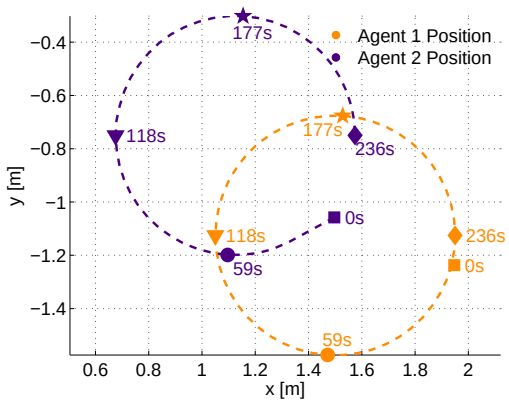

Figure 2.9: Position of the two agents.

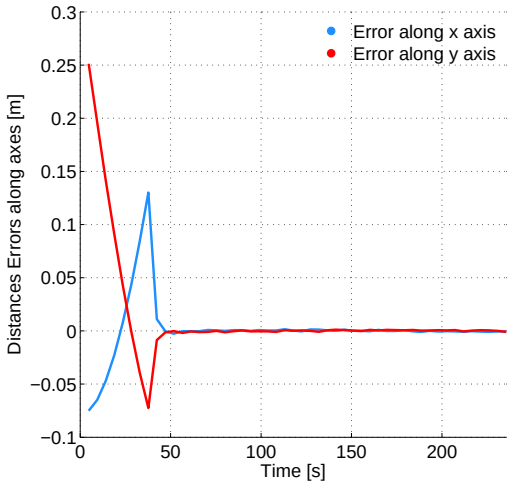

Figure 2.10: Distance errors between the two agents.

2.9 Conclusion

The performances of the algorithm analysed are good also with a unit delay. Considering that also the trajectory is known with a white noise added we can conclude that this technique is reliable and resilient at the external noises.

3

Competition using Receding Horizon

In this chapter, we consider two agents described by a linear discrete-time dynamic system controlled with the Receding Horizon technique. Each agent is provided with a local target, that could be conflicting with the control objective of the others. For example the first wants to catch the other agent, but the second wants reach a safety distance from the first. We show that for this unconstrained system, the agents with conflicting targets, reach an equilibrium along a line and the steady state error is finite. Otherwise if the control objectives are not conflicting the agents will reach them and the steady state position is fixed and depends by the initial conditions. Simulation and experimental results for a group of UGVs are also reported The experimental tests were executed with LEGO robots and more information for educational purposes can find here [8].

In Section 3.1, the linear discrete-time dynamic system and the control objective are defined. In Section 3.2 is proofed that the dynamics of the competitive agents converges along a line and also that the steady state error is finite. In Section 3.3 is proofed that the system doesn't admit limit cycle. In Section 3.4 is shown, when there is cooperation, that the final position depends by the initial condition. In the Section refsec:sim are reported the simulation results and finally, Section VIII provides conclusions.

3.1 Problem Formulation

We consider two agents described by the discrete LTI state equation:

$$s^i_{t+1} = s^i_t + u^i_t, \qquad i = 1,2, \tag{3.1}$$

where $s^i_t \in \mathbb{R}^n$ denotes the state vector, and $u^i_t \in \mathbb{R}^n$ the input vector. We assume that agents can measure each other's state without delays.

For each agent, and for a given value of the state vector s^i_t at time instant t, we introduce the following convex cost function:

$$J_{i,j} = \sum_{k=1}^{N^i-1} (\|s^i_{t+k} - s^j_t + d^{ij}\|^2_{P^i} + \|u^i_{t+k}\|^2_{R^i}), \qquad i \neq j, \tag{3.2}$$

where $\|\cdot\|$ is the euclidean norm, $i,j \in 1,2$, N^i denote the lengths of the control horizon for each agent, P^i, R^i are positive diagonal matrices, and $d^{ij} \in \mathbb{R}^n$ are the desired distances between the state components of the agents [13**?**]. The sign and modulus of d^{ij} determine whether the agents are competing or cooperating.

Definition 3.1.1. If $d_{ij} = -d_{ji}$ in cost function (3.2), then we say the agents control objective is cooperative; otherwise, if $d_{ij} \neq -d_{ji}$ the control objective is considered competitive.

The agents compute their control action minimizing function (3.2) iteratively, according to the RHC algorithm [6]:

Algorithm 3.1.1. *At each time t the ith agent, having state s^i_t, collects information about the state of the other agent s^j_t. Assuming s^j_t is constant in the N^i-step prediction window, the ith agent minimizes cost function (3.2) solving a finite horizon optimal control problem. The optimal input vector $u^i_t, u^i_{t+1}, \ldots, u^i_{t+N}$ is computed, and only the first element of the control vector, u^i_t is applied. This procedure is repeated at each time instant.*

Problem 3.1.1. *Using the RHC algorithm 3.1.1 we want to establish the steady state behavior of system (3.1) as a function of vector d_{ij}.*

3.2 Steady State Behaviour

Theorem 3.2.1. *Consider the two agents described in expression (3.1), controlled using the RHC strategy 3.1.1, without any state and input constraints. We define the agents distance:*

$$e^{ij}_t = s^i_t - s^j_t \tag{3.3}$$

1. *If $d_{ij} \neq -d_{ji}$, the dynamics of the agents converge to a line, for $t \to \infty$ the agents distance is finite, and $\lim_{t\to+\infty} e_t^{ij} \neq -d_{ij}$ for each agent.*

2. *If $d_{ij} = -d_{ji}$ the agents reach a steady state position, and $\lim_{t\to+\infty} e_t^{ij} = -d_{ij}$.*

Proof. The objective of this proof is to find a closed form expression of the control law, in order to evaluate the closed-loop dynamics of the two agents. Due to the simplicity of our problem, we will not use the classical constrained optimization approach [**?**] leading to the algebraic Riccati equation treatment. We now make three observations:

i) We can rewrite the state vector at time $t+k$ explicitly as a linear function of state s_t^i and as a function of the inputs from t to $t+k-1$:

$$s_{t+k}^i = s_t^i + u_t^i + \ldots + u_t^{t+k-1}. \tag{3.4}$$

ii) Because state components are decoupled (there are no off-diagonal elements in the linear state dynamics), we can focus on each component separately. Thus, our problem becomes effectively a scalar RH optimization.

iii) The minimum of the cost function can be found by simply setting to zero its derivative with respect to the input:

$$\frac{\partial J_{i,j}}{\partial u_{t+q}^i} = 2P^i \sum_{k=q}^{N^i-1} (s_{t+k}^i - s_t^j + d^{ij}) + 2R^i u_{t+k}^i = 0, \tag{3.5}$$

where $0 \leq q \leq N^i - 1$, and expression (3.4) can be directly substituted to evaluate the minimum along the system trajectories.

Since we can focus on each scalar component separately, we can rewrite equation (3.5) in matrix form, where each row corresponds to the qth partial derivative of the cost function, with respect to u_{t+q}^i. We simplify our notation as: $N^i = N$, assuming both agents have the same prediction horizon; $P^k = p$ and $R^k = r$ where P^k and R^k indicate the weights of the kth state component. In the following equation the $q-th$ row of the matrix represent. We remark that i and j are indices associated to our two generic agents. We obtain a linear system of equations:

$$Au = b.$$

Where:

$$A = 2\begin{bmatrix} (Np+r) & (N-1)p & \dots & p \\ (N-1)p & ((N-1)p+r) & \dots & p \\ \vdots & & \dots & \vdots \\ p & p & \dots & (p+r) \end{bmatrix},$$

$$u = \begin{bmatrix} u_t^i \\ u_{t+1}^i \\ \vdots \\ u_{t+N-1}^i \end{bmatrix}, \qquad b = \begin{bmatrix} N \\ N-1 \\ \vdots \\ 1 \end{bmatrix} 2p(x_t^j - x_t^i - d_{ij}).$$

We need to: 1) Verify whether matrix A is invertible, and 2) If A is invertible, find explicitly the first row of its inverse: this will yield the first element of the sequence of control actions in the optimization window, which is given by the product between the first row of the inverse of the matrix A and the column vector b. We know that the first input element will be a function:

$$u_t^i = \alpha(p, r, N)(x_t^j - x_t^i - d_{ij}). \tag{3.6}$$

Our objective is to find an expression for α. We start by rewriting matrix A as the sum of two matrices, $H + G$, where:

$$G = 2\begin{bmatrix} r & 0 & \dots & 0 \\ -p & r & \dots & 0 \\ \vdots & & \dots & \vdots \\ -(N-1)p & -(N-2)p & \dots & r \end{bmatrix},$$

$$H = 2p\begin{bmatrix} N & (N-1) & \dots & 1 \\ N & (N-1) & \dots & 1 \\ \vdots & & \dots & \vdots \\ N & (N-1) & \dots & 1 \end{bmatrix}.$$

Matrix H is lower triangular and invertible, while matrix G is singular with rank one. We invoke the main result in [16] (theorem 7.1 reported in Appendix I), which helps us finding the inverse of the sum of two matrices:

$$(G+H)^{-1} = G^{-1} - \frac{1}{1+g} G^{-1} H G^{-1}. \tag{3.7}$$

Matrix G is also a Toeplitz matrix, so we invoke theorem 7.2 in [24] (reported in Appendix

I), which gives us a closed form for its inverse. Denoting the coefficients of the inverse of the Toeplitz matrix G as g_n, we find:

$$g_n = \sum_{k=1}^{n} \frac{(-1)^k k!}{a_0^{k+1}} \sum_k \frac{1}{k_1 \dots k_n!} a_1^{k_1} \dots a_n^{k_n}.$$

Thus, the inverse of G is:

$$G^{-1} = \frac{1}{2} \begin{bmatrix} \frac{1}{r} & 0 & \dots & 0 \\ g_1 & \frac{1}{r} & \dots & 0 \\ \vdots & & \dots & \vdots \\ g_{N-1} & \dots & g_1 & \frac{1}{r} \end{bmatrix}.$$

With few additional steps we can evaluate the inverse of matrix A:

$$g = \mathrm{Tr}\left(HG^{-1}\right) = \frac{p}{r}\frac{N(N+1)}{2} + p\sum_{z=1}^{N-1}(g_z \sum_{w=1}^{N-z} w)$$

$$G^{-1}HG^{-1} = \\ = \frac{p}{2r} \begin{bmatrix} \frac{N}{r} + \dots + g_{N-1} & \frac{N-1}{r} + \dots + g_{N-2} & \dots & \frac{1}{r} \\ * & * & \dots & * \\ \vdots & \vdots & \vdots & \vdots \\ * & * & \dots & * \end{bmatrix}$$

And finally:

$$(H+G)^{-1} = \\ \begin{bmatrix} \frac{1}{2r} - \frac{p}{2(1+g)r}(\frac{N}{r} + \dots + g_{N-1}) & \dots & -\frac{p}{2(1+g)r^2} \\ * & \dots & * \\ \vdots & \vdots & \vdots \\ * & \dots & * \end{bmatrix}$$

Now we can evaluate coefficient α in expression (3.6), which is equal to the first row of

$A^{-1}b$:

$$\begin{aligned}\alpha(p,r,N) =& 2p\left\{\frac{N}{2r} - \frac{Np}{2(1+g)r}\left[\frac{N}{r} + (N-1)g_1 + \right.\right.\\ & \left. \ldots + g_{N-1}\right] - \frac{(N-1)p}{2(1+g)r}\left[\frac{N-1}{r} + \right.\\ & \left.\left. \ldots + g_{N-2}\right] - \ldots - \left[\frac{p}{2(1+g)r}\frac{1}{r}\right]\right\}\end{aligned} \tag{3.8}$$

The first inputs of the optimal control sequence for the two agents are:

$$\begin{aligned}u_t^i &= \alpha(N, P^{i_x}, R^{i_x})(x_t^j - x_t^i - d_{ij})\\ u_t^j &= \gamma(N, P^{j_x}, R^{j_x})(x_t^i - x_t^j - d_{ji})\end{aligned} \tag{3.9}$$

Where in our notation γ is the control coefficient computed exactly as in expression (3.8), but with weights P^{j_x} and R^{j_x}. Finally, the closed-loop dynamics of the two systems are:

$$\begin{aligned}x_{t+1}^i &= x_t^i + u_t^i = (1-\alpha)x_t^i + \alpha x_t^j - \alpha d_{ij}\\ x_{t+1}^j &= x_t^j + u_t^j = (1-\gamma)x_t^j + \gamma x_t^i - \gamma d_{ji}\end{aligned} \tag{3.10}$$

Recalling expression (3.3), the distance among the two agents' states is componentwise given by:

$$e_{t+1}^{ij} = (1-\alpha-\gamma)e_t^{ij} + \gamma d_{ji} - \alpha d_{ij}. \tag{3.11}$$

Boundedness of this distance is guaranteed if the sum of the controller coefficients be bounded between 0 and 2. Thus, stability is assured if both α and γ are in the interval $[0,1]$. These bounds are indeed always verified.
The lower bound can be demonstrated by multiplying $(G+H)^{-1}$ by vector $\not{1}^T = [N \quad N \quad \ldots \quad N]$, instead of by vector $[N \quad N-1 \quad \ldots \quad 1]^T$. We obtain:

$$\alpha(N,p,r) > \frac{Np}{r}(1 - \frac{g}{1+g}) > 0.$$

The upper bound is also always verified; the proof is in Appendix II.
Thus, at steady state we have:

$$e^{ij} = (1-\alpha-\gamma)e^{ij} + \gamma d_{ji} - \alpha d_{ij},$$

where e^{ij} is equal to:

$$e^{ij} = \frac{\gamma d_{ji} - \alpha d_{ij}}{(\alpha+\gamma)}.$$

We can now conclude the proof of point 1) of our Theorem. If $d_{ij} \neq -d_{ji}$, after a transient that depends by the values of α and γ, the agents distance reaches a finite value; thus, the agents achieve a stationary regime and maintain a constant distance. This concludes the first part of our proof.

Now we conclude the proof for point 2). If the two agents cooperate, i.e. $d_{ij} = -d_{ji}$, we have:

$$e^{ij} = \frac{\alpha+\gamma}{\alpha+\gamma} d_{ij} = -d_{ij}$$

In this case the steady state positions of the two agents are fixed because:

$$u_t^i = -\alpha(N,p,r)(d_{ij} + e^{ij})$$
$$u_t^j = -\gamma(N,p,r)(d_{ji} - e^{ij})$$

the above can be true only if $d_{ij} = -d_{ji}$, i.e. if the objectives are cooperative. If the agents are competing, i.e. $d_{ij} \neq -d_{ji}$, after a transient the actuation will achieve a constant non-zero value. In a plane, for example, this means our agents will reach a stationary speed moving along a line. ■

Remark 1: A cooperative problem could become competitive for small variations or errors of the tracking objectives. If $d_{ij} = d_{ji} \pm \epsilon$, with ϵ arbitrarily small, the system does not reach a steady state. This pitfall can be easily overcome by introducing, for instance, an exponential decay factor (operating on a suitable time-scale) that progressively discounts the feedback action.

In the cooperation case, where the agents reach a final equilibrium, is easy to see that the equilibrium depends on the initial position and on the weights of cost function (3.2), as expected in a linear quadratic problem. As an example, consider two agents on a plane with coordinates (x, y), and take into account just the x component of the state s; the system has the following closed-loop dynamics:

$$\begin{bmatrix} x_{t+1}^i \\ x_{t+1}^j \end{bmatrix} = \begin{bmatrix} 1-\alpha & \alpha \\ \gamma & 1-\gamma \end{bmatrix} \begin{bmatrix} x_t^i \\ x_t^j \end{bmatrix} + \begin{bmatrix} \alpha d_{ij} \\ \gamma d_{ji} \end{bmatrix}$$

The constant term can be consider like a constant input vector that we can call c, so the state follows this law:

$$x_t = F^t x_0 + \sum_{k=0}^{t-1} F^{t-k-1} c = F^t x_0 + d_{ij} \begin{bmatrix} -\alpha \\ \gamma \end{bmatrix} \sum_{k=0}^{t-1} F^{t-k-1},$$

because $d_{ij} = -d_{ji}$. Matrix F matrix admits the limit:

$$\lim_{t\to\infty} F^t = \mathbb{1} w^T, \qquad \text{where} \quad w^T = \begin{bmatrix} \frac{\gamma}{\alpha+\gamma} \\ \frac{\alpha}{\alpha+\gamma} \end{bmatrix},$$

and w^T is the right eigenvector of matrix F; this relation holds because F is a stochastic matrix.

3.3 Absence of Limit Cycle

We compare again our two agents componentwise, since the state dynamics are not coupled.

Corollary 3.3.1. *The unconstrained discrete-time linear system* (3.1) *controlled with the RHC algorithm 3.1.1 does not admit oscillatory dynamics.*

Proof. A discrete-time linear system admits a limit-cycle if and only if its eigenvalues are on the unit circle. We rewrite the system to find expressions for the eigenvalues. We considering a generic component x of the state vector s, substituting expression (3.10):

$$\begin{bmatrix} x^i_{t+1} \\ x^j_{t+1} \end{bmatrix} = \begin{bmatrix} 1-\alpha & \alpha \\ \gamma & 1-\gamma \end{bmatrix} \begin{bmatrix} x^i_t \\ x^j_t \end{bmatrix} + \begin{bmatrix} \alpha d_{ij} \\ \gamma d_{ji} \end{bmatrix}$$

We denote the above state matrix as F, and we can compute the characteristic polynomial of the system:

$$\begin{aligned} \Delta_F(z) &= \det(zI - F) = (z+\alpha-1)(z+\gamma-1) - \alpha\gamma \\ &= z^2 + z(\gamma+\alpha-2) + (1-\alpha-\gamma) \end{aligned}$$

If we want the eigenvalues on the unit circle the following condition must hold:

$$\Delta_F(z) = z^2 - 2\cos(\psi)z + 1$$

We obtain that:

$$\begin{cases} 1 = 1 - \alpha - \gamma \\ -2\cos(\psi) = \gamma + \alpha - 2 \end{cases} \quad \begin{cases} \alpha + \gamma = 0 \\ -2\cos(\psi) = \gamma + \alpha - 2 \end{cases}$$

$$\begin{cases} \alpha + \gamma = 0 \\ \cos(\psi) = 1 \end{cases}$$

The condition $\cos(\psi) = 1$ holds if and only if there is a root in one, indeed if we evaluate the characteristic polynomial in one, $\Delta_F(1) = 0$. So there is an eigenvalue in one and cannot exist complex eigenvalues. For this reason there is the absence of limit-cycles. ■

3.4 Agents Final Position

In the unique case where the agents reach a fixed final position is easy to see that it depends by the initial position and by the receding horizon weights. Considering just the x component of the state s the system has the following closed-loop dynamics:

$$\begin{bmatrix} x^i_{t+1} \\ x^j_{t+1} \end{bmatrix} = \begin{bmatrix} 1-\alpha & \alpha \\ \gamma & 1-\gamma \end{bmatrix} \begin{bmatrix} x^i_t \\ x^j_t \end{bmatrix} + \begin{bmatrix} \alpha d_{ij} \\ \gamma d_{ji} \end{bmatrix}$$

The constant term can be consider like a constant input vector that we can call c, so the state follows this law:

$$x_t = F^t x_0 + \sum_{k=0}^{t-1} F^{t-k-1} c = F^t x_0 + d_{ij} \begin{bmatrix} -\alpha \\ \gamma \end{bmatrix} \sum_{k=0}^{t-1} F^{t-k-1}$$

Because $d_{ij} = -d_{ji}$. We can see from the last equation the dependency from the initial condition. Moreover the matrix admits this limit:

$$\lim_{t\to\infty} F^t = \mathbb{1} w^T$$

$$where \quad w^T = \begin{bmatrix} \frac{\gamma}{\alpha+\gamma} \\ \frac{\alpha}{\alpha+\gamma} \end{bmatrix}$$

Where w^T is the right eigenvector of the matrix F, and this relation holds because F is a stochastic matrix.

3.5 Simulation Results

A set of two UGVs moving in $\mathbb{R}^2$ have been considered and we have been simulated a kind of cops and robbers game. The first agent, that by now we call cop, has to catch the other agent, that we call robber, so its objective vector d_{CR} is equal to the vector $0[\mathrm{m}]$. Instead the robber, to be considered safe, has to reach a certain threshold d_{RC} that we consider equal to the vector $[0.75 \quad 0.75]^T[m]$. For each agents we consider a finite horizon $N = 3$ and we try different combination of the P and R parameters. Initial conditions for the agents were chosen to be consistent with our experimental tests.
The dynamic of the agents is given by this equation:

$$\begin{bmatrix} x^i_{t+1} \\ y^i_{t+1} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x^i_t \\ y^i_t \end{bmatrix} + \begin{bmatrix} u^{i_x}_t \\ u^{i_y}_t \end{bmatrix}$$

Our first test is run choosing $P_C = 0.1\mathbb{I}_2$ and $R_C = 20\mathbb{I}_2$ for the cop, and $P_R = 0.1\mathbb{I}_w$ and $R_R = 10\mathbb{I}_2$ for the robber. The initial conditions are $[2.71, -2.05]m$ for the cop and $[2.73, -1.51]m$ for the robber. As shown in Figure 3.1, the steady state distance reached among the two agents is of $0.495m$ and as we could guess the robber take advantage of its faster mobility.

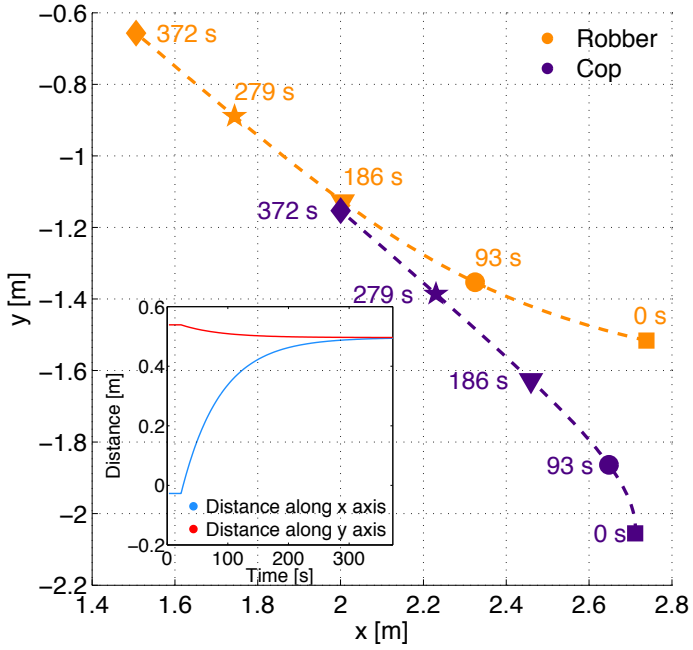

Figure 3.1: Position of the two agents and distance between them. The RHC cost function parameters are $P_C = 0.1\mathbb{I}_2$,$P_R = 0.1\mathbb{I}_2$, $R_C = 10\mathbb{I}_2$ and $R_R = 20\mathbb{I}_2$.

In the second test the parameters are $P_C = 0.1\mathbb{I}_w$ and $R_C = 10\mathbb{I}_2$ for the cop and $P_R = 0.1\mathbb{I}_w$ and $R_R = 20\mathbb{I}_2$ for the robber. The initial conditions are $[2.70, -2.07]m$ for the cop and $[2.77, -1.58]m$ for the robber. The steady state distance reached among the two agents is of $[0.255]m$ and the results obtained is the opposite compared to the first test.

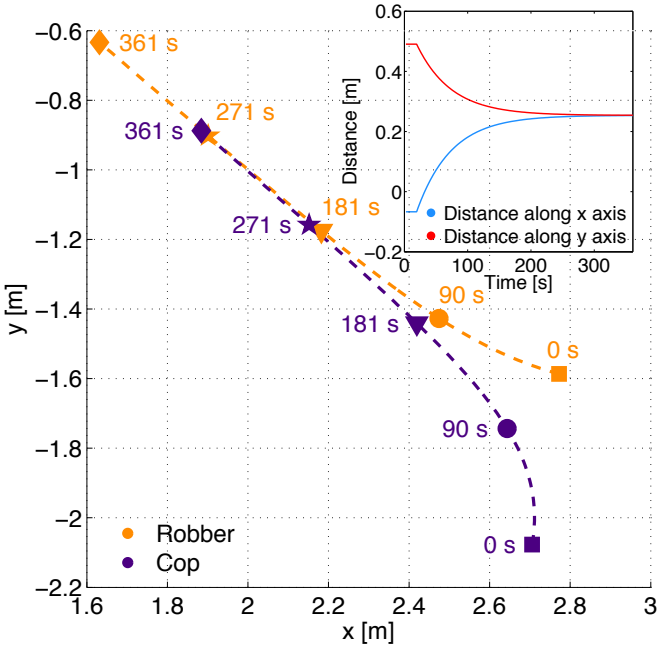

Figure 3.2: Position of the two agents and distance between them. Parameters are $P_C = 0.1\mathbb{I}_2$,$P_R = 0.1\mathbb{I}_2$, $R_C = 20\mathbb{I}_2$ and $R_R = 10\mathbb{I}_2$.

In the last competitive test we choose $P_C = P_R = 0.5\mathbb{I}_2$ and for $R_C = R_R = 20\mathbb{I}_2$. The challenge between the two agents finish with a tie as we can see in Figure 3.3, because $\alpha = \gamma$. The distance between the agents is $0.375m$, exactly half way for each the targets. The initial conditions are $[2.71, 2.11]m$ for the cop and $[2.89, 1.63]m$.

If the agents have coordinated tracking objectives (cooperative case), using the same parameters of the last simulation we can see how the two agents reach a fixed position in a finite time and the final positions depends by the initial conditions, that in this case are $[1.49, -1.05]m$ for the first agent and $[2.79, -1.04]m$. Results are shown in Figure 3.4.

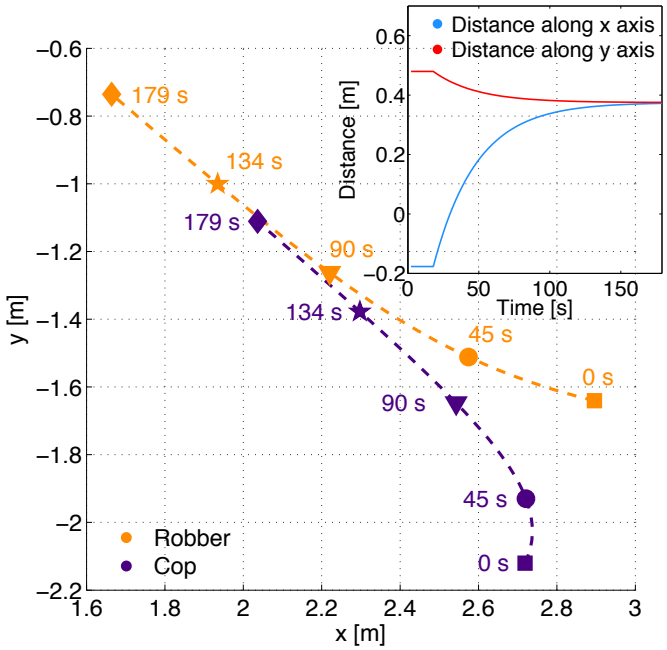

Figure 3.3: Position of the two agents and distance between them. Parameters: $P_C = 0.5\mathbb{I}_2$,$P_R = 0.5\mathbb{I}_2$, $R_C = 20\mathbb{I}_2$ and $R_R = 20\mathbb{I}_2$.

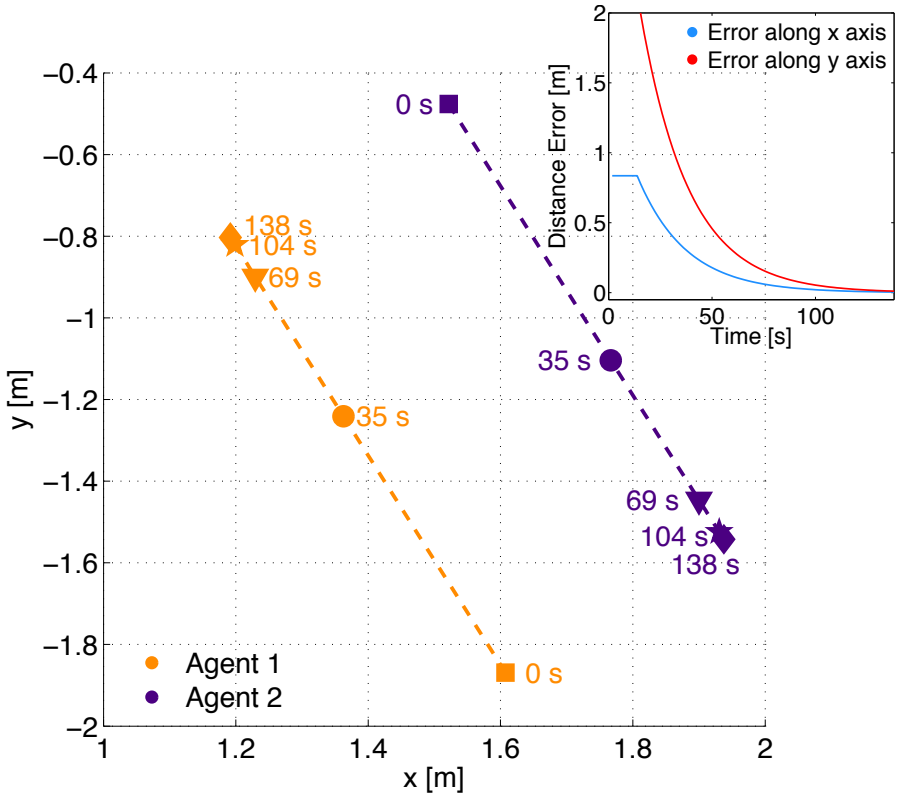

Figure 3.4: Position of the two agents and distance between them. The parameters used are $P_1 = 0.5\mathbb{I}_2$,$P_2 = 0.5\mathbb{I}_2$, $R_1 = 20\mathbb{I}_2$ and $R_2 = 20\mathbb{I}_2$

3.6 Conclusion

We presented a simple RHC framework for a two-agent systems with competitive/cooperative dynamics. Our main contributions are: 1) We find analytic solutions and establish the possible dynamic outcomes within the system. 2) We experimentally (see next chapter) verify our results with a LEGO robot kit, a versatile educational platform. Future research will investigate the scalability of our results to larger teams of agents, and the inclusion of simple estimators in the algorithms. For instance, each agent could build simple ARMA models to predict the future moves of its neighbors or opponents, and use that information within the finite-time RHC optimization; local predictions of other team members strategies has been previously utilized in distributed RHC, but knowledge (rather than estimation) of other agentsÕ models was assumed.

4

Robotic Platform and Experimental Results

The ME Group of the UCR has decided to develop an experimental platform for multiple vehicle with the objective to test and compare different control algorithms for multi-agent systems. In this chapter are explained the software and hardware setup of the experimental platform, that can be used also for educational purposes, so it will be analysed all the details and the source code can be find here [8]. In the first section there is a description of the hardware and of the packages used, in the second is described the software architecture that is divided in functional blocks. In the third are explained all the software details. In the forth section there are the experimental results obtained and the comparison with the simulations results. It is important highlight the limitations of the platform that are caused by the technological limitations of the hardware used, starting from the LEGO NXT Brick, the bluetooth connection and the webcam used.

4.1 Experimental Setup

The hardware used in the experiment is the following:

1. 2 LEGO NXT Robots

2. 1 LEGO NXT Brick as Bluetooth Router

3. Laptop MacBook Pro 13”

4. Webcam Microsoft LifeCam Studio

LEGO Robots

The robots were build using the LEGO Mindstorm NXT kit. The "brain" of the robot is the NXT Intelligent Brick, it can take from up to four sensors, it can control up to three motors with a RJ12 cable and it also has a usb connector for connect the Brick to the PC. It has a 32bit ARM7 processor, 256Kb of flash memory, 64Kb of RAM, 8-bit AVR micro-controller, the bluetooth support and a speaker that plays sounds up to 8kHz. Power is supplied by 6 AA batteries in the consumer version of the kit and by a Li-Ion rechargeable battery and charger in the educational version. The intelligent brick has also a 100x60 monochrome LCD display and four buttons. The kit provides also three servo motors with an incremental encoder, a light sensor, a sound sensor, a touch sensor and an ultrasonic sensor. The model that we used is a tank based on the MULTIBOT project and the building instructions for the base model can be found here [17].

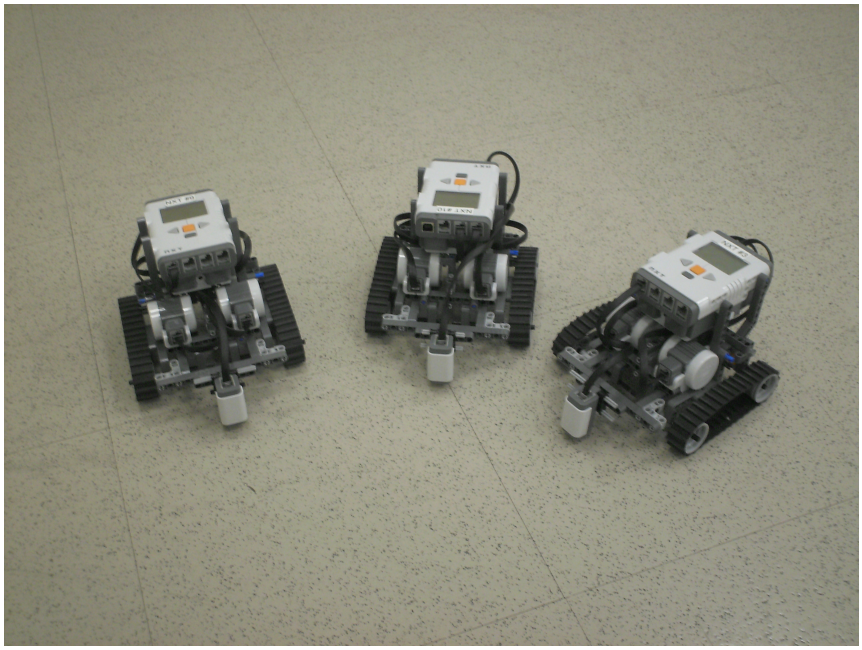

Figure 4.1: The LEGO robots used in the experiments.

Bluetooth Router

In order to establish a communication between the computer and the robots, an Intelligent Brick is used like Bluetooth router. The brick receives informations from the computer through the usb port and forward the informations to the correct robot. A simple communication protocol was implemented that is explained in the next section. More information about the bluetooth routing can be found here [3].

Laptop and Software

The laptop used for the test is an Apple MacBook Pro 13", m.y. late 2010, with a 2.4 GHz Intel Core 2 Duo, 4 Gb of RAM DDR3. All the code that is executed on the laptop is written in MATLAB, and the version used is the 2012a. For the model estimation is used the package $arfit$ that allows to compute quickly the least square problem in order to identify the robot motion model. This package is written by Schneider Tapio of the Caltech and the library can be found here [23]. To communicate with the NXT Brick using the usb port the RWTH free Mindstorm NXT Toolbox is used [20].

Webcam

The wecabm used is a Microsoft LifeCam Studio that is a 1080p HD cam, with auto Focus, High Precision Glass Element Lens, TrueColor technology and ClearFrame technology. It is compatible with MATLAB, and the resolution used is just 800x600 for load reasons. The solid and flexible standing is perfect to fix the cam to the ceiling of the platform room without using any kind of adaptor.

4.2 Software Architecture

The software architecture copies the structure of a closed loop control system. In the picture below the plant is represented by the robot, where the input is the angular speed of the wheels and the output is the position in the Cartesian plane and the orientation respect to the horizon (x-axis). The output is captured by a sensor, that in our case is the vision system with the cam. This system provides us the measurements of the state of the agents with delays and noise. In order to solve the vision problem we used some markers to understand where are the robots and what are their orientations. The setpoints are generated with an algorithm based on the theory developed in the last two chapters. Finally the control is transferred from the computer to the robots using the bluetooth communication that is handled with a custom script.

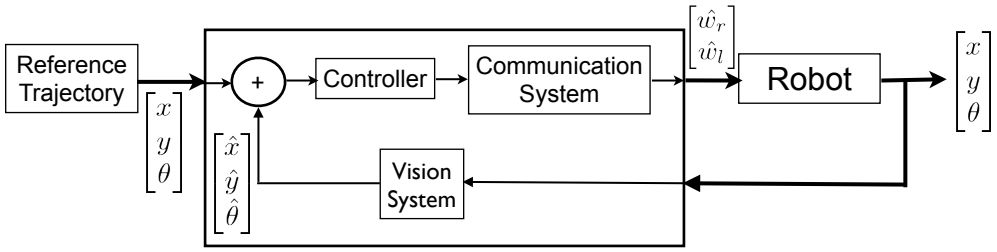

Figure 4.2: Software Architecture.

The structure described above is just a logical structure of the control system, all the code is organized in five functional blocks that now will be described. In the pictures below the controller block receives as input the target to reach and produces as output the position reached. Inside the controller block there are other two functional blocks, the first is the vision system block that when is called returns the position of the agents, the second is the communication block that receives as input all the actuation computed by the controller and it forwards the information to the two agents thorough the bluetooth router in a transparent way.

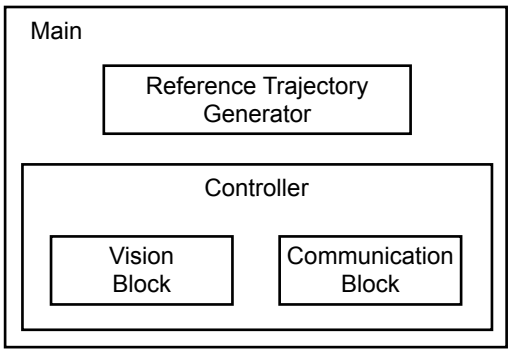

Figure 4.3: Software Functional Blocks.

4.3 Robot Model

The robots used in the experiment can be modelled like an uncycle because are tracked and the two drive wheels are handled independently. This kind of model is also used for a lot of vehicles in the industry or in the common life, an important characteristic of the unicycle is that it can rotate around his axis (e.g. using only one motor), but is also subject to a constraint: it can't move on the direction parallel at its wheelbase. The state vector is formed by three parameters: the (x,y) position and the orientation θ in respect with the x-axis. The

forward kinematics is regulated by the following non linear system:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos(\theta) & 0 \\ \sin(\theta) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{4.1}$$

We need also to control the wheels speed of the robot so the link between the angular speed of the wheels and the linear and angular speed of the robot is provided by the following static system:

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} \frac{r}{2} & \frac{r}{2} \\ \frac{r}{d} & -\frac{r}{d} \end{bmatrix} \begin{bmatrix} \omega_R \\ \omega_L \end{bmatrix} \tag{4.2}$$

Where r is the radius of the wheel with the track and d is the wheelbase length. For the robots r is equal to $0.016m$ and d to $0.135m$, so the matrix is invertible and we can find the inverse relation.
This part of the control is usually implemented into the firmware of the vehicle with a PID controller and in this way is possible control the linear and the angular speed of the robot knowing the equation above.
It is also useful compute the inverse kinematics, but to do it is convenient introducing a modified version of the forward kinematics, because we need the relationship between the robot position and the angular speed of the wheels.

$$\begin{aligned} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} &= \begin{bmatrix} \cos(\theta) & 0 \\ \sin(\theta) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{r}{2} & \frac{r}{2} \\ \frac{r}{d} & -\frac{r}{d} \end{bmatrix} \begin{bmatrix} \omega_R \\ \omega_L \end{bmatrix} \\ &= \begin{bmatrix} \frac{r}{2}\cos(\theta) & \frac{r}{2}cos(\theta) \\ \frac{r}{2}sin(\theta) & \frac{r}{2}sin(\theta) \\ \frac{r}{d} & -\frac{r}{d} \end{bmatrix} \begin{bmatrix} \omega_R \\ \omega_L \end{bmatrix} = A \begin{bmatrix} \omega_R \\ \omega_L \end{bmatrix} \end{aligned} \tag{4.3}$$

The pseudo-inverse of A is the matrix that provide the inverse kinematics equations:

$$\begin{bmatrix} \omega_R \\ \omega_L \end{bmatrix} = A^{\dagger} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} \tag{4.4}$$

In order to control the robot is implemented a simple controller based on a P controller. The idea is that given a final destination the robot has to reach it (supposing that is reachable). The first action is an alignment of the robot in the direction of the target point, after this action the robot starts with a fixed slow linear speed and the P controller adjusts the angular

position in order to reach the destination.

4.4 Software details

In this section are analysed all the important details of the software, first of all is analysed the firmware of the LEGO agents, after how the communication and vision blocks work.

NXT Firmware

The firmware, developed in NXC, is the core of the robot and handles the motion. The robot receives a setpoint composed by the two wheels' angular speed to follow. In the firmware are also implemented two independent speed controllers for the two wheels based on a simple P.I. controller with an antireset-windup that avoid the drift of the integral component. The two motors are provided of two incremental encoders and the speed is obtained with a discrete derive of the position.
The function responsible of the motion control is the function setSpeed() that is here below.

```
1  void setSpeed(int wl,int wr)
2  {
3     int speed1,speed2,err1,err2,u1,u2;
4     int Ki, Kp;
5
6     \\ Controller Gains
7     Ki = 10;
8     Kp = 30;
9
10    \\ Actual angular speed
11    getSpeed(speed1,speed2);
12
13    \\ Error computing
14    err1 = wl - speed1;
15    err2 = wr - speed2;
16
17    \\ Integral component + antireset wind-up
18    integ1 = integ1 + err1/Ki;
19    integ2 = integ2 + err2/Ki;
20
21    if(integ1 > 100)
22       integ1 = 100;
```

```
23      if(integ2 > 100)
24          integ2 = 100;
25      if(integ1 < -100)
26          integ1 = -100;
27      if(integ2 < -100)
28          integ2 = -100;
29
30      \\ Control Computing
31      u1 = integ1 + err1/Kp;
32      u2 = integ2 + err2/Kp;
33
34      \\Actuation
35      OnFwd(OUT_A, u1);
36      OnFwd(OUT_B, u2);
37
38  }
```

From the function getSpeed() the algorithm receives the angular speed of the wheels and with this information is implemented a PI control with an antireset windup. Could be useful for debugging purposes shows the controller parameters information during the motion.
The function getSpeed() computes the angular speed of the two wheels simply reading the position using the incremental encoder in two different time instant and making a discrete derive. In order to avoid error in the speed could be useful reset the count of the encoder before starting a speed reading. Here there is the code.

```
1   int getSpeed(int &wr,int &wl)
2   {
3       int begin1,begin2,end1,end2;
4
5       \\ Reset of the encoder to avoid error in the count
6       ResetTachoCount(OUT_A);
7       ResetTachoCount(OUT_B);
8
9       \\   Encoder initial position
10      begin1 = MotorRotationCount(OUT_A);
11      begin2 = MotorRotationCount(OUT_B);
12
13      \\ Wait 500 ms
14      Wait(500);
15
16      \\ Encoder final position
```

```
17      end1 = MotorRotationCount(OUT_A);
18      end2 = MotorRotationCount(OUT_B);
19
20      \\ Speed estimation
21      wr = (end1-begin1)*2;
22      wl = (end2-begin2)*2;
23
24  }
```

Communication Block

The communication between the computer and the LEGO robots is realized using a NXT brick as Bluetooth Router. The computer sends the messages to the Router, via USB connection using the library provided by the RWTH Toolbox, that forwards all the messages received using the Bluetooth connection to the correct recipient. The maximum number of connections that the NXT Brick is able to handle is three, if there is the necessity to control more than 3 agents another brick must be used. We call MASTER the NXT Brick that works as bluetooth router and SLAVEs the agents.

Communication Protocol

In order to communicate correctly is necessary establish a simple protocol that allows at the robots to understand the PC packets. The computer with Matlab sends the command using strings and the syntax is this:

NA:**OA1**:**RS1**:**LS1**[.**OA2**:**RS2**:**LS2**.**OA3**:**RS3**:**LS3**]

Where the fields mean:

- NA: Number of Agents, it defines the number of agents involved in the communication. This number must be between 1 and 3.
- OAi:Operation referred to agent i, can take the values *G* or *S*, the first stands for GO and in this case the following two fields are mandatory, the second stands for STOP and in this case the values of others two fields are ignored, but they must be filled.
- RSi: Right Wheel Speed referred to agent i, it is considered only if the operation is equal G and it is the set point value for the left wheel angular speed of the robot, this must be integer because the robot doesn't support floating point numbers.

- LSi: Left Wheel Speed referred to agent i, it is considered only if the operation is equal G and it is the set point value for the left wheel angular speed of the robot, this must be integer because the robot doesn't support floating point numbers.

If we fill the first filed with one the following three fields are mandatory, otherwise if the first filed is equal to 2 o 3 the following 6 or 9 fields are mandatory.

PC to MASTER Communication

In order to communicate with the Master from the PC using the USB connection is important download and install the RWTH Toolbox for Matlab. The software written for Matlab is based on four functions: BTconnect, BTdisconnect and BTmove. The first function establishes the USB connection between the PC and the NXT Master Brick, the second handles the disconnection and release the resources in a correct way.
The third function receives as input the number of robots to control, the linear speeds, the angular speeds and the actions to take (the operations mentioned in the protocol explanation). Using the inverse kinematic of the robot (the wheelbase and the radius of the robots are known, equal and fixed for all the vehicles) the angular speeds of the wheels are computed and the string command obtained filling the packet is forwarded to the blutooth router through the usb connection.

```
1  function [] = BTmove(n_robot,command,lin_speed, ang_speed)
2
3      d = 0.135;          % Wheelbase dimension
4      r = 0.016;          % Wheel dimension
5
6      % Speed conversion deg -> rad
7      ang_speed = deg2rad(ang_speed);
8
9
10     % Compute the wheel angular speed from the linear and
11     % angular speed
12     cmd = sprintf('%d',n_robot);
13     for i=1:n_robot
14
15         w =  [1/r d/(2*r) ; 1/r -d/(2*r)]*[lin_speed(i);
               ang_speed(i)];
16         % wheel speed conversion rad -> deg
17         w = round(rad2deg(w));
18         cmd = strcat(cmd,sprintf('.%c:%d:%d',command(i),w(1),
               w(2)));
```

```
19      end
20
21
22      % Create and send the message to the robot
23      NXT_MessageWrite(cmd,0);
24
25  end
```

MASTER and SLAVE Firmware

The MASTER firmware waits packets from the computer and when it receives one, analyses the first field that is the number of robots involved in the communication. So in according to this number the firmware analyse the next fields and forward the submessages to the correct agents.
The SLAVE firmware also waits for a packet and when it receives one, first analyses the operation, if is "STOP" it waits for another action otherwise if is "GO" it proceeds to control the wheels.
Here there are two flows diagram that describe how the firmwares work.

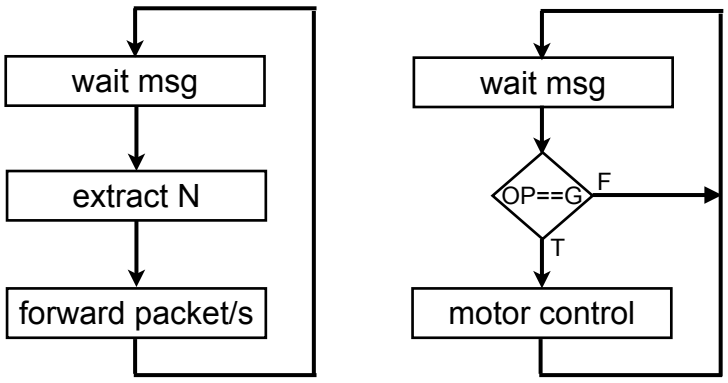

Figure 4.4: Logical Diagram of the NXT Firmware.

Vision Block

The vision software recognizes the position and the orientation of the agents. In order to solve the task on each robot is placed a marker. The marker is a red rectangle with a small blue square inlet. The software first of all searches the red objects in the image, it makes a crop around the markers and looks for the blue squares. When it knows the center of the red and blue squares it knows the position of the robot (the center of the red square) and the orientation is given by the angle formed by the line that pass for the two given points and

the horizon. The area of the markers is used to recognize different agents. Here there is an example of the marker.

The most important part in the vision software is the image filtering in order to use in the

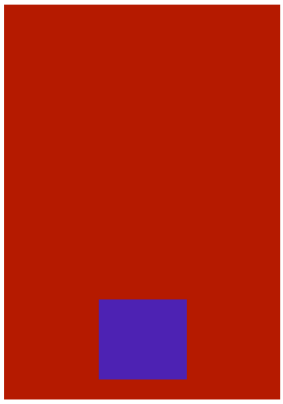

Figure 4.5: Example of a marker.

best way the MATLAB function regionprops to find the interesting objects. At the beginning the red component is subtracted to the gray scale frame in order to brought out the red parts. To reduce the noise is used a median filtering. After this the image is converted from color to a binary image and the threshold is computed by MATLAB using the function greythresth. As last operation all the objects that count less 20 pixels are removed because can be considered noise.

At this point calling the function regionprops the center of the markers and the box that contains each of them are found. The last information can be used to crop the image around the markers and repeating the filtering sequence, the blue squares that allow us to find the orientation of the agents could be found.

The image sequences here shows how the image is filtered.

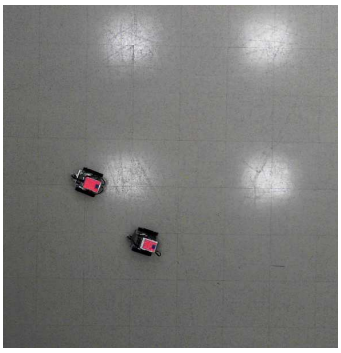

Figure 4.6: Original frame.

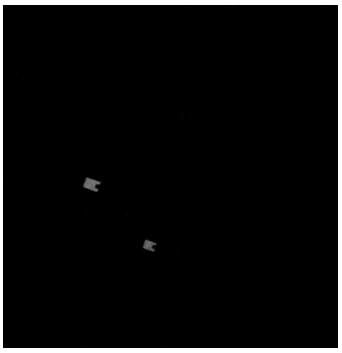

Figure 4.7: Frame after the median filtering.

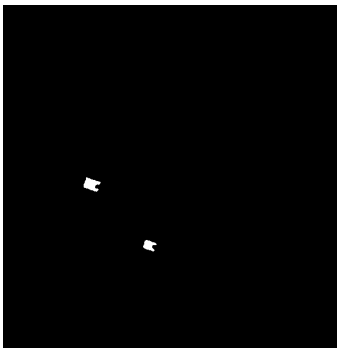

Figure 4.8: Frame ready for the regionprops function.

4.5 Experimental Results

The tests performed are the same of the simulations. They are divided in two parts, the first is the tracking with one or two agents and the second is the competitive (cooperative) dynamic.

Tracking Tests

In the first test there is a single agent that tracks a circular trajectory, the parameters of the receding horizon are $R = 0.1\mathbb{I}_2$, $P = \mathbb{I}_2$, $N = 3$ and the initial conditions are $[2.70, -1.25]m$. The results obtained are really good because the tracking error is really small and also the delay in knowing the circular trajectory is responsible of the the error.

The second tracking test is performed with two agents, the first tracks a fixed circular trajectory, the second tracks the first agent with a distance of 0.375m. The parameters used are for both the agents the same of the last test. he initial conditions are $[1.49, -1.05]m$ for the first and $[1.94, -1.18]m$ for the second.

Competitive and Cooperative Dynamics Tests

For this tests we tried different values of the parameters in order to validate the results obtained in the last chapter. We started using two agents with cooperative objective, the distances between them is setted to 0.75m. The receding horizon parameters are fixed for both the agents to $P = 0.5\mathbb{I}_2$ and $R = 20\mathbb{I}_2$. As we can see the agents reach a fixed final

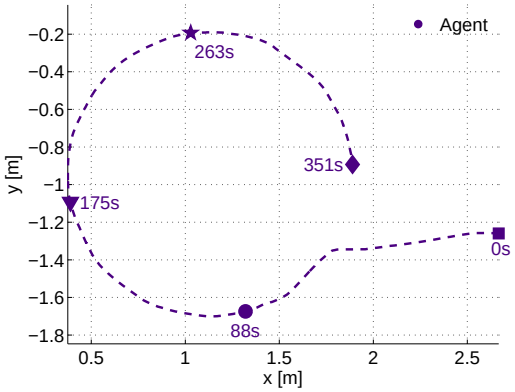

Figure 4.9: Tracking of a circular trajectory using RHC with parameters $R = 0.1\mathbb{I}_2$ and $P = \mathbb{I}_2$.

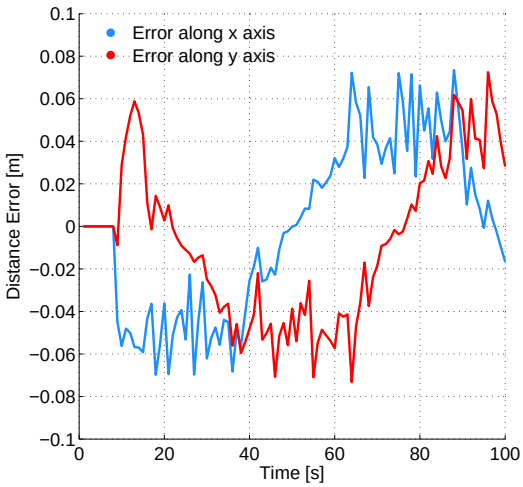

Figure 4.10: Distance errors between the two agents.

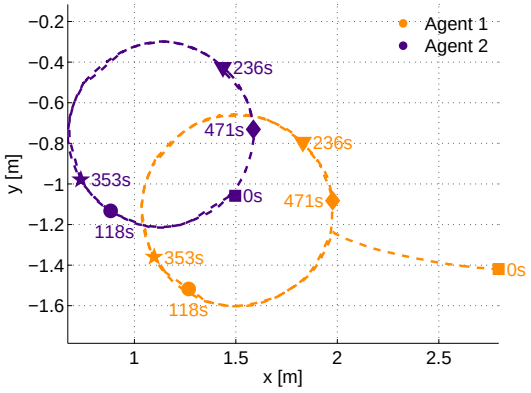

Figure 4.11: Position of the two agents.

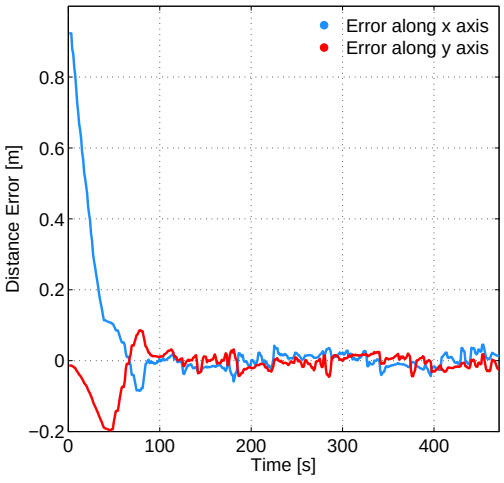

Figure 4.12: Tracking error of a circular trajectory known with a one step delay.

position (that depends by the initial conditions), and the distance error go to zero.

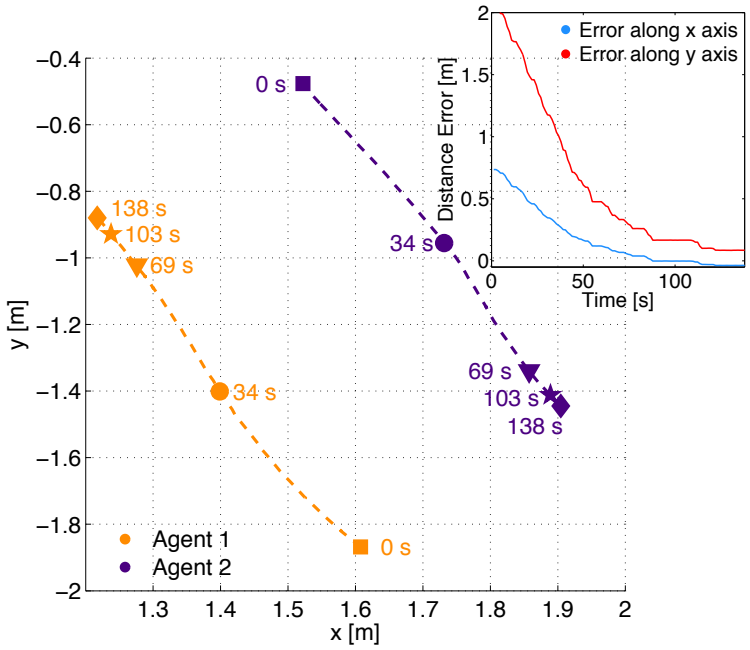

Figure 4.13: Plot error of the component x of the state s. Parameters: $P_1 = 0.5\mathbb{I}_2$,$P_2 = 0.5\mathbb{I}_2$, $R_1 = 20\mathbb{I}_2$ and $R_2 = 20\mathbb{I}_2$. A video of this experiment is available at [8], video m2.

The last three test are made with competitive dynamics. The objectives are changed to 0m for the first agent and to 0.75m for the second. Using the same receding horizon parameters of the last test we obtain that the distances between the agents is equal to 0.375m, that can be considered a kind of tie. Otherwise changing the parameters to $R_C = 10\mathbb{I}_2$, $R_R = 20\mathbb{I}_2$, and $P_C = P_R = 0.1\mathbb{I}_2$ we obtain that the first agent is an advantage, if we swap the R parameters the results is the opposite.
So the experimental results confirm the theory and the simulations. The initial conditions are the same of the simulations and the control horizon is still equal to three.

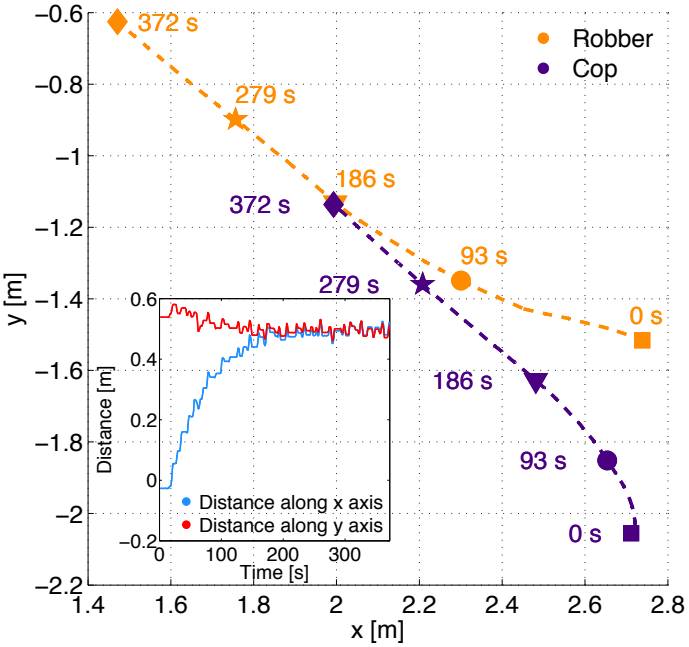

Figure 4.14: Plot error of the component x of the state s. The RHC cost function parameters are $P_C = 0.1\mathbb{I}_2$,$P_R = 0.1\mathbb{I}_2$, $R_C = 10\mathbb{I}_2$ and $R_R = 20\mathbb{I}_2$.

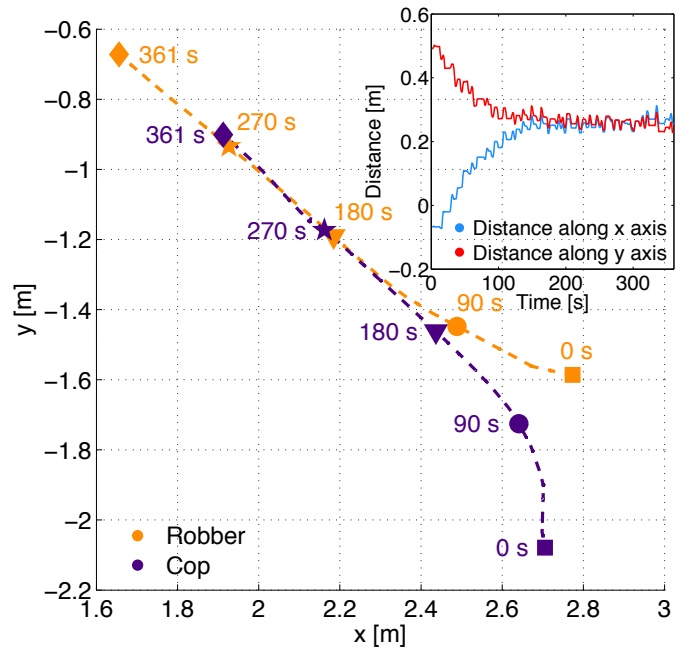

Figure 4.15: Plot error of the component x of the state s. Parameters are $P_C = 0.1\mathbb{I}_2$,$P_R = 0.1\mathbb{I}_2$, $R_C = 20\mathbb{I}_2$ and $R_R = 10\mathbb{I}_2$. A video of this experiment is available at the webpage [8], video m1.

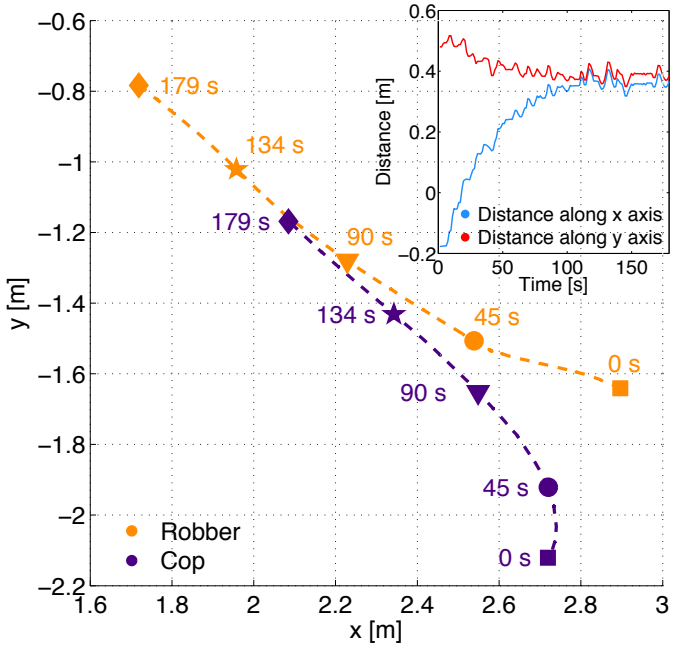

Figure 4.16: Plot error of the component x of the state s. Parameters: $P_C = 0.5\mathbb{I}_2$,$P_R = 0.5\mathbb{I}_2$, $R_C = 20\mathbb{I}_2$ and $R_R = 20\mathbb{I}_2$.

5
Appendix A

Theorem 5.0.1. *Let G and $G+H$ non singular matrices where H is a matrix of rank one. Let $g = Tr(HG^{-1})$. Then $g \neq 1$ and*

$$(G+H)^{-1} = G^{-1} - \frac{1}{1+g} G^{-1} H G^{-1} \tag{5.1}$$

Theorem 5.0.2. *Define $a_n = 0$ if $n < 0$ and assume that $a_0 \neq 0$.Let*

$$T_N = (a_{r-s})_{r,s=1}^{N}, \qquad N = 1, 2, \dots$$

be the family of lower triangular Toeplitz matrices generated by the formal power series

$$f(z) = \sum_{n=0}^{N} a_n z^n.$$

Calling g_n the coefficients of the inverse of the Toeplitz matrix, they are equal to:

$$g_n = \sum_{k=1}^{n} \frac{(-1)^k k!}{a_0^{k+1}} \sum_{k} \frac{1}{k_1 \dots k_n!} a_1^{k_1} \dots a_n^{k_n}$$

where the $\sum_k$ is over all partitions of k as a sum of non-negative integers $k_1 \dots k_n$ such that

$k_1 + k_2 + \ldots + k_n = k$ *and* $k_1 + 2k_2 + \ldots + nk_n = n$.

6
Appendix B

Theorem 6.0.3. *Given the linear system*

$$\begin{bmatrix} (Np+r) & (N-1)p & \dots & p \\ (N-1)p & ((N-1)p+r) & \dots & p \\ \vdots & & \dots & \vdots \\ p & p & \dots & (p+r) \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_N \end{bmatrix} = p \begin{bmatrix} N \\ N-1 \\ \vdots \\ 1 \end{bmatrix}$$

where $N \in \mathbb{N}$ and $p, q \geq \mathbb{R}^+$ then $\alpha_1 \leq 1$.

Proof. Using the Cramer's rule:

$$\alpha_1 = \frac{\det \begin{bmatrix} Np & (N-1)p & \dots & p \\ (N-1)p & (N-1)p+r & \dots & p \\ \vdots & & \dots & \vdots \\ p & p & \dots & p+r \end{bmatrix}}{\det \begin{bmatrix} Np+r & (N-1)p & \dots & p \\ (N-1)p & ((N-1)p+r) & \dots & p \\ \vdots & & \dots & \vdots \\ p & p & \dots & (p+r) \end{bmatrix}} = \frac{A(N)}{B(N)}$$

Using the induction principle:

Base case: for $N = 2$

$$\det(A(2)) = \det \begin{bmatrix} 2p & p \\ p & p+r \end{bmatrix} = p^2 + 2pr$$

$$\det(B(2)) = \det \begin{bmatrix} 2p+r & 1 \\ p & p+r \end{bmatrix} = p^2 + 3pr + r^2$$

So $\det(B(2)) > \det(A(2))$.

Inductive Step: if $\det(B(N-1)) > \det(A(N-1))$ hold then:

$$\det(A(N)) = \det \begin{bmatrix} Np & (N-1)p & \dots & p \\ (N-1)p & (N-1)p+r & \dots & p \\ \vdots & & \dots & \vdots \\ p & p & \dots & p+r \end{bmatrix} = Np \det(B(N-1)) + c(N,p,r)$$

$$\det(B(N)) = \det \begin{bmatrix} Np+r & (N-1)p & \dots & p \\ (N-1)p & ((N-1)p+r) & \dots & p \\ \vdots & & \dots & \vdots \\ p & p & \dots & (p+r) \end{bmatrix} = (Np+r) \det(B(N-1)) + c(N,p,r)$$

So $\det(B(N)) - \det(A(N)) = r \det(B(N-1)) \geq 0 \rightarrow \alpha_1 \leq 1$. ■

7
Appendix C

Theorem 7.0.4. *A second order real polynomial has all the roots on the unit circle if can be written in this way:*

$$f(x) = x^2 - 2\cos(\psi)x + 1$$

Where θ identify the angle of the roots.

Proof. We can write the second order polynomial with two roots in the unit circle in this way:

$$f(x) = (x - \theta - j\omega)(x - \theta + j\omega) \qquad \theta, \omega \in \mathbb{R}$$

Where the condition $\theta^2 + \omega^2 = 1$ holds because the roots are on the unit circle.
We also can define ψ like:

$$\psi = \begin{cases} \arctan(\frac{\omega}{\theta}) & \theta > 0 \\ \arctan(\frac{\omega}{\theta}) + \pi & \omega \geq 0 \quad \theta < 0 \\ \arctan(\frac{\omega}{\theta}) - \pi & \omega < 0 \quad \theta < 0 \\ \frac{\pi}{2} & \omega > 0 \quad \theta = 0 \\ -\frac{\pi}{2} & \omega < 0 \quad \theta = 0 \end{cases}$$

Now we can rewrite the polynomial, remembering the Euler representation of a complex

number $\theta + j\omega = e^{j\psi}$:

$$\begin{aligned} f(x) &= x^2 - (\theta - j\omega + \theta + j\omega)x + (\theta^2 + j\theta\omega - j\theta\omega + \omega^2) \\ &= x^2 - (e^{j\psi} + e^{-j\psi})x + (\theta^2 + \omega^2) \\ &= x^2 - 2cos(\psi)x + 1 \end{aligned}$$

■

8 Acknowledgement

This project would not have been possible without the support of several people who, in one way or another, contributed to its completion. I would like to express my gratitude to all those people to give me the possibility to complete this thesis.

I am deeply indebted to my supervisor, Elisa Franco,for all her advices, her presence and her guidance during all my time in Riverside. Elisa is an excellent teacher, a brilliant scientist and a very educated and witty woman.

Thanks to my Italian supervisor, prof. Luca Schenato, for his unbelievable patience, for answering to all my questions and doubts with the fastest and clearest answers.

I wish to acknowledge Alessandro Beghi that has given me the possibilities to go in California at the UCR.

Thanks to my best classmates during my Bachelor and Master: Diego, Alberto, Daniele and Diane. Especially for the patience when I was nervous or sad, and for always listening my never-ending problems. Without them I would never reached these results.

Thanks to Robin, Yelena and Adam, they took care of my mental sanity when I was in

Riverside, I will never forget all the funny moments spent together.

I want to say thank you to my ARCADE mates: Marco, Fabio, Franz, Fren, Livia and Gabriele for the incredible adventure. I am so happy that our friendship is not ended with (or like?) the experiment, even if I guess that it cannot survive to another joke.

It is mandatory mention Simone. He has always been able to stand and support me with patience and understanding. Without you everything would have been much more harder.

Last but not least, many thanks to my family: without their support, I should have never been where I am now; thank you, for having always struggled to give me the best. In particular I want to say thank you to my parents, Massimo and Giuliana, for all the advices, to my sister Valentina for the patience to correct my bad english, to my uncle Mario for the mail when I was in California and to my uncles Sergio and Marco for the nights spent together playing guitar.

References

[1] **Azevedo C., Poignet P., and Espiau B.** Moving horizon control for biped robots without reference trajectory. In *Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on*, 2002.

[2] **Beck R., Richert F., Bollig A., Abel D., Saenger S., Neil K., Scholt T., and Noreikat K.-E.** Model predictive control of a parallel hybrid vehicle drivetrain. In *Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC '05. 44th IEEE Conference on*, 2005.

[3] **Benedettelli D.** Matlab bluetooth router, 2009. URL http://robotics.benedettelli.com/BT_router.htm.

[4] **Borrelli F., Bemporad A., Fodor M., and Hrovat D.** An mpc/hybrid system approach to traction control. *Control Systems Technology, IEEE Transactions on*, may 2006.

[5] **Borri A., Bopardikar S. D., Hespanha J. P., and Benedetto M. D. D.** Hide and seek with directional sensing. *CoRR*, 2011.

[6] **Camacho E. F. and Bordons C.**, editors. *Model predictive control*. Springer-Verlag, 1999.

[7] **Camponogara E., Jia D., Krogh B., and Talukdar S.** Distributed model predictive control. *Control Systems, IEEE*, feb 2002.

[8] **Carron A.** Lego platform. Technical report, UCR, 2012. http://competitivedyanamics.altervista.org.

[9] **Clarke D. W., Mohtadi C., and Tuffs P. S.** Generalized predictive control - part i. the basic algorithm. *Automatica*, 1987.

[10] **Cutler C. and Ramakar B.** Dynamic matrix control. *Proceedings of the Joint Automatic Control Conference, San Francisco*, 1980.

[11] **Dunbar W. and Murray R.** Distributed receding horizon control for multi-vehicle formation stabilization. *Automatica*, 42(4):549–558, 2006.

[12] **Dunbar W. B. and Desa S.** Distributed model predictive control for dynamic supply chain management, 2005.

[13] **Franco E., Parisini T., and Polycarpou M.** Design and stability analysis of cooperative receding–horizon control of linear discrete–time agents. *International Journal on Robust and Nonlinear Control*, 17:982–1001, 2007.

[14] **Franz R., Milam M., and Hauser J.** Applied receding horizon control of the caltech ducted fan. In *American Control Conference, 2002. Proceedings of the 2002*, 2002.

[15] **Hespanha J., Kim H. J., and Sastry S.** Multiple-agent probabilistic pursuit-evasion games. In *Decision and Control, 1999. Proceedings of the 38th IEEE Conference on*, 1999.

[16] **Miller K. S.** On the inverse of the sum of matrices. *Mathematics Magazine*, 1981.

[17] **Projects N.** Multi-bot. URL http://www.nxtprograms.com/NXT2/multi-bot/index.html.

[18] **R. R. Bitmead M. G. and Wertz V.** *Adaptive optimal control: The thinking man's gpc.*

[19] **Richalet J., Rault A., Testud J., and Papon J.** Model predictive heuristic control: Applications to industrial processes. *Automatica*, 1978.

[20] **RWTH** . Mindstorm nxt toolbox, 2012. URL http://www.mindstorms.rwth-aachen.de/.

[21] **Shimizu T., Igakura T., Ishibashi R., and Kojima A.** A modeling of pedestrian behavior based on hybrid systems approach; an analysis on the direction of confluence. In *SICE Annual Conference (SICE), 2011 Proceedings of*, 2011.

[22] **Skogestad S.** Control structure design for complete chemical plants. *Computers and Chemical Engineering*, 2004.

[23] **Tapio S.** Arfit: Multivariate autoregressive model fitting, 2001. URL http://www.gps.caltech.edu/~tapio/arfit/.

[24] **Trench W. F.** Inverses of lower triangular toeplitz matrices. 1960.

[25] **Vidal R., Shakernia O., Kim H., Shim D., and Sastry S.** Probabilistic pursuit-evasion games: theory, implementation, and experimental evaluation. *Robotics and Automation, IEEE Transactions on*, 2002.