USING A REAL-TIME OPERATIONAL SYSTEM TO EMBED A KINEMATIC CONTROLLER IN AN OMNIDIRECTIONAL MOBILE ROBOT

Diego Stéfano F. Ferreira*, Cristiane C. Paim*, Augusto Loureiro da Costa*

*Robotics Laboratory,
Postgraduate Program in Electrical Engineering
Federal University of Bahia, 40210-630 Salvador, Bahia, Brazil

Emails: diego.stefano@gmail.com, cpaim@ufba.br, augusto.loureiro@ufba.br

Abstract— In this paper, a kinematic controller is designed and embedded on a omnidirectional robot. The robot has several sensors from which it has to acquire data, and then a real-time operational system was used to accomplish concurrency requirements. The robot hardware has a microcontrollers network, but only one of the nodes is used for the purposes of this paper, the PSoC 5LP, based on an ARM Cortex-M3 processor. The embedded software is described in terms of the tasks implemented and their scheduling by the real-time operational system. Practical results obtained from tests using the described system are presented.

Keywords— Embedded control systems, real-time operational systems, omnidirectional mobile robotics, kinematic control, task scheduling.

1 Introduction

A mobile robot embedded software, in order to allow successful performance of the robot, must cope with various kinds of activities, such as control of the robot actuators, processing of sensor readings and communication. These activities are time-critical and should be performed ideally at the same time. But due to computational limitations, this perfect parallelism is almost never achieved in real world applications. The control of the robot's movement should, thus, be developed with these computational restrictions in mind.

This work deals with the implementation of a kinematic controller for an omnidirectional robot, which must also perform sensor readings concurrently. The first section will describe the robot used in the experiment, followed by a section that describes its kinematic model. Then the kinematic controller is described, and in the next section the behavior of the embedded software with the real-time operational system is described. Some results of experiments are described, and finally the conclusions drawn based on these results are presented.

2 The Omnidirectional Robot AxeBot

The kinematic controller described in this work was embedded in the omnidirectional robot AxéBot. It consists of three swedish wheels whose axes are physically displaced 120° from each other. The swedish wheels and a schematic representation of its disposition are illustrated in Fig. 2.

The actuators used are the Maxon A-max $6V$ DC motors. They have a 16 counts-per-revolution quadrature encoder and a 19:1 planetary geartrain with ball bearings built in, with a $4mm$ output shaft. To drive these motors, three $3A$ low-voltage H-bridges from Acroname were used.

A network consisting of three microcontrollers composes the computational nucleus of the AxeBot. The three microcontrollers are:

- the **DIL/NetPC 2486**, from SSV Embedded Systems, with a Vortex86SX 300MHz processor and 64MB SDRAM;
- the **mbed**, based on the microcontroller NXP LPC1768 which, in turn, is based on an ARM Cortex-M3 processor; and
- the Cypress Semiconductors' **PSoC 5LP**, also based on an ARM Cortex-M3 core.

The network is heterogenous in the sense that it uses two different communication protocols between its nodes: a CAN (Controller Area Network) network connects mbed and PSoC, and an Ethernet network connects mbed and DNP 2486 (Figure 1). The network was so designed for the embedding of the concurrent autonomous agent described in (da Costa and Bittencourt, 1999).

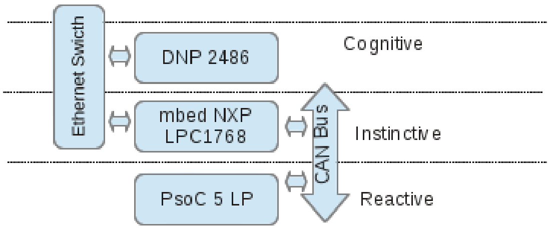

Figure 1: AxeBot's Microcontrollers Network.

The Cypress Programmable System-on-Chip (PSoC) 5LP is the computational unit of the AxeBot robot where the kinematic controller described in this paper was embedded. The main features of the PSoC 5LP, as exposed by (Fosler, 2012), are summarized in Table 1.

The AxeBot is also equipped with eight Sharp GP2D02 infrared distance sensor, one Dimension

Table 1: PSoC 5LP main features

Feature	**PSoC 5LP**
Core	ARM Cortex-M3 1.25 DMIPS/MHz
Flash	256KB
SRAM	64KB
EEPROM	2KB

Engineering DE-ACCM5g two axis accelerometer and one Devantech CMPS03 Magnetic Compass Module. The GP2D02 uses a two-wire digital synchronous interface method to communicate its readings, the DE-ACCM5g uses two analog channels (one for each axis) and the CMPS03 uses a I2C interface.

3 Kinematic Model

The kinematic model for the AxéBot was developed based on the coordinate systems shown in Fig. 2.

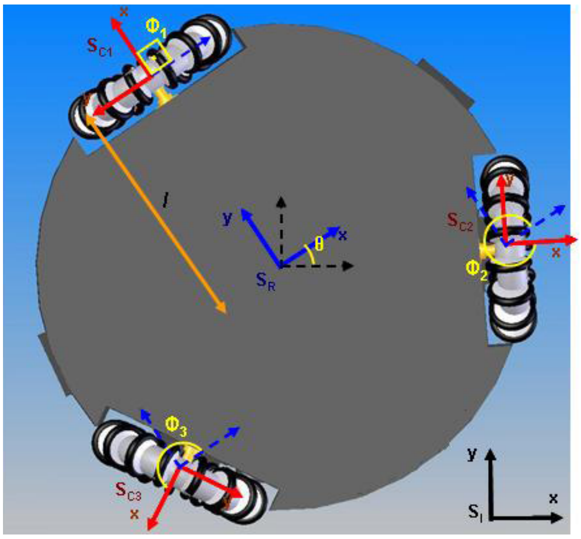

Figure 2: Coordinate systems for kinematic modelling.

For a complete derivation of the AxeBot's kinematic model, one should refer to (Bitencourt et al., 2008). Here only the final expression of the model is presented: in Equation (1) the matrix od the kinematic model is shown and then used in Equation (2) to form the complete model.

$$P(\theta) = \begin{bmatrix} -\frac{2\cos\theta}{3} & \frac{\sqrt{3}\cos\theta - 3\sin\theta}{3\sqrt{3}} & \frac{\sqrt{3}\cos\theta + 3\sin\theta}{3\sqrt{3}} \\ -\frac{2\sin\theta}{3} & \frac{\sqrt{3}\sin\theta + 3\cos\theta}{3\sqrt{3}} & \frac{\sqrt{3}\sin\theta - 3\cos\theta}{3\sqrt{3}} \\ \frac{1}{3l} & \frac{1}{3l} & \frac{1}{3l} \end{bmatrix} \tag{1}$$

$$\begin{bmatrix} v_{xI} \\ v_{yI} \\ \dot{\theta} \end{bmatrix} = P(\theta) \begin{bmatrix} \dot{\phi}_{1x} \\ \dot{\phi}_{2x} \\ \dot{\phi}_{3x} \end{bmatrix} . \tag{2}$$

In Equation (2), v_{xI}, v_{xI} and $\dot{\theta}$ represents, respectively, the velocities in x and y directions and the angular velocity of the robot's center of mass in the S_I coordinate system, $\dot{\phi}_{ix}$ represents the angular velocity of the wheel i relative to the x axis of the S_{Ci} coordinate system, and R is the radius of the wheels.

4 Kinematic Position Controller

Once one has the kinematic model of the robot it is possible to use this model to design a controller to the robot that will allow it to stabilize on some given position or follow some given trajectory. This controller is the kinematic position controller, and the mentioned variants leads to its two types: a point-to-point controller and a trajectory tracking controller.

4.1 Point-to-Point Kinematic Controller

The input to the controller in this case is the desired (set-point) position. It then calculates the difference vector between the desired or reference position (x_I^R, y_I^R and θ^R), and the actual position of the center of the robot (x_I, y_I and θ), which is the error vector given in Equation (3).

$$\mathbf{e}(t) = \begin{bmatrix} x_I^R \\ y_I^R \\ \theta^R \end{bmatrix} - \begin{bmatrix} x_I(t) \\ y_I(t) \\ \theta(t) \end{bmatrix} \tag{3}$$

This error vector is used to calculate a Proportional-Integral (PI) control law. For point stabilization control, according to (Tsai et al., 2005), the control law is given by Equation (4), where K_P and K_I are symmetric and positive definite $3x3$ matrices.

$$\mathbf{u}(t) = K_P\,\mathbf{e}(t) + K_I \int_0^t \mathbf{e}(\tau)\,d\tau \tag{4}$$

4.2 Trajectory Tracking Kinematic Controller

For the trajectory tracking control, the reference vector $\mathbf{p}^T(t) = \begin{bmatrix} x_I^R(t) & y_I^R(t) & \theta^R(t) \end{bmatrix}$ is now dependent of time and describes the parameterized expression of the reference trajectory.

$$\mathbf{e}(t) = \mathbf{p}(t) - \begin{bmatrix} x_I(t) \\ y_I(t) \\ \theta(t) \end{bmatrix} = \begin{bmatrix} x_I^R(t) \\ y_I^R(t) \\ \theta^R(t) \end{bmatrix} - \begin{bmatrix} x_I(t) \\ y_I(t) \\ \theta(t) \end{bmatrix} \tag{5}$$

The control law is modified by adding de derivative of the reference vector, as it is shown in Equation (6).

$$\mathbf{u}(t) = K_P\,\mathbf{e}(t) + K_I \int_0^t \mathbf{e}(\tau)\,d\tau + \dot{\mathbf{p}}(t) \qquad (6)$$

The control action vector is then applied in the (2) of the kinematic model in order to obtain the wheels desired velocities through Equation (7).

$$\begin{bmatrix} \dot{\phi}_{1x} \\ \dot{\phi}_{2x} \\ \dot{\phi}_{3x} \end{bmatrix} = \frac{1}{R}\,P^{-1}(\theta)\,\mathbf{u}(t). \qquad (7)$$

4.3 Low level controllers

The wheels velocities just obtained with Equation (7), in turn, corresponds to the set-points of another controller, but in a lower level. This is the actuators control: the velocities of the wheels must be adjusted to the given value in order to correct the position error. Thus, the low level control deals with the velocity control of the motors. This control was described and successfully implemented for AxeBot's actuators in (Ribeiro, 2010) and (Ribeiro et al., 2011), and this work will use those results, in which a PI controller was applied.

Now, with the help of the motors' encoders, the actual velocity of the wheels are read and applied back in Equation (2) to obtain, by integration of the result, the new actual position of the robot's center of mass in the global reference frame. These, in turn, are used to compute a new error vector and continue the loop.

The kinematic controller consists, thus, of two loops: one external or high-level loop, in which the kinematic relationships are calculated, and a internal or low-level loop, responsible for the control of the actuators, as shown in Figure 3.

5 Embedding the Kinematic Controller

In the previous section the kinematic controller was described, and now this section will deal with the software architecture requirements for the adequate embedding of such a controller.

As was said before, the AxeBot omnidirectional robot computational unit should accomplish a series of tasks in order to perform properly. It has to:

- **acquire from the distance sensor**, which requires, at each reading, sending a logic low-level during 70ms to the sensor, followed by a pulse train of total length not greater than 10ms to obtain at each pulse falling edge one of the eight bits of the distance reading;
- **acquire from the accelerometer**, which requires a analog to digital conversion at each reading;
- **acquire from the magnetic compass** through I2C communication;
- **send data through UART to a computer** in order to supervise the system;
- **perform kinematic control**, which is made in a cycle (loop) with a fixed period T_C; and
- **perform motors velocity control** using a control loop with a period of $T_M \leq T_C$.

An illustration of the overall system and its components is given in Figure 4.

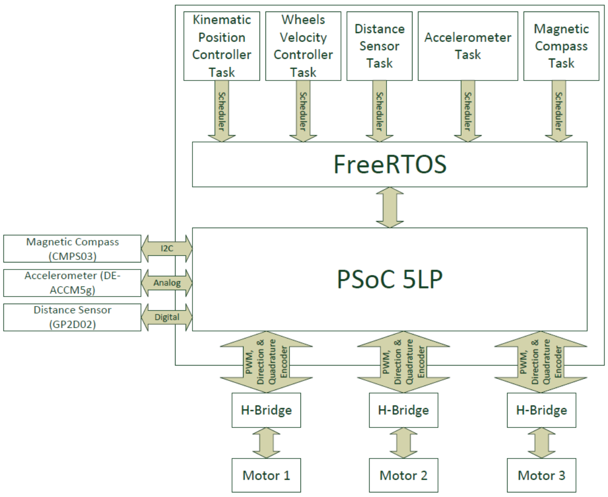
Kinematic Position Controller Task
Wheels Velocity Controller Task
Distance Sensor Task
Accelerometer Task
Magnetic Compass Task
Scheduler
Scheduler
Scheduler
Scheduler
Scheduler
FreeRTOS
Magnetic Compass (CMPS03)
Accelerometer (DE-ACCM5g)
Distance Sensor (GP2D02)
I2C
Analog
Digital
PSoC 5LP
PWM, Direction & Quadrature Encoder
PWM, Direction & Quadrature Encoder
PWM, Direction & Quadrature Encoder
H-Bridge
H-Bridge
H-Bridge
Motor 1
Motor 2
Motor 3

Figure 4: Illustration of the overall system.

In this context, multitasking is an indispensable requirement, and a Real-Time Operating System (RTOS) may fulfill it, since it is capable of managing multiple tasks through a synchronization mechanism involving task prioritization and scheduling, providing an abstraction layer in the software design (Stankovic and Rajkumar, 2004).

The RTOS used in this work was the FreeRTOS which, as the name suggests, is free and has a porting for the PSoC 5LP. Besides, it is written in the C programming language, the same used by PSoC Creator, the IDE for PSoC 5LP development.

In the FreeRTOS the tasks may be in one of four states: *running*, in which it is executing; *blocked*, in which it is not able to run until some event occurs; *ready*, in which it is not executing, but is able to; and *suspended*, in which it is not running and it is not able to run. The corresponding state transitions diagram is shown in Figure 5. All tasks are created in the *ready* state, and the task scheduler decides whether or not it can be in the running state based on its priority: if some higher priority task is already running, the task in

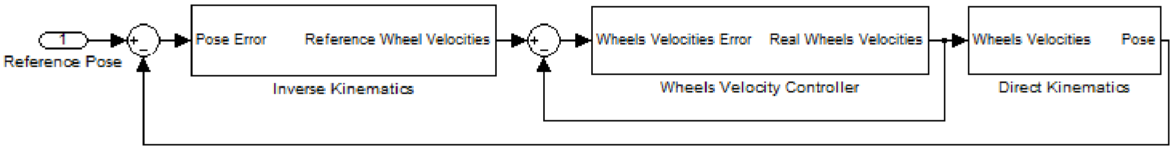

Figure 3: Block Diagram of the controller.

the ready state waits until it gets blocked or suspended; if the running task has a lower priority than the task in the ready state, then the former is preempted by the last (Barry, 2010).

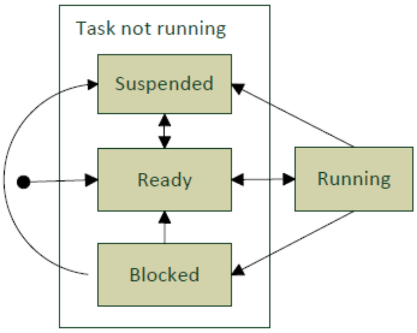

Figure 5: FreeRTOS tasks state diagram.

The tasks created for the AxeBot are shown in Table 2. In terms of the controller, the priorities were set in order to allow the loops to perform correctly, i. e., the priority of the kinematic controller task should be higher than the priority of the motor control task, in such a way that after each iteration of the kinematic controller it blocks and allows the motor controller to perform its loop $\frac{T_C}{T_M}$ times. After a time interval of T_C the kinematic controller preempts it and execute one new iteration.

Table 2: Tasks implemented.

Task	**Priority**
Kinematic Control	3
Motor Control	2
Compass Acquisition	1
Distance Sensor Acquisition	3
Accelerometer Acquisition	1

The timing of both the control loops were provided by timers interrupts from PSoC and the synchronization with the corresponding tasks was achieved through the use of semaphores (Barry, 2010; Stankovic and Rajkumar, 2004). After every T_C period the kinematic controller task receives a semaphore from the corresponding timer interrupt and preempts any other task that is in the running state. It then executes its code and goes to the blocked state waiting for the semaphore from T_C timer interrupt. The same occurs with the motor control task and the T_M timer. But the execution time of these iterations are generally less than both T_C and T_M. Then the remaining time is took by the remaining tasks.

For the distance sensor, it is made with a priority as high as the kinematic controller because when it is called first it just sends the digital ”0” to the sensor input and then block for $70ms$ (more than enough for the other tasks to execute) but when this time is up, the task must execute immediately, obeying the synchronous restrictions of the sensor, to acquire to sensor reading, which takes about $3ms$. Then the other sensors acquire within the aforementioned $70ms$ in between the iterations of the controllers.

6 Results

The kinematic controller was implemented with $T_C = 50ms$ and $T_M = 10ms$. It means that after each iteration the low-level controllers will perform at most five iterations, which is enough for it to stabilize (Ribeiro et al., 2011; Ribeiro, 2010; Santos, 2010). These controllers are discrete PI controllers, with proportional and integral constants equal to $k_p = 0.239$ and $k_i = 0.051$, respectively (Ribeiro et al., 2011; Ribeiro, 2010).

For the high-level controller, the matrices K_P and K_I are given by $K_P = 3\,I_{[3x3]}$ and $K_I = 0.002\,I_{[3x3]}$, where $I_{[3x3]}$ is the $3x3$ identity matrix (Tsai et al., 2005).

In Figure 6 the scheduling of the tasks, according to Table 2 and the previously described, is illustrated.

Two tests were performed: one for point stabilization control and another for trajectory tracking control.

6.1 Point Stabilization

As was said before, in point stabilization a reference or desired pose is given as input to the controller and the robot must stabilize in that pose. The results are shown in Figure 7 for a desired pose given by $\begin{bmatrix} x_I^R & y_I^R & \theta_I^R \end{bmatrix}^T = \begin{bmatrix} 2\,m & 3\,m & 4\,rad \end{bmatrix}^T$.

6.2 Trajectory Tracking

In the trajectory control problem, the reference is now dynamic. The reference chosen for this test was a circular trajectory with a $2m$ radius, whose points were generated internally by the kinematic

Other Tasks (*CS)	🟨	🟨	🟨	🟩	🟨	🟩	🟨	🟩	🟨	🟩	🟨	🟨	🟨	🟩	🟨	🟨	🟩
Wheels Controller	🟨	🟨	🟩	🟥	🟩	🟥	🟩	🟥	🟩	🟥	🟨	🟨	🟩	🟥	🟨	🟩	🟥
Kinematic Controller	🟩	🟩	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟩	🟩	🟥	🟥	🟥	🟥	🟥
Distance Sensor	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟩	🟥	🟥
Period	<10ms		<5ms	>5ms	<5ms	>5ms	<5ms	>5ms	<5ms	>5ms	<10ms		<5ms	>5ms	3ms	<5ms	>5ms

*CS = Context Switching 🟩 Running 🟨 Ready 🟥 Blocked

Figure 6: Illustration of the task scheduling.

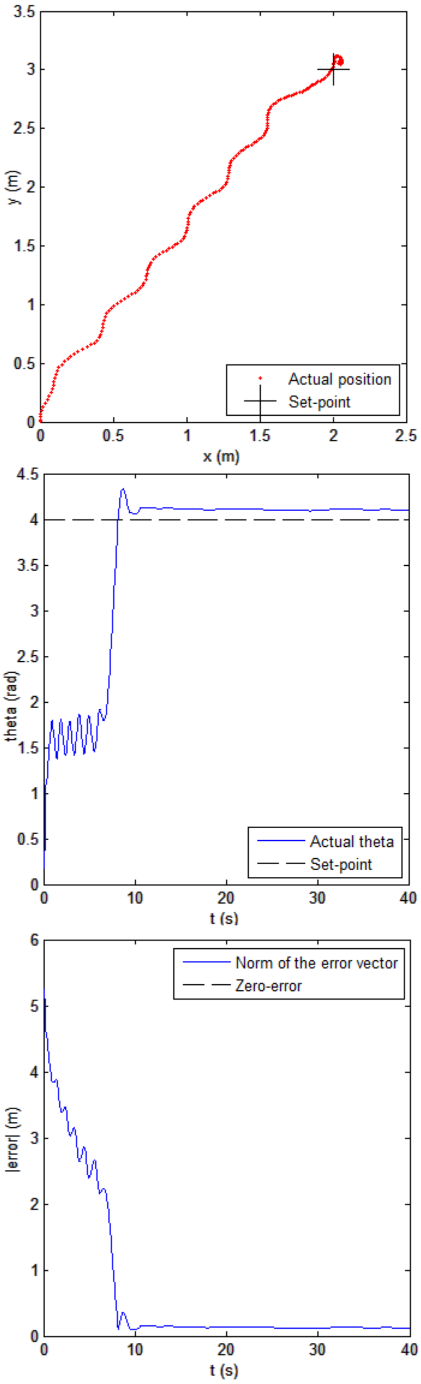

Figure 7: Results for point stabilization.

controller task through the parametric equation of the circle, the orientation was fixed in $\pi/2\,rad$ and the angular velocity of the reference trajectory is $\omega^R = 0.2\frac{rad}{s}$. The parametric equations to generator the desired trajectory are given in Equation 8. The results are shown in Figure 8.

$$\begin{bmatrix} x_I^R(t) \\ y_I^R(t) \\ \theta_I^R(t) \end{bmatrix} = \begin{bmatrix} 2\,\cos\left(\theta_I^R(t) + \omega^R\,t\right) \\ 2\,\cos\left(\theta_I^R(t) + \omega^R\,t\right) \\ \pi/2 \end{bmatrix} \tag{8}$$

7 Conclusion

In this paper a RTOS was applied to embed a kinematic controller in omnidirectional mobile robot AxeBot, equipped with various different sensors. The sensors required processor time to be read, and then the implemented software made use of the prioritizing and semaphores mechanisms from the RTOS to allow for a synchronized processor time use, without compromise any of the tasks that the robot should perform.

The tests results of the kinematic controller were satisfactory, in the sense that the errors were small, and the robot was still able to perform sensing activities.

Acknowledgements

Authors acknowledge the support of CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior).

References

Barry, R. (2010). *Using the FreeRTOS(tm) Real Time Kernel*, Real Time Engineers Ltd.

Bitencourt, A. C. P., da C. e S. Franco, A., de Souza, M. E., de O. Fontes, C. H. and da Costa, A. C. P. L. (2008). Internal model control for trajectory tracking of an omnidirectional robot, **3**: 363–372.

da Costa, A. C. P. L. and Bittencourt, G. (1999). *From a concurrent architecture to a concurrent autonomous agents architecture*, Springer.

Fosler, R. M. (2012). An77759 - getting started with psoc® 5lp.

Ribeiro, T. T. (2010). Sistema de controle em tempo real aplicado à robótica móvel. Trabalho final de graduação.

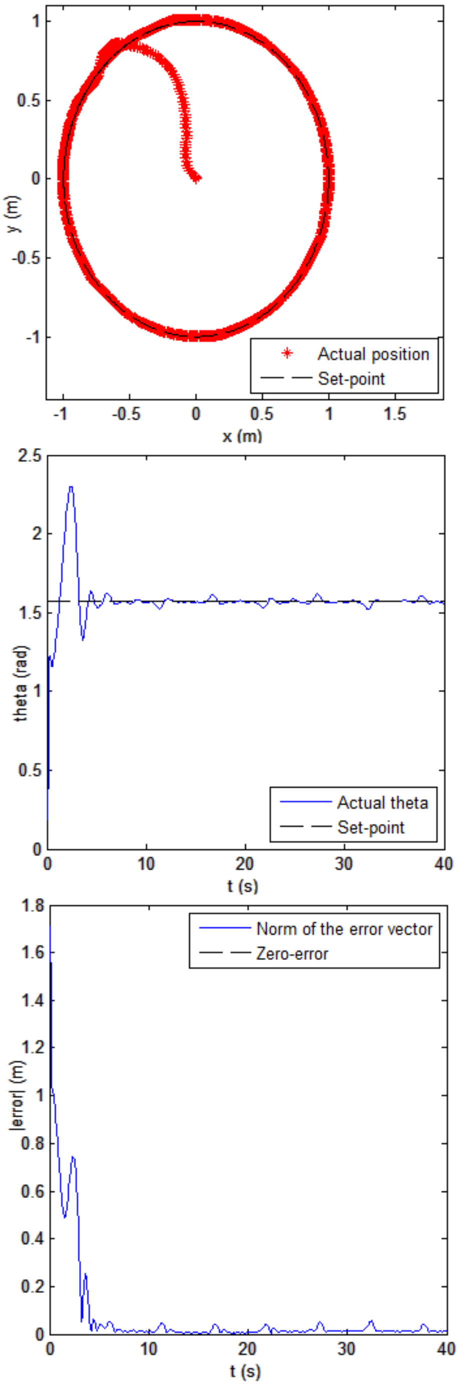

Figure 8: Results for trajectory tracking.

Ribeiro, T. T., dos Santos, J. T., Conceição, A. G. S. and da Costa, A. L. (2011). Sistema microprocessado para controle em tempo real de robôs móveis ominidrecionais, *X SBAI Simpósio Brasileiro de Automação Inteligente*.

Santos, J. T. (2010). Projeto e desenvolvimento de um sistema microprocessado aplicado à robótica móvel. Trabalho final de graduação.

Stankovic, J. A. and Rajkumar, A. (2004). *Real-Time Operating Systems*, Vol. 28, Kluwer Academic Publishers.

Tsai, C.-C., Jiang, L.-B., Wang, T.-Y. and Wang, T.-S. (2005). Kinematics control of an omnidirectional mobile robot, *Proceedings of 2005 CACS Automatic Control Conference*.