A New Approach in Mobile Robot Fault Tolerant Control. Minimizing Costs and Extending Functionality.

CRISTIAN AXENIE
Dunarea de Jos University
Department of Automation and Industrial Informatics
Stiintei Street, Galati
Romania
cristian.axenie@gmail.com,http://robotics.viviti.com

Abstract: Today's trends in control engineering and robotics are blending gradually into a slightly challenging area, the development of fault tolerant real-time applications. Hence, applications should timely deliver synchronized data-sets, minimize latency in their response and meet their performance specifications in the presence of disturbances and faults. The fault tolerant behavior in mobile robots refers to the possibility to autonomously detect and identify faults as well as the capability to continue operating after a fault occurred. This paper introduces a real-time distributed control application with fault tolerance capabilities for differential wheeled mobile robots. ARTEMIC combines the power and flexibility of real-time open-source software, the robustness of nonlinear control schemes, the efficiency of Kalman filtering and the extensibility of high-level programming languages for hardware interfacing. Specific design, development and implementation details will be provided in this paper.

Key–Words: Mobile Robotics, Fault Tolerant Control, Sliding Mode Control, Extended Kalman Filter, Real Time Linux, Distributed Control, Grafcet

1 Introduction

By accepting the challenge to ensure fault tolerant real-time control of a self designed and built wheeled differential mobile robot the author has been able to develop a hierarchical software application that minimizes costs and supports extensibility. Analytical software redundancy is implemented rather than hardware redundancy for fault detection and identification for the minimal robot structure. Following a brief application architecture overview is given so that one can get a proper image on the implementation specific. The main application is designed on a distributed, client-server pattern, based on a TCP/IP wireless communication. Next, the main layers of the server application are listed maintaining a direct connection to the mobile robots features. The applications base level is built over the interface with the actuators (H-bridge MOSFET driver and 2 DC motors) and sensors (encoders and bumpers) found in the robots structure. The next level is the control and fault tolerance level, to the extent that the control algorithm is implemented here and loops concurrently with the monitoring and fault tolerance task to ensure the robot's autonomy and performance in trajectory tracking operation. The Grafcet was chosen to be the tool to support various control algorithm implementations and ensures proper serial / parallel execution of the control and monitoring tasks. The fault tolerant module is based on an set of Extended Kalman Filters used to estimate the current robot position and is using a residual computation and a statistical analysis to determine if a fault occurred in the system and to discriminate between the faults. The fault tolerant module is comprised of a fault detection sub-module, a fault identification sub-module and also a control reconfiguration submodule. The third level is a responsible with the communication task, to the extent that it implements a data server and a control server to ensure reliable data and command flows over a wireless network to the other nodes in the distributed system. The client application was designed to meet some specific requirements. The first type of client is responsible of interacting with the robot operation, basic start, stop, pause actions of the robot, but also with monitoring the status of the robot by receiving specific packets with sensor data. Due to the need of interactivity when injecting the faults a software fault injection framework was developed at this level. It's purpose is to inject a certain fault at a certain moment in time chosen by the client user. The second type of client is limited only to access data logs

from the robot, currently not used. ARTEMIC or Autonomous Real Time Embedded Mobile robot Intelligent Controller was developed as the diploma thesis project by the author.

2 Server-side application description

Next an extended description of the server-side application running on the robot embedded computer is given. All architectural and functional aspects will be presented, with focus on both software development elements and implemented control engineering concepts. The server-side application is a real-time Linux C/C++ application. The real-time capabilities were obtained at the OS level by modifying the standard Linux kernel with the RTAI (Real Time Application Interface) patch based on ADEOS (Adaptive Domain Environment for Operating Systems)to the extent that the enhanced micro-kernel can ensure a proper preemption level and I/O latency minimization. The hardware interface with the sensors and actuators is ensured by a data acquisition card (NI-PCI6024E) managed by a powerful kernel-space real-time driver and a rich API in user-space, both part of the COMEDI (Linux COntrol and MEasurement Device Interface). To properly manage the execution of the two main tasks, namely the control task and the monitoring task the application is organised on an abstract Grafcet (Graphe Fonctionnel de Control des Etapes et Transitions) which supports the higher level parallelism in the execution of the two tasks. The control task is comprised of two inner speed control PID loops (running at 50ms) for the 2 DC motors and a main robot position control loop based on a sliding mode controller(running at 200ms). Concurrently, the monitoring and diagnosis task receives sensor data which is then processed by a set of 5 Extended Kalman Filters every 50 ms. By examining the real and predicted data the application can decide if a fault occurred and what kind of fault based only on sensor information using a simple thresholding mechanism and a statistical test on the residuals. Next an architectural overview of the whole application is depicted in figure 1.

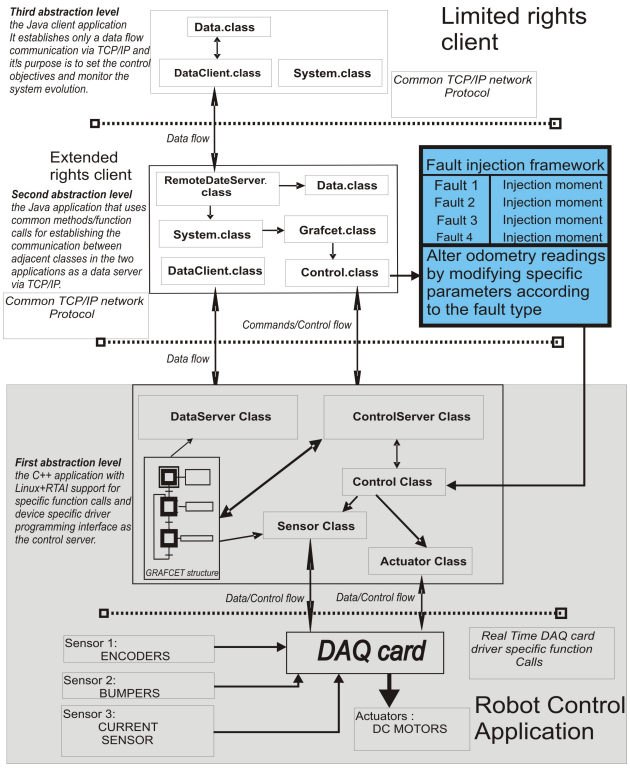


Figure 1: Main application architecture.

2.1 Base level description

The base level is the main support for the control and monitoring tasks to the extent that the sensors and actuators are interfacing with the software by the real-time data I/O features given by the Linux+RTAI and COMEDI tools. The developed application is based on a typical object-oriented pattern abstracting all the hardware through generic and flexible classes that are encapsuling specific functionality. As mentioned earlier the main application was developed on top of an enhanced Linux OS to extend its real-time features by using a dual-kernel approach introduced by RTAI.[5] Currently the root OS is a Red Hat 9 Linux distro using a 2.4.24 kernel and RTAI 3.0. This dual kernel approach implies that the real-time micro-kernel is inserted between the hardware and the root Linux OS, it has it's own task scheduler and it doesn't depend on the Linux critical sections. The micro-kernel has priority to catch I/O interrupts for it's real-time routines before Linux (that has a secondary priority level and is receiving only virtual interrupts). To minimize latency the micro-kernel ensures low task-switching periods and some specific inter process communication mechanisms with Linux. Due to it's main advantages described earlier RTAI was chosen to support the development of the application. On top of the new

OS a real-time I/O driver was added. The COMEDI DAQ card driver fits well on top of RTAI and enhances data transfers and I/O with low latencies. COMEDI is comprised of a kernel-space driver and an user space API named Comedilib built on a hierarchical pattern to the extent that each DAQ card is composed of devices, subdevices and channels. Next the presentation shall focus on the base level of the server-side application running on the embedded robot computer. The specific I/O functions are presented in the application context. The server-side application is described in figure2.

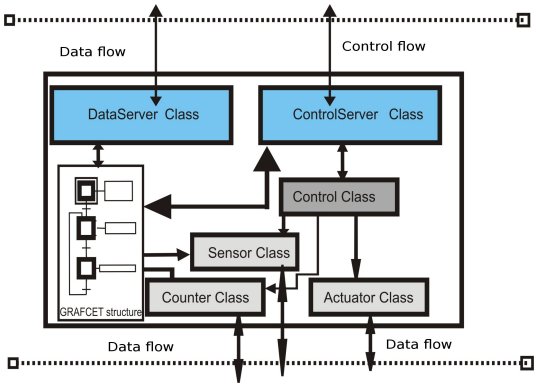


Figure 2: Server-side application architecture.

To properly understand how the robot control application works, a typical operation context is described, emphasizing class communication during execution. Each sampling period the application reads the data through the DAQ card from the encoders and computes, according to the read values, a control signal for the 2 DC motors. The specific functions to obtain a value from a sensor or to set a value to an actuator are comprised in the Counter, Sensor and Actuator classes. Each of these classes calls specific user-space COMEDI functions and accesses specific sensors through derived classes like the AnalogSensor that monitors the current variation in the motors when the load changes. For the bumpers a DigitalSensor class object was created and a Counter object is used to access the values (number of pulses) received on the DAQ card's 12bit counters from the two encoders, used in the odometry computation. Next the outer sliding mode loop computes the control signal for the robot to achieve the next point in the trajectory in a timely manner and then the computed control signal is transformed in speed set-points for the inner PID loops. The PID loops are computing a speed control signal for the motors which is then transformed using a look-up table in the corresponding input voltage for the H-bridge driver which outputs PWM signals to each of the two motors. The monitoring and diagnosis task that runs concurrently is fed with sensor data, in fact the odometry computations and each sampling period the 5 EKF are testing if a fault occurred and if occurred, what kind of fault is it. The fault tolerant control module is then issuing specific warning messages to the client application regarding the current status of the robot operation. Next an in depth description of the two tasks running on the embedded robot controller is given.

2.2 Control algorithm level description

As mentioned earlier the robot is operating in trajectory tracking mode so its main objective is to follow a certain trajectory under time constraints. The control algorithm goal is to minimize the position errors, namely the longitudinal error, the lateral error and the heading error. To ensure proper error convergence in the presence of disturbances and modelling uncertainties a sliding mode controller was chosen. A sliding mode controller based on the kinematic model of the robot was used. By using a sliding mode controller a trade-off between tracking performance and parametric uncertainty was made, to the extent that it provides a good solution in maintaining the stability and consistent performance in the face of modelling imprecision and disturbances.[6,9] Next the complete robot control system is presented in figure3.

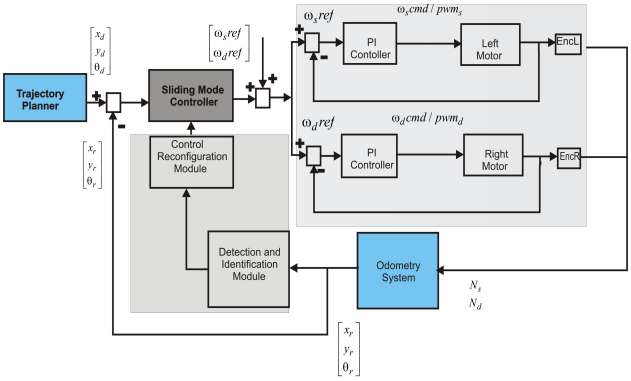


Figure 3: Control system architecture.

The robot should track the reference parametrized curve with time constraints and so real-time control signal computation is critical. The main idea behind the trajectory tracking operation mode considers a virtual robot that tracks the reference trajectory with no error and "pulls" the real robot after it to the extent that the controller reacts to minimize the errors.[3] The generic trajectory tracking problem is depicted in figure4.

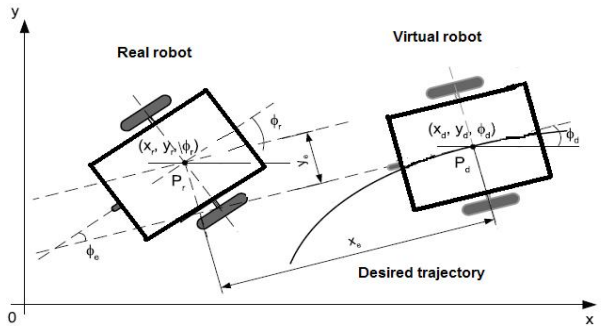


Figure 4: Robot operation in trajectory tracking.

Let the robot be defined by the next position vector (state vector) $[x_r, y_r, \theta_r]^T$ and we define the error vector as $[x_e, y_e, \theta_e]^T = [x_d, y_d, \theta_d]^T - [x_r, y_r, \theta_r]^T$, where $[x_d, y_d, \theta_d]^T$ is the virtual robot state vector, the set-point vector to track.[2] Next the robot's kinematic model and tracking error computation is considered.

$$\begin{cases} \dot{x_r} = v_r \cos\theta \\ \dot{y_r} = v_r \sin\theta, \\ \dot{\theta_r} = \omega_r \end{cases} \quad \begin{cases} \dot{x_d} = v_d \cos\theta_d \\ \dot{y_d} = v_d \sin\theta_d, \\ \dot{\theta_d} = \omega_d \end{cases} \quad \begin{bmatrix} x_e \\ y_e \\ \theta_e \end{bmatrix} = \begin{bmatrix} \cos\theta_d & \sin\theta_d & 0 \\ -\sin\theta_d & \cos\theta_d & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_r - x_d \\ y_r - y_d \\ \theta_r - \theta_d \end{bmatrix}$$

The main objective was to design a stable and robust controller for robot position control that should output a control signal vector $[v_c, \omega_c]$ (linear and angular speeds) in the presence of disturbances. The error derivative is given by

$$\dot{x_e} = -v_d + v_r cos(\theta_e) + y_e \omega_d$$
$$\dot{y_e} = v_r sin(\theta_e) - x_e \omega_d$$
$$\dot{\theta_e} = \omega_r - \omega_d$$

where the set-point values for the speed control loops are v_d and ω_d.[7] The set-point for the main robot position control loop is generated by a trajectory planner based on 5th degree equations for smoothing the robot's operation and control effort. The trajectory planner outputs linear and angular speed and acceleration set-points at each sampling period. These values are transformed using inverse kinematics equations in vectorial form $[x_d, y_d, \theta_d]^T$. The next step was to design the switching surfaces for the sliding mode controller such that it can ensure proper error convergence. The first switching surface was chosen to ensure lateral error convergence and is given by the next equation $s_1 = \dot{x_e} + k_1 x_e$. To ensure proper longitudinal and heading error convergence the second switching surface combines the two objectives and is given by $s_2 = \dot{y_e} + k_2 y_e + k_0 sgn(y_e)\theta_e$, where the k_1, k_2, k_3 parameters are specific weights associated to each error component.[11] Practically the control signal could be computed using $\dot{s} = -Qs - Psgn(s)$ where Q,P are positive constant values that are determining the error convergence speed. By deriving the surfaces equations and replacing the error components one can easily obtain the control signal vector under the form of

$$\dot{v_c} = \frac{-Q_1 s_1 - P_1 sgn(s_1) - k_1 \dot{(x_e)} - \dot{\omega_d} y_e - \omega_d \dot{y_e} + \dot{v_r}\theta_e sin(\theta_e) + \dot{v_d}}{cos(\theta_e)}$$

$$\omega_c = \frac{-Q_2 s_2 - P_2 sgn(s_2) - k_2 \dot{(y_e)} - \dot{v_r} sin(\theta_e) + \dot{\omega_d} x_e + \omega_d \dot{x_e}}{v_r cos(\theta_e) + k_0 sgn(y_e)} + \omega_d$$

To ensure a good convergence speed of the robot state vector to the switching surface (which in fact gives the desired system dynamics) and also to eliminate chattering (caused by fast control signal switches) the P and Q terms should be cautiously chosen.[10] To ensure the stability of the system we defined a Lyapunov candidate function given by $V = \frac{1}{2} s^T s$ and with $\dot{V} = s_1 \dot{s_2} + s_2 \dot{s_1} = -\dot{s^T} Qs - P_1 |s_1| - P_2 |s_2|$ the main condition is that $V\dot{V} < 0$ achieved if $Q_i >= 0$ and $P_i >= 0$. As mentioned earlier the sliding mode controller outputs angular speed set-points for the inner PID loops. The PID controllers for the two DC motors were implemented using first-order-hold discretization method. The PID loops are running at 20Hz and the specific parameters were tuned to meet a fast response time (0.5s) and a small overshoot (10%). Next some results from the closed loop fault-free operation of the robot are given to analyze the robot behavior on a specific trajectory.

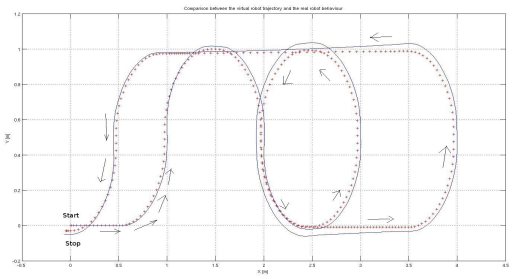


Figure 5: Robot operation in trajectory tracking with sliding mode controller in fault free operation.

The real robot trajectory is represented by a continuous line and the reference trajectory is given with

crosses. The movement direction is indicated by arrows. As one can see the position error is kept in acceptable limits by the robust sliding mode controller and the robot can achieve it's objective in time. Once more one can observe that good performance is achieved in using the sliding mode controller. To analyze the sliding mode controller operation next the linear and angular velocities variations of the robot are presented. Each graph contains the set-point evolution and the real robot velocity.

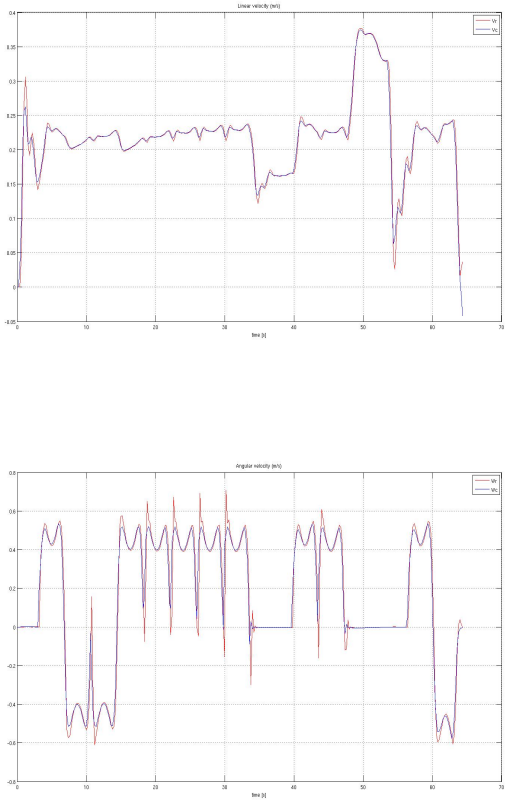


Figure 6: Linear and angular velocities variation during robot operation.

Previously depicted, the set-points given by the trajectory planner are marked in red and the real robot velocities are marked in blue. Once more one can observe that good performance is achieved in using the sliding mode controller. Another important aspect regards the analysis of the error components in the fault free operation, because it can be considered as a proper validation for the controller. Next the three error components are depicted.

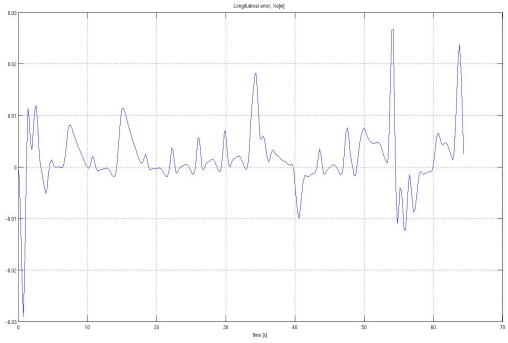


Figure 7: Robot longitudinal error (x_e).

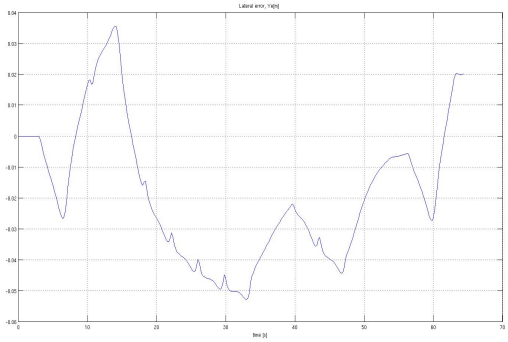


Figure 8: Robot lateral error y_e.

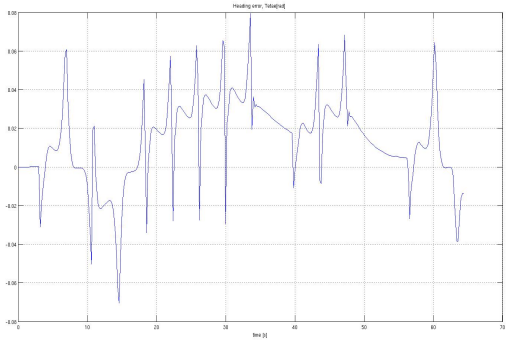


Figure 9: Robot heading error θ_e.

One can see that the error convergence is properly ensured by the controller. The longitudinal error in the fault free operation is under 3cm, the lateral error is under 6cm and the heading error in under 8 rads.

Experiments have shown that even for simpler trajectories (simple 2m line, 0.75m radius circle) the controller behaves well and despite the fact that the robot structure had some hardware limitations (small encoder resolution, 500PPR) the real-time control goals were met even for slightly high sampling periods;all the results are validated with the earlier error analysis. Following the next level of the application is presented, namely the fault detection and identification module and the support for fault tolerant control.

2.3 Monitoring and diagnosis level description

Mobile robot fault tolerant control refers to the possibility to autonomously detect by software if a fault occurred in the robot's structure or operation environment so that the robot cannot satisfy its objectives.[1] Due to hardware limitations in injecting faults the application is also responsible to inject faults by software using a fault injection framework. By benefiting of a certain degree of robustness given by the sliding mode controller the application resides on a simple mechanism for fault detection, identification and support for control reconfiguration based on the Extended Kalman Filter (EKF). So, the application implements a set of 5 EKF that form the fault detection and identification module. It is known that the Kalman filter is basically a set of equations implementing an optimal estimator, by minimizing the estimate error covariance when certain conditions are satisfied. It can be used for both parametric or state estimation. The extended Kalman filter, EKF, is used mostly when the process model or the measurement-process relations are nonlinear, so it fits well in this application. This approach using EKF is a multi-model based method for fault detection.[4] The current application considers a fault benchmark comprised of faults manifested by system's parameter variation specifically mechanical faults that alter the robot behavior. Hence two fault classes were defined each containing two types of faults. The first class of faults introduces the flat tire fault for each of the robot wheels. At the implementation level this fault exhibits by diminishing the wheel radius with a certain value. As a consequence the whole kinematic model of the robot is modified and visible modifications in the robot's behavior will be present. The second class of faults introduces a periodic variation of the wheel radius (a bump, caused hypothetically by an object attached to the wheel). Hence, for a small time period the variation will manifest itself and alter the robot operation. As mentioned earlier a fault injection framework was implemented and its main purpose is to alter the values from the sensors in odometry specific computations. Hence, when no fault is injected the robot will behave normally (as presented when the control level was described) but when a fault was injected the robot position determining system (in fact the control system feedback, odometry) will feed altered data to the controller as if a physical fault occurred. Next a depiction of the fault detection and identification module is given.

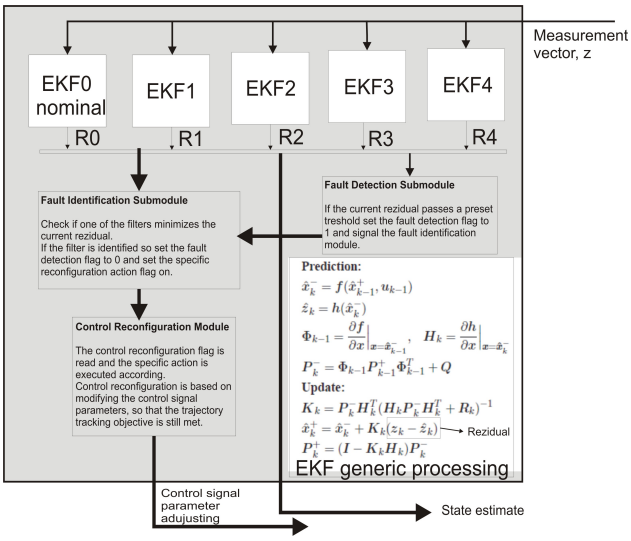


Figure 10: Fault detection and identification module.

Each of the five EKF encapsules a kinematic model of the robot but with different parameters. The main idea resides on the fact that for the same input vector, z, with noise, each filter generates a prediction of the robot's state vector. Each filter is associated with a certain fault type. EKF0 is the nominal filter corresponding to the fault free robot operation. EKF1 filter contains the same robot kinematic model but with modified parameters to emphasize the right tire flat fault, so the right wheel radius has a smaller value. Hence, the prediction of EKF1 will be the robot state vector if a right tire flat occurred. The EKF2 filter is similar to EKF1 excepting it encapsules the left tire flat fault specific model. In the same way the other two filters associated with the second fault class, EKF3 and EKF4, are predicting the robot's state vector in the presence of a second fault class fault. Besides the state estimate each EKF generates a measurement vector estimate during the prediction stage, which is used in the correction stage of the filter. Basically the detection method is based on a nominal residual computation and a simple thresholding test on the nominal residual which can output that a fault occurred. It is the only one affected by the fault occurrence so it will step out from some preset value intervals. When identifying the fault the application just

finds the smallest residual from the 5 ones, because when a fault occurs the main measurement vector is altered and it nears the predicted measurement vector by the EKF encapsuling that fault behavior. Next, specific implementation details of the EKFs are presented. Starting from the generic structure during the implementation some simplifying hypothesis were issued and we know that Q is the process noise covariance matrix, R is the measurement covariance matrix, $\sigma_x = \sigma_y = \sigma_\theta = 0.1$ are the standard deviations for the process noise, $\sigma_{meas} = 0.01$ is the standard deviation of the measurement noise.[8] To correctly detect if a fault occurred some statistical tests on the residual are required. The statistical parameters of the residual (mean and variance) will be compared on-line at each sampling period with the values obtained for these parameters in fault-free operation. In fault-free operation the residual is a a white noise sequence with 0 mean and $\eta_k = H_k(\hat{x_k^-})P_k^- H_k^T(\hat{x_k^-}) + R$ variance. Basically if a fault occurs in the system it will determine a modification of the next residual based standard sequence,

$\eta s_k = (\eta_k = H_k(\hat{x_k^-})P_k^- H_k^T(\hat{x_k^-}) + R)^{-1/2}(y_k - h_k(\hat{x_k^-})) = \eta_k^{-1/2} r_k$ The goal is to determine the estimate of the real value of a sample given by $\hat{\bar{\eta}_s} = \frac{1}{N}\Sigma\eta_{sj}$, where N is the number of samples. So, in the hypothesis H_0 of no fault $\hat{\bar{\eta}_s}$ has a gaussian distribution with 0 mean and 1/N covariance. Over a certain acceptance threshold the H_0 hypothesis is no longer true and so a new hypothesis H_1 is now true marking a fault occurrence. Next some experimental results are presented. First it is useful to analyze the residuals generated by the nominal filter when a fault occurs, in fact these residuals are those predicted by the fault embedding filters, from the moment the fault occurs. For example if a right tire flat fault occurs at sample 100 in the execution the residual evolution from 100th sample is the predicted residual by the EKF1.

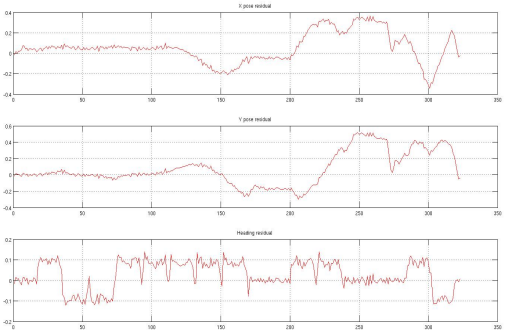


Figure 11: Residual analysis for first fault class.

More, if a left wheel bump fault occurs at sample 100 in the execution the residual evolution from 100th sample is the predicted residual by the EKF4.

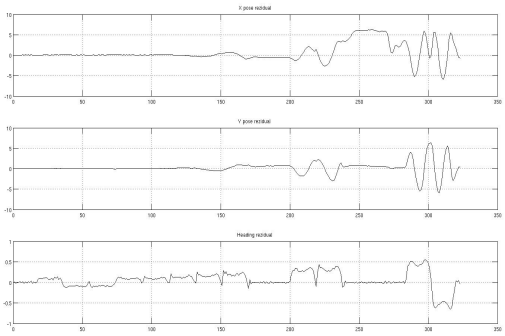


Figure 12: Residual analysis for a second fault class fault.

From this separate residual analysis one can see that each EKF is sensitive to a certain fault type and using these properties the detection and identification modules were built. Assuming that a right tire flat fault was injected at moment t=10s and the new radius of the wheel is 3cm smaller than the initial value the behavior of the robot is altered as one can see in the next figure.

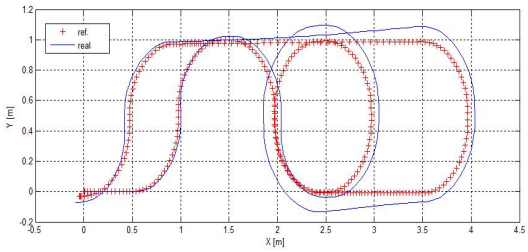


Figure 13: Robot faulty operation vs. reference trajectory.

Next an extended position error analysis over the fault free and faulty operation is given (red marks fault free operation and blue the faulty operation).

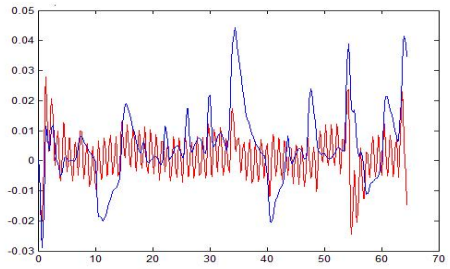


Figure 14: Longitudinal error comparison fault-free/faulty operation.

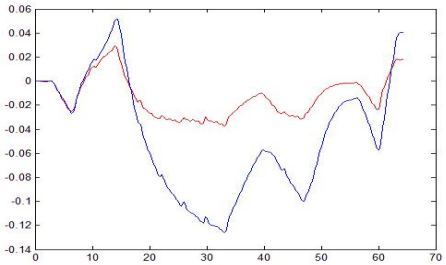


Figure 15: Lateral error comparison fault-free/faulty operation.

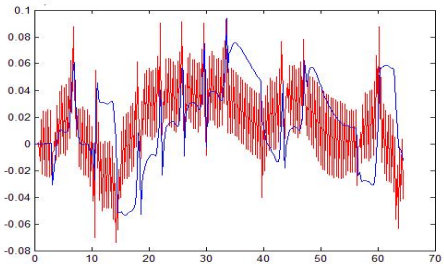
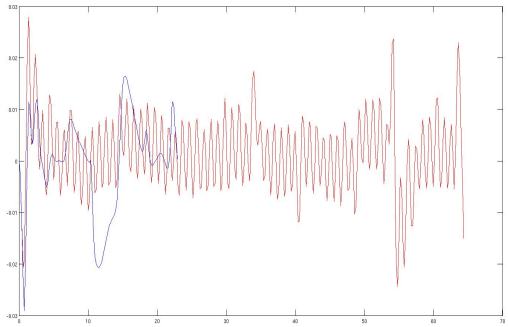
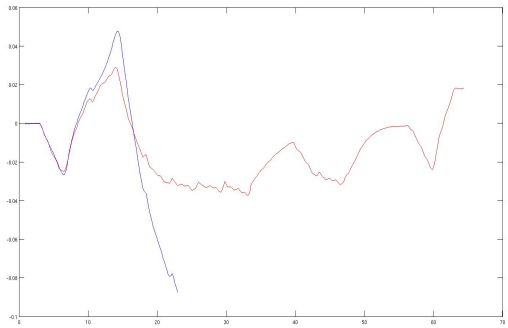


Figure 16: Heading error comparison fault-free/faulty operation.

As one can see there are major variations from the nominal behavior after a fault occurred in the system. To validate the detection mechanism the latency, measured as the time between the fault occurred (in fact injected by the injection framework) and the moment when it has been detected, is very important. Next the latency is marked by pausing the robot operation when a fault was detected.





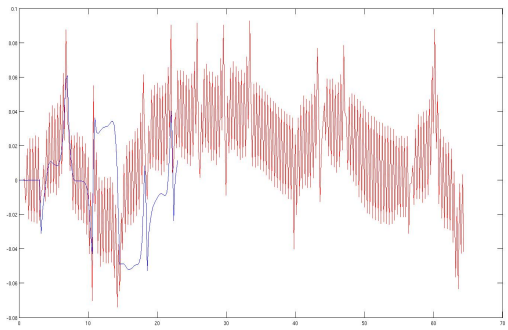


Figure 17: Detection analysis.

It is important to mention that the thresholding mechanism reacts if at least two components of the residual vector are bigger than the preset values. So, the fault could be visible in all the residual components or not. As a typical operation context when a fault is detected, it is then identified and the client application is informed what kind of fault occurred. At the moment only one fault can be detected and

identified, no fault queue or priority system are implemented. Following a brief description of the client application is given.

3 The client-side application description

The two main components of the application are linked through a wireless communication which sustains data and control flows between the distributed system nodes, namely the server, an embedded linux C++ application and the client , a Java GUI application. The communication level is based on two server type classes ControlServer and DataServer both based on socket level communication and using specific designed functions like connect(), serve_requests(), serve_client(), send() and receive(). The client application is responsible with the external events that determine the robot operation. It implements a similar structure with the embedded application and supports the fault injection framework to the extent that the user operating the client can choose the moment, type and gain of the fault before starting the robot operation. Next a synthetic image of the client application architecture is given.

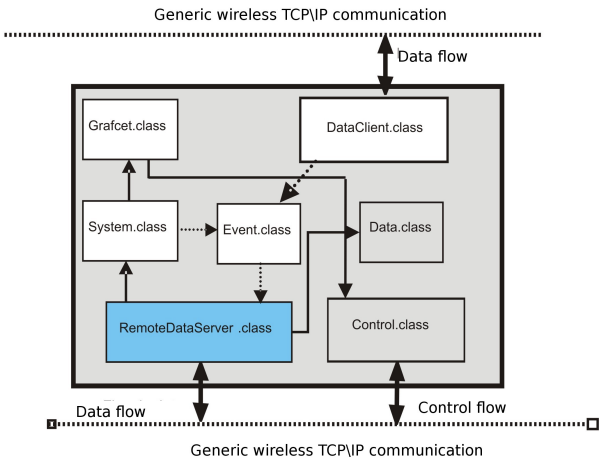


Figure 18: Client-side Java application architecture.

Besides the interactive task of injecting faults the client application is responsible to choose the Grafcet structure that encapsules the execution rules for the control algorithm and and the specific supported actions (start/pause/stop) for the robot operation, also it offers the possibility to choose between multiple trajectories and to visualize the robot's current position.

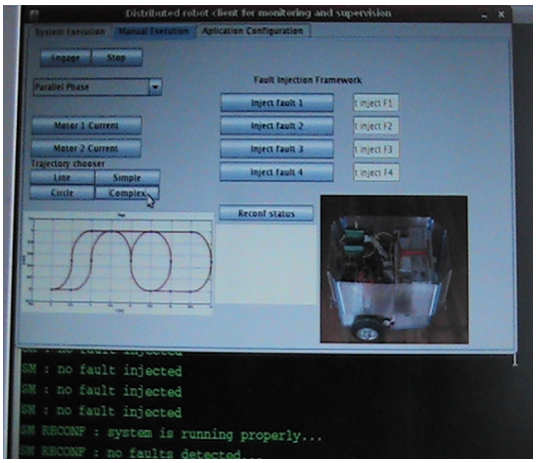


Figure 19: Client-side Java application GUI.

The two possible Grafcet execution schemes serial/parallel assume a serial or parallel execution of user defined tasks for the robot. Currently the two implemented tasks, real-time control and monitoring and diagnosis must run concurrently to ensure consistent and valid results in the fault tolerant control problem. Next are depicted the two possible generic definitions for the internal Grafcet scheme used in the application.

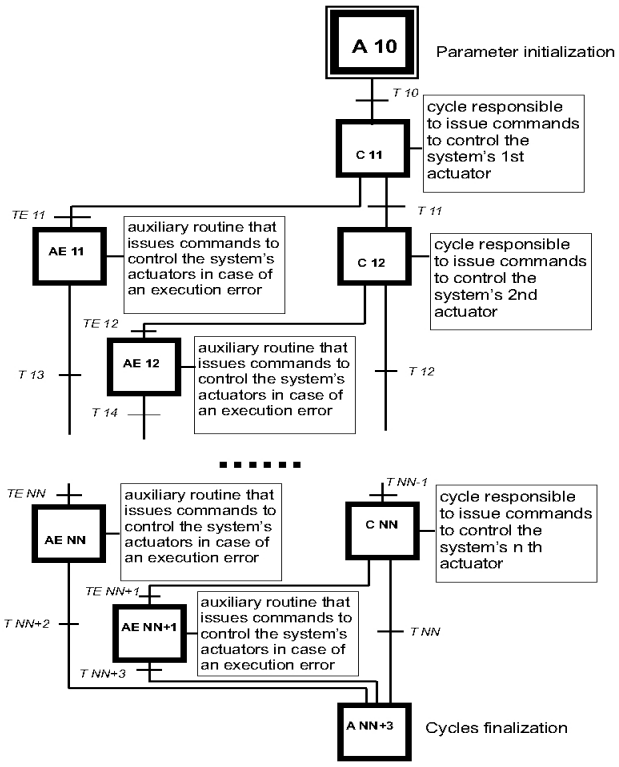


Figure 20: Serial Grafcet scheme.

By using the serial scheme the application will run the two main tasks sequentially with respect for the chosen sampling period.

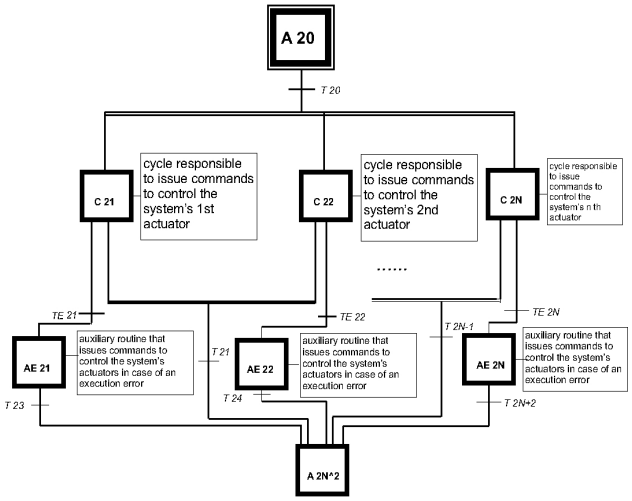


Figure 21: Parallel Grafcet scheme.

The current implemented scheme is the parallel one to ensure proper task concurrently but it sets some constraints in choosing the sampling time due to the fact that the two tasks are complex and complex operations should be executed concurrently. Using a full load of the embedded system latency measurements were taken. So, by using a 50ms sampling period for the inner control loop and the monitoring and diagnosis tasks and a 200ms sampling period for the robot position controller good average ms level latencies were obtained. Some measured latencies for the standard, Low-Latency, preemptible and preemptible with lock-breaking kernels while running different loads in background. All the experiments were performed using the Latency Benchmark tool.[10]

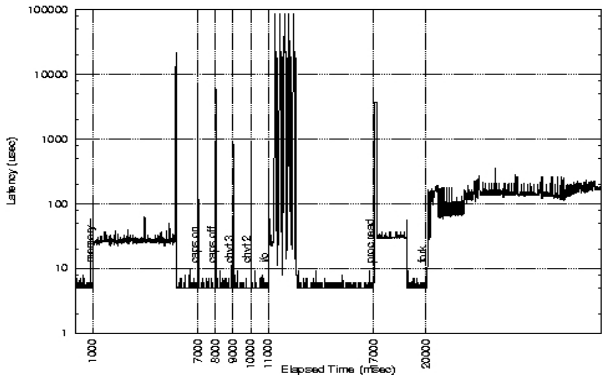


Figure 22: Latency in the Standard Kernel.

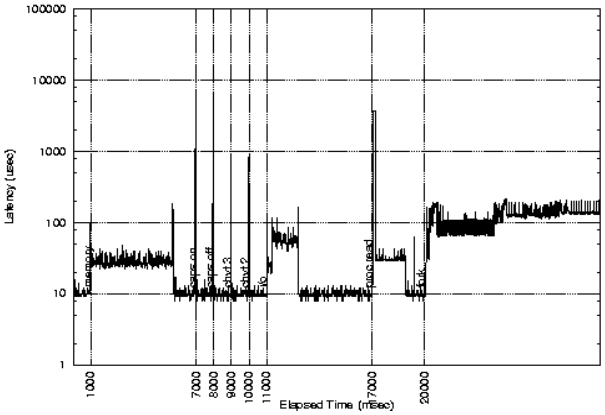


Figure 23: Latency in the Low Latency Kernel.

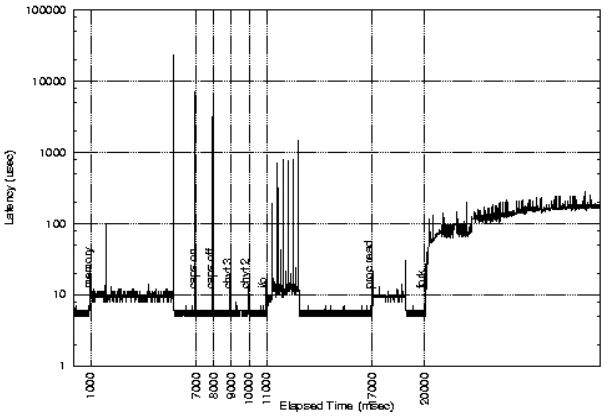


Figure 24: Latency in the preemptible Kernel.

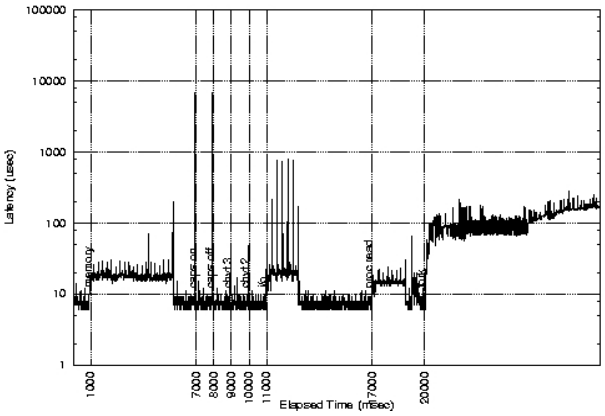


Figure 25: Latency in the Real-Time Kernel.

From the first figure, it is possible to see that standard Linux exhibits high latencies at the end of

the memory stress, during the I/O stress, when accessing the /proc file system, and when switching the caps/lock led. The large latency at the end of the memory stress is due to the munmap() system call. Comparing the figures it is possible to see that the low latency kernel solves all the problems except the /proc file system access and the caps/lock switch. On the other hand, the preemptible kernel eliminates the large latency in the /proc fs access, but does not solve the problem with the memory stress, and is not as effective as the low latency kernel in reducing the latency during the I/O stress. Finally, the real-time kernel seems to provide good real-time performance, with good performance during I/O stress. Following a short synthetic overview over the paper is given emphasizing the main advantages and some future work ideas.

4 Conclusions and future work

ARTEMIC was designed as a flexible application dedicated to mobile robotics fault tolerant control. Its architecture is open and supports extensibility at a ”plug-an-play” level because at the lowest level it supports and offers a simple mechanism to add new hardware to the robot and provides a generic and compatible interface with the hardware. At the upper level it supports extended connectivity. The current fault tolerant control problem was solved by properly connecting specific application architectural elements with control engineering concepts. By using open-source software the costs were minimized and performance was maximized. As future work a port of the developed application from the current Intel Celeron based embedded computer to a MPC8315ERDB PowerPC platform is started. The new platform uses an enhanced RTOS named Xenomai that is a super-set of RTAI and it provides better performance and it is dedicated to embedded architectures. Another direction in the future work regards the possibility to combine multiple fault detection and identification methods to gain the advantages of each composing method and obtain a hybrid and efficient mechanism. This approach shall be combined with an adaptive sliding mode controller to the extent that it can self-adjust it’s parameters based on optimal criteria to ensure faster error convergence. Next figure introduces the ARTEMIC power robot.

Figure 26: The ARTEMIC powered mobile robot.

References:

[1] Patton R., Frank P., Clark R. Fault Diagnosis in Dynamic Systems:Theory and Applications Prentice Hall, New York, 1989.

[2] Bruno Siciliano, Lorenzo Sciavicco, Luigi Villani, Giuseppe Oriolo Robotics, Modelling, Planning and Control Springer, 2009

[3] Razvan Solea, Urbano Nunes Trajectory Planning and Sliding Mode Control for WMR Trajectory Tracking and Path Following Rescpecting Human Comfort Travel CONTROLO Conference Portugal, 2006.

[4] Mayback P.S.,Cox I.J.,Wilfong G.T.(eds) The Kalman Filter:An Introduction to Concepts in Autonomous Robot Vehicles Springer-Verlag, 1990

[5] Burns A., Wellings A. Real-Time Systems and Programming Languages AddWesley, California, 1996.

[6] Utkin V.I. Sliding Modes in Optimization and Control Problems Springer-Verlag, New York, 1992.

[7] I. Susnea, G. Vasiliu, A. Filipescu Real-time, embedded fuzzy control of the Pioneer3-DX robot for path following 12th WSEAS International Conference on SYSTEMS, Heraklion, Greece, July 22-24, 2008

[8] EVGENIA SUZDALEVA,UTIA AV CR Initial Conditions for Kalman Filtering: Prior Knowledge Specification Proceedings of the 7th WSEAS International Conference on Systems Theory and Scientific Computation, Athens, Greece, August 24-26, 2007

[9] YONG XU, WANSHENG TANG, ZHUZHI YUAN Control Design for Uncertain Systems Based on Integral-Type Sliding Surface Proceedings of the 6th WSEAS International Conference on Robotics, Control and Manufacturing Technology, Hangzhou, China, April 16-18, 2006

[10] STERGIOS PAPADIMITRIOU, KONSTANTINOS TERZIDIS Comparative evaluation of the recent Linux and Solaris kernel architectures Proceedings of the 11th WSEAS International Conference on COMPUTERS, Agios Nikolaos, Crete Island, Greece, July 26-28, 2007

[11] GAZENFER RUSTAMOV, MANAFADDIN NAMAZOV, REFIK SAMET Sliding Modes in Finite-Time Control Systems with Variable Structure Proceedings of the 9th WSEAS International Conference on Automatic Control - Modeling and Simulation, Istanbul, Turkey, May 27-29, 2007