Robust Digital Control for Differential Soccer-Player Robots

João Monteiro* and Rui Rocha
a501020736@alunos.deec.uc.pt, rprocha@deec.uc.pt
ISR – Institute of Systems and Robotics
Department of Electrical and Computer Engineering
University of Coimbra, 3030-290 Coimbra, Portugal

***Abstract*— Robot soccer is a popular challenge due to its game dynamics. In particular, mobile robots must exhibit high responsiveness to motion commands and suitable pose control. This article presents a digital controller for pose stability convergence, developed to small-sized soccer robots. Special emphasis has been put on the design of a generic controller, which is suitable for any mobile robot with differential kinematics. The proposed approach incorporates adaptive control to deal with modeling errors and a Kalman filter which fuses odometry and vision to obtain an accurate pose estimation with high sampling rate. Experimental results validate the quality of the proposed controller.**

I. INTRODUCTION

This paper describes the work that is presently being made in the field of digital control and real time systems for differential mobile robots within the RACbot-RT M.Sc. project. The control method described herein is being applied on soccer-player robots endowed of an embedded computer, so that results can be obtained to fully validate the proposed method.

Many approaches for differential control of mobile robots have been proposed. For instance, [1] presents a generic controller where pose estimation is extracted from the robot's kinematics, and an adaptive control block is introduced to deal with modelling errors. In [3], a Lyapunov based nonlinear kinematic controller is presented where the influence of the control parameters is studied, without giving emphasis to modeling errors. The approach proposed herein brings together the simplicity of the Lyapunov mathematical laws, the adaptive control concept to deal with modeling errors and proper fusion of two sensorial data – vision and odometry – for robust pose estimation [4]. The project is divided into two main parts: 1) the digital controller, and 2) the real time system. Both parts have two phases: one dedicated to research aiming to find the best approach to be used, and next the implementation where simulations and on-the-field tests are made to validate the projected method as best as possible. The article covers only the first part.

II. PROBLEM FORMULATION

The development of theories in the field of robotic control is usually based on the extraction of a proper mathematical model of the hardware to control. Instead, this project emphasizes the minimization of the impact of modeling errors in control which, once implemented, can impose stability problems. In a mobile robot, these errors are cumulative with $t \to \infty$, resulting in evident pose mismatches. Therefore, the developed controller is focused on the kinematic stabilization of generic mobile robots possessing differential configuration, making more difficult the development of a suitable approach. Such effort was highly compensated in the performance observed in experimental tests. Few parameters representative of the physical structure need to be extracted; among them, there are mass, inertia, wheel radius and distance between wheels. Despite the mentioned efforts to avoid erroneous modeling, measurement errors can be present in the extracted parameters. Having this in mind, the concept of adaptive control is introduced, enhancing the robustness to unmodeled or poorly extracted parameters.

A. Robotic Soccer Players

The RAC[1] robot soccer team has five robot soccer players, each equipped with the following hardware: an embedded computer – the PC/104 –, a compact flash card reader, an FPGA programmed with HOSTMOT4 firmware, a suitable power supply, and an H-bridge driver for the two DC motors with installed encoders. During development, a laptop hard drive is incorporated on the robot's stack. Communication is possible via a wireless USB stick connected to the computer. In the rear of the robot, a castor wheel is mounted to provide proper physical balance. The software module that contains the controller – the system control module – is downloaded via SSH and compiled on the robot. Once running, it first connects with the *vision* and *brain* software modules. After the connections are established, the control module listens to instructions from the *brain*.

1) Command instructions: The possible instructions to be performed by the robots are *setVelocity*, *setTrajectory*, *Halt* and *Kick*. The *setVelocity* command is a package containing the desired velocity vector's angle and magnitude . To perform such command, the control module calculates the actual heading error and places virtual points referred to the world's coordinates in the direction of the vector. These virtual points will constitute the desired trajectory that the robot will perform to execute such vector with a velocity approximated to its magnitude. The *setTrajectory* defines target points on the field, each with an associated velocity vector. *Halt*, is a critical instruction having the effect of stopping the robot and aborting the execution of the previous instruction commands at any time.

B. System Control Module

This module is the principal subject of the present paper and its development was divided into two phases: project

[1]Robtica Acadmica de Coimbra – www.rac-uc.pt.vu

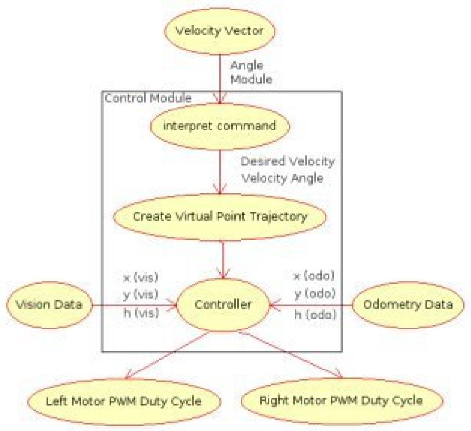

Fig. 1. Use case of the control module upon a *setVelocity* instruction receival.

and implementation. Due to its importance, Fig. 1 depicts a very high level representation of the control module in the presence of a *setVelocity* command. In the next subsections, the mathematical approach of the problem to be dealt by the control module will be stated.

1) Trajectory definition and Robot Kinematics: The studied 2D path planner defines a trajectory as a time variant pose vector represented in the playing field, which has its own global cartesian system defined. The robot's pose in the world coordinates frame contains three degrees of freedom (DOF), which are represented by the pose vector

$$q = \begin{bmatrix} x \\ y \\ h \end{bmatrix}, \tag{1}$$

where x, y are the robot's coordinates and h is its heading. The latter is assumed to be positive in counter-clockwise direction, beginning at the positive xx axis. The state q_0 is denoted as the zero pose state $(0, 0, 2n\pi)$, where n is an integer value. Since the robot is capable of moving in the world, the pose q is a function of time t. The integer representation of a set of points $(x(t), y(t))$ is the trajectory, and if the derivatives $\dot{x}$, $\dot{y}$ exist, $h(t)$ is no longer an independent variable, since

$$h(t) = tan^{-1}\frac{\dot{y}(t)}{\dot{x}(t)}, \tag{2}$$

from where one can see that the robot's orientation as a function of t depends of its velocity over each axis. The movement of the robot is controlled by its linear and angular velocities, v and ω respectively, which are also functions dependent of t. The robot's kinematics is defined by the following Jacobian matrix

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{h} \end{bmatrix} = \dot{q} = Jp = \begin{bmatrix} cos(h) & 0 \\ sin(h) & 0 \\ 0 & 1 \end{bmatrix} p, \tag{3}$$

where the velocity matrix is defined by

$$p = \begin{bmatrix} v \\ w \end{bmatrix} \tag{4}$$

This kinematics is common for all non-holonomic robots in which the number of controllable DOF's is less than the number of DOF's that the robot possesses.

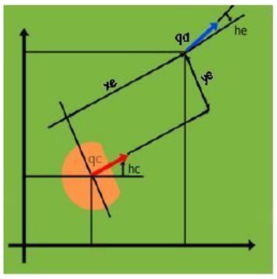

Fig. 2. Representation of q_c and q_d.

C. Pose Error

The main objective of any trajectory controller is to reduce as much as possible the pose error of the robot referred to a desired point. For the system to be implemented, two pose vectors need to be defined: the actual pose of the robot already represented in (1), and the other is the desired – or target – pose vector defined by

$$q_d = \begin{bmatrix} x_d \\ y_d \\ h_d \end{bmatrix}, \tag{5}$$

which, by definition, is the target pose that the robot should reach at the end of the movement. We will define the pose error q_e as the transformation of the reference pose q_d to the local coordinate system of the robot with origin (x_c, y_c), where the actual robot's heading is given by h_c's amplitude (Fig. 2). Such transformation is the difference between q_d and q_c,

$$q_e = \begin{bmatrix} x_e \\ y_e \\ h_e \end{bmatrix} = \begin{bmatrix} cos(h_c) & sin(h_c) & 0 \\ -sin(h_c) & cos(h_c) & 0 \\ 0 & 0 & 1 \end{bmatrix} .(q_d - q_c) \tag{6}$$

One can easily see that if $q_d = q_c$, the pose error is null, being this the ideal final state.

1) Robot dynamic model: Based on the Lagrange's mathematical modelation of mechanical systems on [1], and considering $G(q) = C(q, \ddot{q}) = 0$, the dynamic equations of the mobile robot can be written as

$$\begin{bmatrix} m & 0 & 0 \\ 0 & m & 0 \\ 0 & 0 & I \end{bmatrix} = \begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{h} \end{bmatrix} = \tag{7}$$

$$\frac{1}{R}\begin{bmatrix} \cos(h) & \cos(h) \\ \sin(h) & \sin(h) \\ L & -L \end{bmatrix} . \begin{bmatrix} \tau_1 \\ \tau_2 \end{bmatrix} + \begin{bmatrix} sin(h) \\ cos(h) \\ 0 \end{bmatrix} \lambda, \tag{8}$$

where τ_1 and τ_2 are the left and right motor torques respectively, m and I are the robot's mass and inertia, R is the wheel's radius, L is the line distance between the two wheels, and λ are the Lagrange multipliers of constrained forces. The non-holonomic restriction is deduced from the above equation, and is given by

$$\dot{x}\sin(h) - \dot{y}\cos(h) = 0, \tag{9}$$

from where it is imposed that a non-holonomic mobile robot can only move in the direction normal to the axis of the driving wheels.

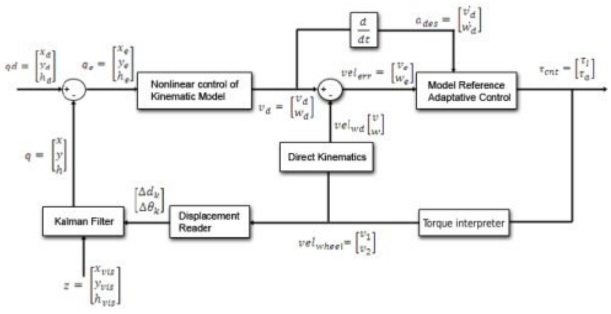
$qd = \begin{bmatrix} x_d \\ y_d \\ h_d \end{bmatrix}$
$q_e = \begin{bmatrix} x_e \\ y_e \\ h_e \end{bmatrix}$
Nonlinear control of Kinematic Model
$v_d = \begin{bmatrix} v_d \\ w_d \end{bmatrix}$
$\frac{d}{dt}$
$a_{des} = \begin{bmatrix} v'_d \\ w'_d \end{bmatrix}$
$vel_{err} = \begin{bmatrix} v_e \\ w_e \end{bmatrix}$
Model Reference Adaptative Control
$\tau_{cnt} = \begin{bmatrix} \tau_l \\ \tau_a \end{bmatrix}$
$vel_{wd} \begin{bmatrix} v \\ w \end{bmatrix}$
Direct Kinematics
$q = \begin{bmatrix} x \\ y \\ h \end{bmatrix}$
$\begin{bmatrix} \Delta d_k \\ \Delta \theta_k \end{bmatrix}$
Kalman Filter
Displacement Reader
Torque interpreter
$vel_{wheel} = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$
$z = \begin{bmatrix} x_{vis} \\ y_{vis} \\ h_{vis} \end{bmatrix}$

Fig. 3. Control scheme.

III. Digital Controller Design

The controller is designed in three parts. In the first, kinematic stabilization is achieved using nonlinear control laws. For the second, the acceleration is used for exponential stabilization of linear and angular velocities. The uncertainties related with the robot's physical structure modeled parameters are compensated using an adaptive control block. Introducing suitable Lyapunov functions, stability of the system's state variables is achieved. For the final part, pose estimation is made by fusing odometry and vision for robust pose feedback information by means of a Kalman filter. The latter was designed in a way that independence of the mathematical system's model is achieved, boosting the overall performance.

A. The control scheme

The developed approach is depicted in Fig. 3 . It's a feedback controller, in which the input state is the desired robot's pose $[x_d\ y_d\ h_d]'$. At its output, proper update of the torques for each wheel is done to fulfill the controller's objective of eliminating the pose error. The adaptive control block is present to guarantee that a stabilized condition is achieved independently of the presence of modeling errors. The estimation error is brought to zero in finite time. Next, the control blocks will be explained.

B. Pose error generator

The error dynamics is written independently of the inertial (fixed) coordinate frame by Kanayama transformation. Expanding (6), and considering the robot's heading as h, the following is given,

$$q_e = \begin{bmatrix} x_e \\ y_e \\ h_e \end{bmatrix} = \begin{bmatrix} \cos(h) & \sin(h) & 0 \\ -\sin(h) & \cos(h) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_d - x \\ y_d - y \\ h_e - h \end{bmatrix}, \tag{10}$$

which will compose the pose error vector.

C. Nonlinear Kinematic Controller

Lyapunov based nonlinear controllers are very simple and yet, at the same time, very successful in kinematic stabilization. So, bringing together the concepts simplicity and functionality, the Lyapunov stability theorem proved to be of great utility for this project. Let's consider, in the present case, the following Lyapunov candidate function that represents, as stated in [2], the total energy of the robot[3],

$$V = \frac{1}{2}(x_e^2 + y_e^2) + (1 - \cos(h_e)). \tag{11}$$

[3]The sum of the kinetic and potential energies.

To prove the stability condition, $\dot{V}$ needs to be obtained. Based on [1] and [3], the following is extracted,

$$\begin{aligned} \dot{V} = v_r x_e \cos(h_e) - v_d x_e + v_r \sin(h_e) y_e + \\ w_r \sin(h_e) - w_d \sin(h_e) \end{aligned}$$

$$\Leftrightarrow \dot{V} = v_r \cos(h_e - v_d) x_e + \sin(h_e)(v_r y_e + w_r - w_d),$$

where v_r and w_r are the reference velocities. v_d and w_d are chosen to make $\dot{V}$ become negative semi-definite:

$$v_d = v_r \cos(h_e) - x_e \tag{12}$$

$$w_d = w_r + v_r y_e + \sin(h_e). \tag{13}$$

Replacing these expressions in $\dot{V}$, the following is given,

$$\dot{V} = -x_e^2 - v_r \sin^2(h_e), \tag{14}$$

from where, one can easily see that $\dot{V}$ is always negative semi-definite. Let's take a specific instance of the control rule at this block's output,

$$v_d = v_r \cos(h_e) - K_x x_e \tag{15}$$

$$w_d = w_r + K_y v_r y_e + K_h \sin(h_e), \tag{16}$$

where K_x, K_y and K_h are positive constants. By La Salle's principle of convergence and proposition 1 of [3], the null pose state q_0 is always an equilibrium state if the reference velocity is higher than zero ($v_r > 0$). By this, there are three weighting constants for the pose error, without interfering in the overall pose stability of the robot.

D. Model Reference Adaptive Control

The motivation to include this block comes from the need of the controller to cooperate with parameter uncertainties. Based on [1], one can extract the adaptation rules for the linear velocity,

$$\frac{d\theta_1}{dt} = -\epsilon_1 e \dot{v}_d \Leftrightarrow \theta_1 = \int -\epsilon_1 e \dot{v}_d dt \tag{17}$$

$$\frac{d\theta_2}{dt} = -\epsilon_2 e \dot{v}_d \Leftrightarrow \theta_2 = \int -\epsilon_2 e \dot{v}_d dt. \tag{18}$$

Identically, one can find similar rules for the angular velocity:

$$\frac{d\theta_3}{dt} = -\epsilon_3 e' \dot{\omega}_d \Leftrightarrow \theta_3 = \int -\epsilon_3 e' \dot{\omega}_d dt \tag{19}$$

$$\frac{d\theta_4}{dt} = -\epsilon_4 e' \dot{\omega}_d \Leftrightarrow \theta_4 = \int -\epsilon_4 e' \dot{\omega}_d dt. \tag{20}$$

where v_d and w_d are the desired linear and angular velocities, and e and e' the respective velocity errors. The parameters ϵ_1, ϵ_2, ϵ_3 and ϵ_4 are manually tuned for best performance achievement. The practical implementation of this block is not simple, since it is necessary to determine the virtual velocity before proceeding with the actual real velocity's calculation, which in turn is needed to evaluate the error of the modeled parameters m and I. So, for better organization and comprehension, it was important to create a schematic for this block (Fig. 4), which will allow for a better understanding in the implementation phase. From this figure, one can easily see the virtual velocity's determination before evaluating the velocity error due to uncertainty of m and I.

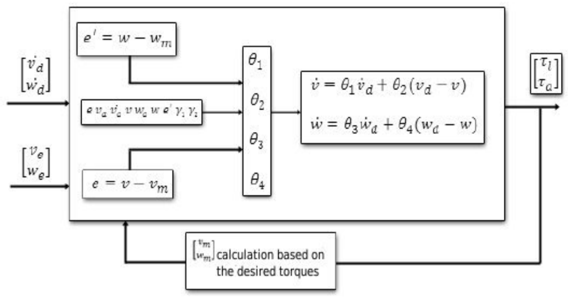
$\begin{bmatrix} \dot{v}_d \\ \dot{w}_d \end{bmatrix}$
$\begin{bmatrix} v_e \\ w_e \end{bmatrix}$
$e' = w - w_m$
$e\ v_d\ \dot{v}_d\ v\ w_d\ w\ e'\ \gamma_1\ \gamma_2$
$e = v - v_m$
θ_1 θ_2 θ_3 θ_4
$\dot{v} = \theta_1 \dot{v}_d + \theta_2 (v_d - v)$
$\dot{w} = \theta_3 \dot{w}_d + \theta_4 (w_d - w)$
$\begin{bmatrix} \tau_l \\ \tau_a \end{bmatrix}$
$\begin{bmatrix} v_m \\ w_m \end{bmatrix}$ calculation based on the desired torques

Fig. 4. Adaptive Control block scheme.

E. DC Motor Control based on the Desired Torques

From the controller scheme, the output consists of a torque vector but it is not explicit how the motors will be actuated in response to this torque command. The amplitude of the electric current that controls the motor's power for the RAC's soccer robots is regulated by the drive board. This board implements a H bridge to drive each motor individually by a PWM signal. Following the RAC's specific case, and accordingly with the datasheet of the used DC motors, the torque-current relation is given by,

$$I = k_i.\tau, \tag{21}$$

where k_i is the current constant and has a value of 0.487. For a motor (left or right) for which a rotor torque of τ_d is desired, the necessary current that the motor must consume, can be extracted as follows,

$$I_d = k_i.\tau_d \tag{22}$$

Referring to the datasheet, the maximum current value that the motor should consume is 0.57A, being the ideal value 0.35A. Therefore, the I_{MAX} reference value should be between 0.35 and 0.57, letting the hardware capabilities be fully used, but avoiding permanent overload. So, a value of 0.4 is admitted to be suitable and the maximum power consumption for the robot's 12VDC motors is calculated by

$$P_{MAX} = VI_{MAX} = 12 * 0.4 = 4.8W. \tag{23}$$

By assuming that the motor will not work above a maximum value of its active power of $P_{MAX} = 4.8W$, its safety is ensured and, at the same time, the robot takes advantage of its performance. It's also guaranteed that it will not overheat. Based on such value, the relation *duty cycle-current* can be determined. Let V_{ap} be the average voltage value applied to the motor as a consequence of a command *duty cycle* of DC, the following is given,

$$V_{ap} = \frac{DC}{100} * 12 \tag{24}$$

therefore, if with $12VDC$ the motor consumes a maximum of $0.4A$, the following is extracted,

$$I_{ap} = \frac{V_{ap}}{12} * 0.4. \tag{25}$$

Bringing together the two above equations, the following is given,

$$DC = k_{ap} I_{ap}, \tag{26}$$

where $k_{ap} = \frac{1}{0.4}.100$. If the *torque-current* relation is applied, the useful *duty cycle-torque* relation can be extracted,

$$DC = k_{ap}.k_i \tau_{ap}. \tag{27}$$

Based on this equation, it is possible to know by a simple calculation what value should be placed in the FPGA's *duty cycle* register for a motor that is wished to have τ_{ap} torque.

F. Binary Interpreter

This block makes proper measurement of the actual velocity based on the encoder displacement during a sample time. First, a measurement of the encoder count is made and t_{samp} time is waited. After this short time (the shortest possible), a new measurement is sampled and the counting is made finding the difference of the previous and last pulse counter value. Given that the used motors possess a velocity box of 1:3.71, and that the encoder is capable of producing 512 pulses, in a row there are 4*512 transactions when the encoder reading is in quadrature mode (which is defined in the drive board). The angular increment of the encoder is $\frac{2\pi}{2048} rad$ referred to the motor. Referencing to the wheel, this result is $\frac{2\pi}{2048x3.71} rad$. By this, the wheel's velocity in rpm is,

$$v_{wheel} = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}, \tag{28}$$

where $v_1 = v_2 = \frac{count}{t_{samp}.1000} \cdot \frac{1}{2048x3.71}.60$.

G. Direct Kinematics

We need to know the linear and angular velocities of the platform to evaluate the robot's velocity error at a given moment. Therefore, it is useful to extract its direct kinematics given the wheel velocities. The resulting vector is, as one can see in the controller's scheme, vel_{wd}. From this, the following equations for linear and angular platform's velocity are given,

$$v = \frac{v_1 + v_2}{2} \ \ ; \ \ w = \frac{v_1 - v_2}{2.r_b}, \tag{29}$$

where r_b is the wheel radius.

H. Displacement reader

Due to the fact that the implemented Kalman filter needs a specific input vector of Δd_k and Δh_k, which are the linear and angular displacement respectively, it is necessary to calculate these values for each loop,

$$\Delta d = \frac{d_1 + d_2}{2} \ \ ; \ \ \Delta h = \frac{d_1 - d_2}{b}, \tag{30}$$

where b is the distance between wheels and d_1 and d_2 are the distance in meters "walked" by the left and right wheels respectively.

I. Kalman Filter

A differential robot with odometry system is equipped with an encoder in each motor. An angular displacement of α radians on the rotor corresponds to a performed distance d on the periphery of the wheel, and subsequently to an encoder count. The distance is given by $d = k\alpha$ with $k = \frac{1}{r}$, where r is the wheel radius. If the robot's movement is assumed to be linear, the distances d_1 and d_2 performed by the left and right wheels respectively, can be transformed in linear and angular displacements by (30). For a particular sample instant, the following is given:

$$\Delta d_k = \frac{d_{1,k} + d_{2,k}}{2} \;\; ; \;\; \Delta h_k = \frac{d_{1,k} - d_{2,k}}{b}. \tag{31}$$

The robot's coordinates referenced on the world's coordinates can be determined by the following equations:

$$X_{k+1} = X_k + \Delta d_k \cos(h_k + \frac{\Delta h}{2}) \tag{32}$$

$$Y_{k+1} = Y_k + \Delta d_k \sin(h_k + \frac{\Delta h}{2}) \tag{33}$$

$$h_{k+1} = h_k + \Delta_k. \tag{34}$$

These coordinates constitute the state vector, and are observed by the vision coordinate vector z. These measurements can be described as a nonlinear function c of the robot's coordinates, which possesses an independent noise vector v. Defining the above equations as vector α and placing Δd_k and Δh_k in an input vector u_k, the robot can be modeled by the following equations

$$x_{k+1} = a(x_k, u_k, w_k, k) \tag{35}$$

$$z_k = c(x_k, v_k, k), \tag{36}$$

where $w_k\tilde{}N(0, Q_k)$ and $v_k\tilde{}N(0, r_k)$, being both not correlated, i.e., $E[w_l v_l^T] = 0$.

We can now design the extended Kalman filter, using the odometry-based system model:

$$\hat{x}_{k+1} = a(x_k, u_k, w_k, k) \tag{37}$$

$$P_{k+1} = A_k P_k A_k^T + Q_k \tag{38}$$

$$K_k = P_k C_k^T [C_k P_k C_k^T + R_k]^{-1} \tag{39}$$

$$\hat{x}_k = \hat{x}_k + K_k[z_k - C_k \hat{x}_k] P_k = [I - K_k C_k] P_k \tag{40}$$

The process noise is modeled by two Gaussian white noises applied on the two odometry displacement measurements Δd_k and Δh_k.

J. Filter simulation

In the developed simulation, when a voltage is present at the input of each motor transfer function, a displacement is produced at its output, making it possible to simulate the robot's movement. In this case, and because there is no feedback loop since only the evaluation of the filter's performance is desired, the output will rise indefinitely in a linear form, except at the start phase of the motor. We want to observe the behavior of the filter in the presence of vision and odometry noise, analyzing the estimated state variables.

– Filter test case: Robot in $x = 0$, $y = 0$, $heading = 0$, $\sigma_v^2 is = 1$, $\sigma_o^2 do = 1$

The displacement made by the robot will be indefinitely linear along the xx axis, being the displacement over yy and the robot's heading equal to zero. Fig. 5 shows this situation, being the blue slope the displacement over xx, the red slope the displacement over yy and the green one the robot's heading. The magenta slope represents the vision error. As one can see, the filter possesses little but visible sensitivity to vision noise, contrary to what happens with odometry noise. In Fig. 6, one can see a screenshot of the visualizer tool developed for the controller module. The robot accurately goes to

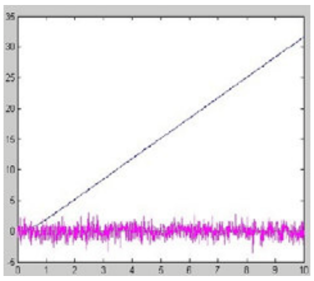
Fig. 5. Filter test case results.

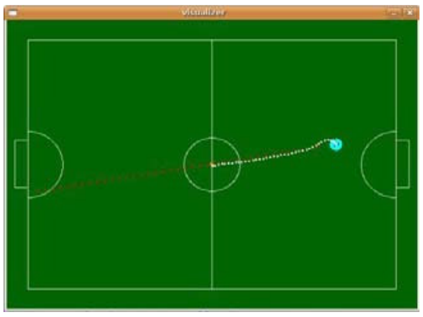

Fig. 6. *SetPoint* command to $x = 0, y = 0$.

the defined *setPoint*, but possessing a yy axis precision error of 1 to 2 cm maximum. This precision error exists because of two main causes. The first is the backward force exerted by the energy cable that feeds the robot in test environment[4]. The second is because of the defined tuned parameter for the influence of the robot's error over the yy coordinate – K_y. Tuning for near-zero error is possible but leads to a very *hard* control scheme in the presence of a physical disturbance, making the controller produce high overshoot for compensation. Since the RAC soccer robot is destined to walk on a field where collisions with other robots may be present, the revealed accuracy perfectly suits the team needs. For collision-free applications, where minimum physical errors exist and depending on the world's space, the control can be made *harder*, raising K_y.

IV. On-the-field results

– Setpoint command to (0,0)

For this test, a *setPosition* command to $(0, 0)$ is sent.

– setVelocity with v = 0.3mps and desired velocity angle = 0

For this test, the robot was subjected to extreme noise conditions. Referring to fig. 7, the robot is subjected to two disturbances done by blocking its left wheel, evident by the multiple white dots in the same place. Also, the colored pattern, which is placed on the robot to track its pose through the vision module, was blocked for a while so that no vision data was being received by the controller for it to estimate the actual pose. Controller's robustness is proved.

– Sequence of setVelocity commands with $v = 0.3mps$

[4]During tests, it is preferred to have the robot constantly fed with energy instead of placing the batteries that discharge with time.

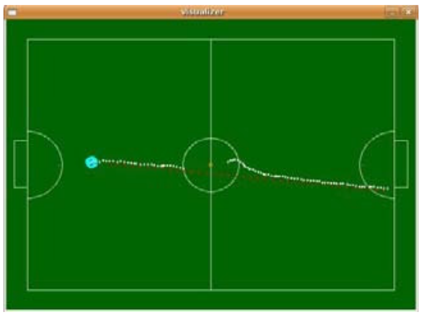

Fig. 7. *setVelocity* with $v = 0.3mps$ and desired $velocity\ angle = 0$.

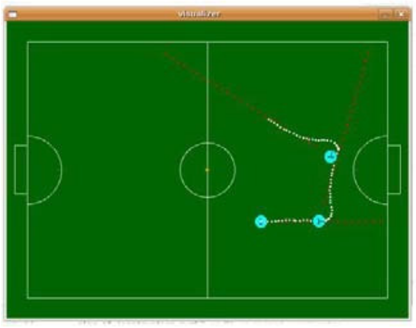

Fig. 8. Sequence of *setVelocity* commands with $v = 0.3mps$ and $velocity\ angle = 1.3rad$.

and $velocity\ angle = 1.3rad$

In this final test (Fig. 8), a sequence of *setVelocity* commands is sent to evaluate the control module's response in the presence of new velocity instructions. This test's characteristics are more realistic under the robot soccer game, where a high movement dynamic is required for the robot. As one can conclude, the robot accurately executes the performed commands, evidencing the software module's robustness.

V. CONCLUSION

A controller for pose error elimination of a soccer-player robot was projected and its practical results have been shown. In the theoretical formulation of the controller, particular emphasis was given to devise a generic control scheme, so as to be robust against errors in the estimation of the robot dynamic parameters. On-the-field tests reveal that the projected approach is not only valid, but also robust. It allows the robot to correct its trajectory, making it converge to the desired pose.

A real-time system for the robot is now under development. Studies have been made to find the best approach to schedule properly the sub-tasks carried out by the robot when executing motion commands. It is important that the scheduler makes hard tasks meet their deadlines, and also provide fast average response times for tasks with soft deadlines. This effort will boost the robot's response capability, improving the global system's performance.

REFERENCES

[1] A. Gholipour, M. J. Yazdanpanah,"t Dynamic Tracking Control of Nonholonomic mobile robot with model reference adaptation for uncertain parameters", University of Tehran, 2000.

[2] M. Vidygascar, *Nonlinear Systems Analysis*, Prentice Hall, New Jersey, NJ; 1993.

[3] Y. Kanayama, Y. Kimura, F. Miyazaki, Tetsuo Nogushi, "A Stable Tracking Control Method For An Autonomous Mobile Robot", cH2876-1/90/oooO/0384$01, 1990 IEEE.

[4] T. Larsen, M. Bak., N. A. Andersen, O. Ravn,"Location Estimation for an Autonomously Guided Vehicle using an Augmented Kalman Filter to Autocalibrate the Odometry", Technical University of Denmark, 2000.

[5] A. M. Bloch, M. McClamroch, "Control and Stabilization of Non-holonomic Caplygin Dynamic Systems",England, 1991.

[6] D. Wang, Guangyan Xu, "Full State Tracking and Internal Dynamics of NonHolonomic Wheeled Mobile Robots", Proceedings of the American control Conference, June, 2000

[7] A. Martinelli, R. Siegwart, Estimating the Odometry Error of a Mobile Robot During Navigation, Autonomous Systems Lab, Swiss, Federal Institute of Technology Lausanne (EPFL), 1996.