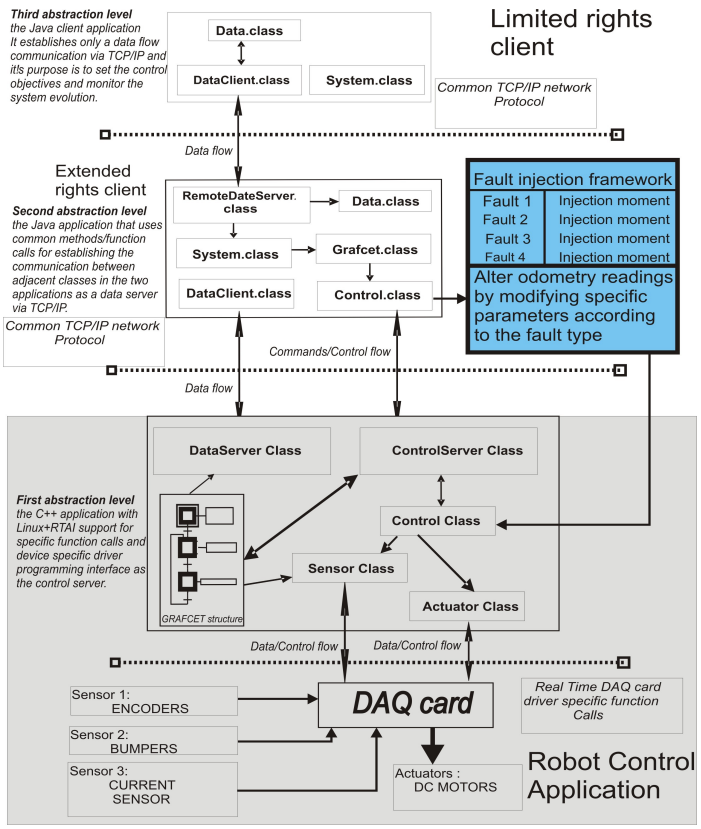
Bachelor thesis report

Mobile Robot Fault Tolerant Control. A fault tolerant real time mobile robot control software application.

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. 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.

Introduction. System and application overview

By accepting the challenge to ensure fault tolerant real-time control of a self made wheeled differential mobile robot I have 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 tolerance. Following a brief application architecture overview is given so that one can get a proper image on the implementation specific. Note that during this presentation I shall focus only on the specific levels of the application directly involved in the fault tolerance aspects and neglecting the description of the other levels. The main application is designed on a distributed, client-server model, based on TCP/IP wireless communication. Next, the main layers of the server application are described with direct connection to the mobile robot's features. The application's lowest level is based on the interface with the actuators and sensors found in the robot's 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. 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 Extended Kalman Filter bank used to estimate the current robot position and is using a residual computation to determine if a fault appeared in the system. The fault tolerant module is comprised of a fault detection sub-module, a fault identification sub-module and a control reconfiguration sub-module (future work). The third level is a responsible with the communication task, to the extent that it implements a data server and a control server to link over a wireless network to the other nodes in the distributed architecture. Although not presented here, 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 and actuator data and also hosts the fault injection framework. The second type of client is a limited only to access data logs from the robot, currently not used. In the next figure one can get a proper overview over the whole distributed application architecture represented at a higher conceptual level.



Next an in depth description of the application is given with focus on the fault tolerant control implementation.

Base level description

As mentioned earlier the base level handles the real time I/O operations issued by the core application to the robot's sensors and actuators. The real time capabilities were implemented by installing the Real Time Application Interface (RTAI), over a standard Linux kernel (currently a RedHat9, 2.4.24 kernel). The RTAI kernel uses an enhanced process scheduler that has an alternate priority system and by using specific algorithms minimizes the latency during I/O operations. The main I/O system is a PCI based DAQ card, namely the NI-6024E that interfaces with the RTOS by using a specific device driver interface, Control and Measurement Data Acquisition Interface (COMEDI), that fits over the real time kernel and manages the accesses to the robot's hardware. By ensuring real time I/O the application minimizes the jitter between the moment the robot controller computes the control signal and the moment the signal is issued by the data acquisition board. Currently the DAQ board outputs a continuous voltage signal to the H-bridge power driver, who transforms it into a PWM signal and send it to the two robot's actuators, two 12V DC motors. The feedback is ensured by a set of sensors. As mentioned earlier the robot was designed with a minimal structure, so the sensor network on the robot is comprised of two incremental encoders with 500 PPR, two bumpers (front, back) and two current sensors for DC motor monitoring. The robot's current processing unit is an Intel Celeron mini-ITX based board. The specific RTAI and Comedi function calls are integrated throughout the application's upper level functions to gain direct access to the base level hardware. Next the control and fault tolerant module is presented.

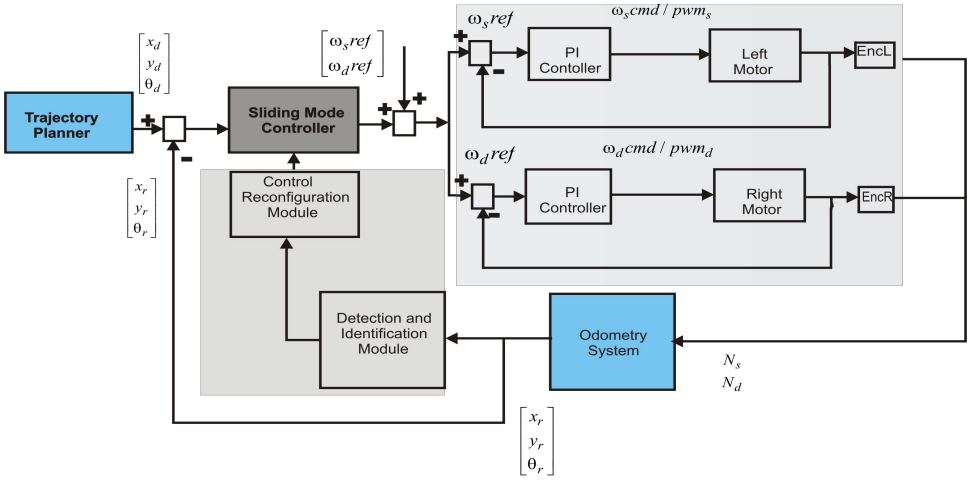
Description of the robot controller and fault tolerant module level

The core of the application is the control and diagnosis level and throughout this short presentation I shall focus only on this aspect. By using the Grafcet, as a specific tool that offers support for parallel or serial execution of both control and fault detection and identification, the application gained flexibility and extensibility. The current implementation uses a cascade control loop, to the extent that there are 2 inner PI (Proportional Integral) control loops for the two DC motors running at 20Hz, and an external loop that uses a Sliding Mode controller, with the kinematic model running at 5Hz in order to control the robot's position in trajectory tracking operation. Next some specific details about the design of the Sliding Mode controller are given to mark its particular features in mobile robot trajectory tracking. I've considered the kinematic model equations for the differential mobile robot by accounting that the geometric centre and the rotation centre are not identical. The error vector for trajectory tracking can be obtained by knowing the virtual robot position given by the trajectory planner. It is supposed that the desired trajectory for the mobile robot is pre-specified by a trajectory planner, comprised of speed and acceleration references for the robot. The problem was to design a robust controller so that the robot tracks the desired trajectory under disturbances. The selected sliding surfaces that are marking in fact the desired dynamics are ensuring longitudinal and lateral and heading error convergence.
The implemented control law is:

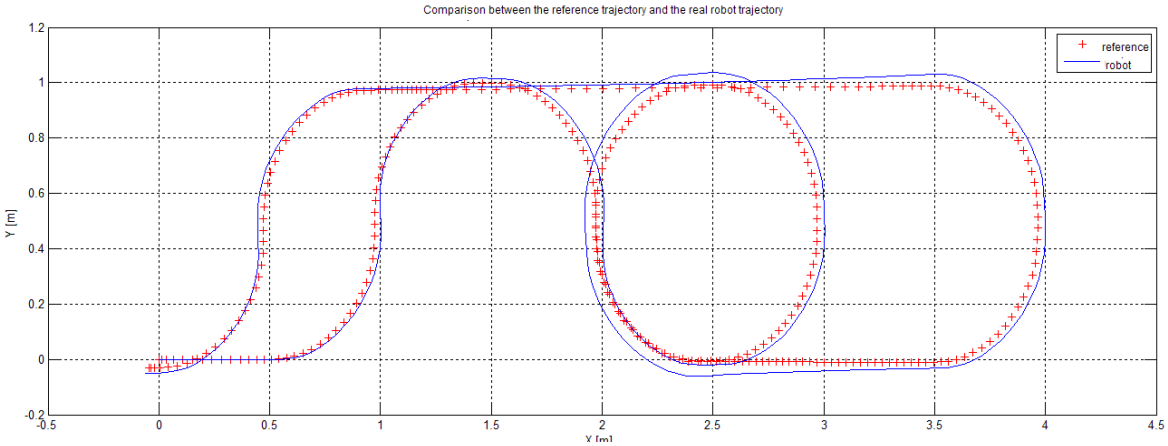
$$\dot{v}_r = \frac{-Q_1 s_1 - P_1 \operatorname{sgn}(s_1) - k_1 \dot{x}_e - \dot{\omega}_d y_e - \omega_d \dot{y}_e + v_r \dot{\theta}_e \sin\theta_e + \dot{v}_d}{\cos\theta_e} \quad \text{and}$$

$$\omega_r = \frac{-Q_2 s_2 - P_2 \operatorname{sgn}(s_2) - k_2 \dot{y}_e - \dot{v}_r \sin\theta_e + \dot{\omega}_d x_e + \omega_d \dot{x}_e}{v_r \cos\theta_e + k_0 \operatorname{sgn}(y_e)} + \omega_d \ .$$

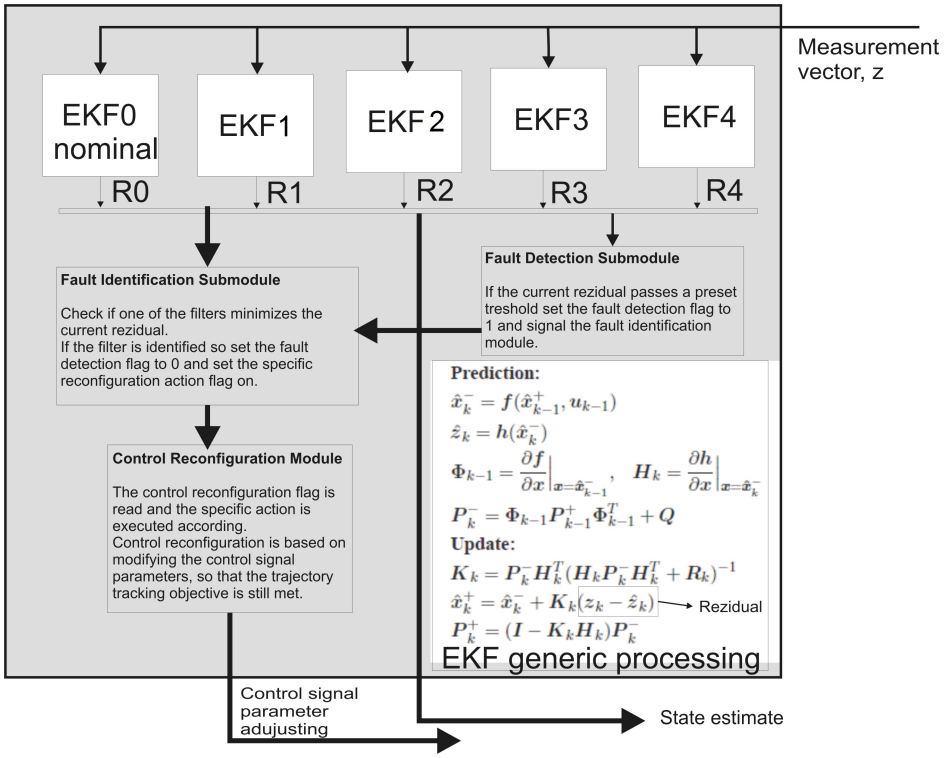
The Q_i and P_i terms were determined such that they ensure good convergence of the state trajectory to the surfaces. The main control loop is next depicted.



The demo trajectory is now depicted and the comparison between the reference and real robot trajectories are emphasized.

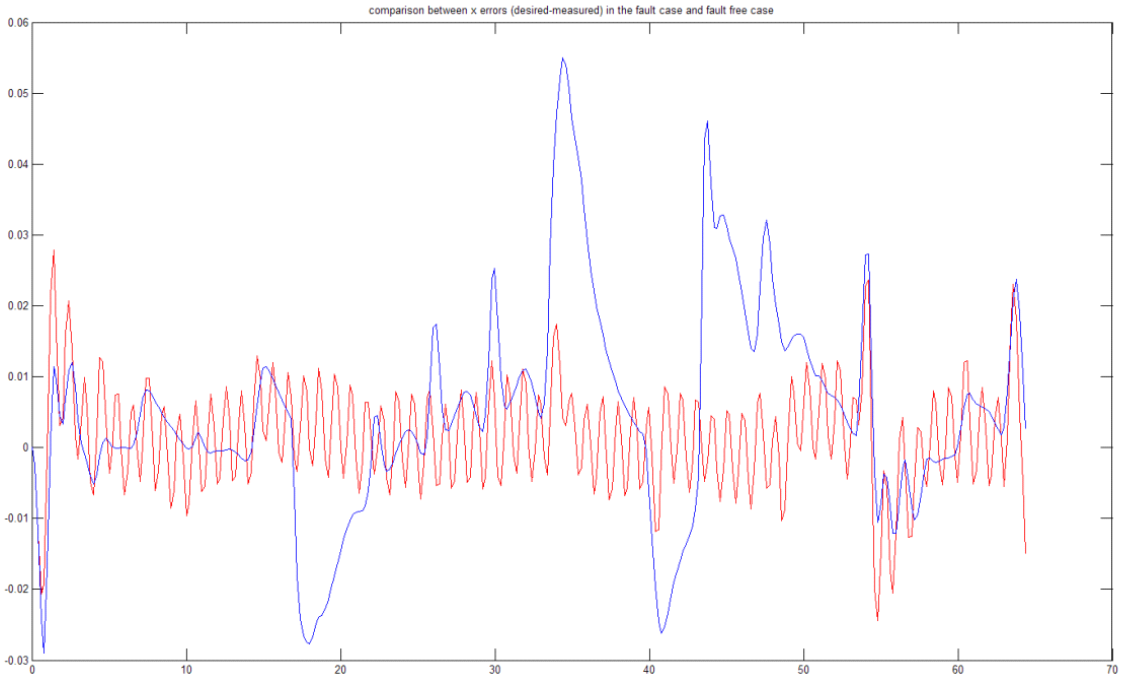


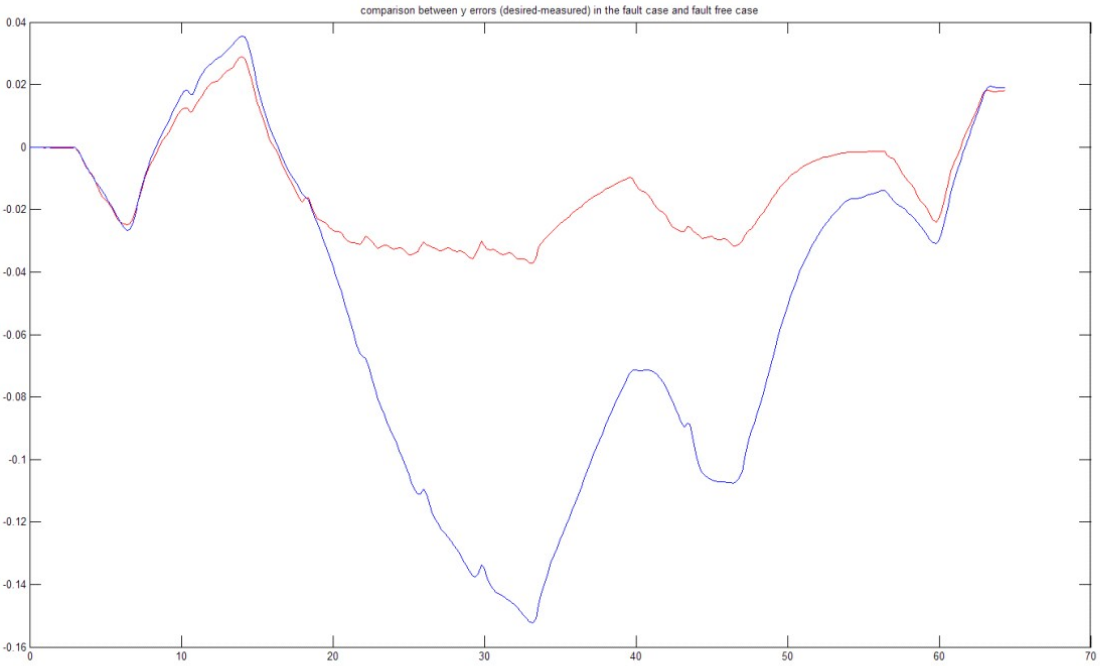
A typical operating context is now described. At each sampling moment (every 50ms) each of the DC motor PI controllers are controlling the motors' speed by tracking the reference set by the Sliding Mode controller (running at 200ms) which is responsible for the accuracy in robot positioning. The odometry system is responsible to output the current robot position by using only the encoder data. Also the odometry system feds in data for the Fault Detection and Identification Module which is using 5 Extended Kalman Filters used to detect and discriminate between several faults and then issue specific signals to the Control Reconfiguration Module. Each of the 5 Kalman filters embeds in his structure a similar kinematic model of the robot, but with different parameters, to the extent that it shall give a state vector estimate, $[x_{est},y_{est},teta_{est}]^T$ in the context of a specific fault and by comparing it to the measured state vector $[x_m,y_m,teta_m]^T$, it can decide if a specific fault occurred. The implemented Kalman filters work for a considered fault from the developed benchmark, like the robot wheel radius variation fault and the wheel periodic bump fault. As one can see each EKF computes it's own residual by considering the presence of the fault. At each sampling time a current residual is computed by subtracting from the current measurement vector, *z*, each estimated measurement vector $\hat{z}$ for each filter, and the one minimizing the residual determines the identification of a specific fault. In fact the detection is based on the analysis of the statistical properties of the residual, to the extent that each filter computes a standard sequence of the residual and by using a thresholding method it can effectively decide if a fault occurred. Next the Fault Tolerance subsystem is depicted.

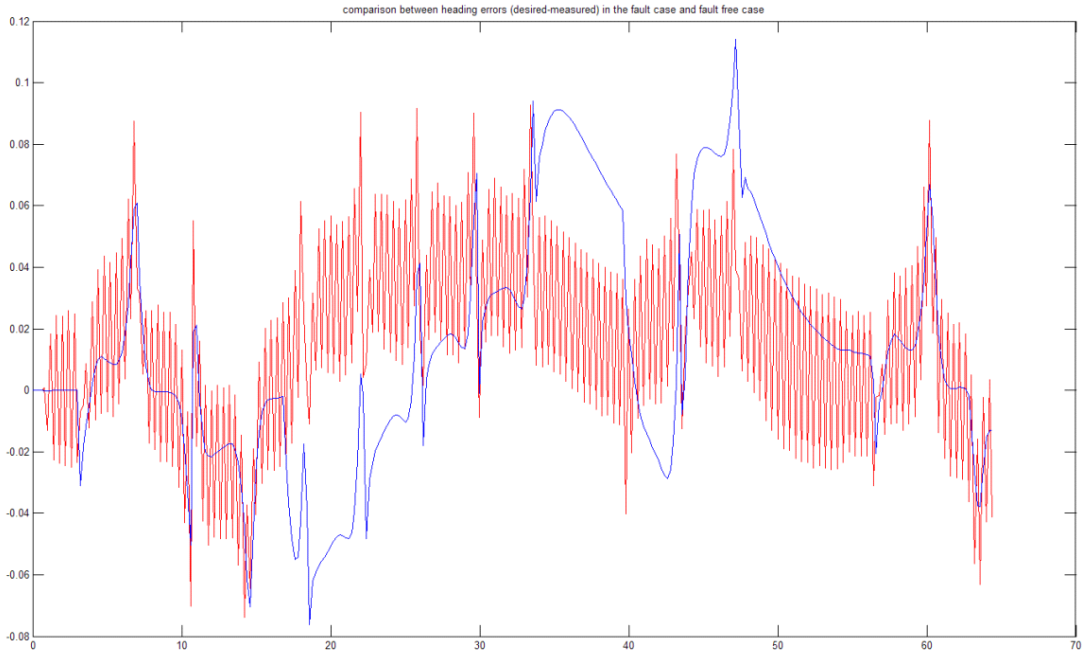


The EKF0 filter embeds a copy of the kinematical model with no modifications. It predicts the nominal behavior of the robot (fault free case). The EKF1 filter embeds a copy of the kinematical model with altered parameters for the first fault (smaller right wheel radius), it shall predict the behavior in the presence of this specific fault and so it generates a specific residual. In a similar way the EKF3, EKF4 filters are embedding the model with altered parameters for the wheel radii to predict the behavior in the presence of a periodical bump of the wheel. After a fault is detected all residuals are compared and the smallest is selected to be the one that is representative for that fault. A minimal residual is obtained in the context that the

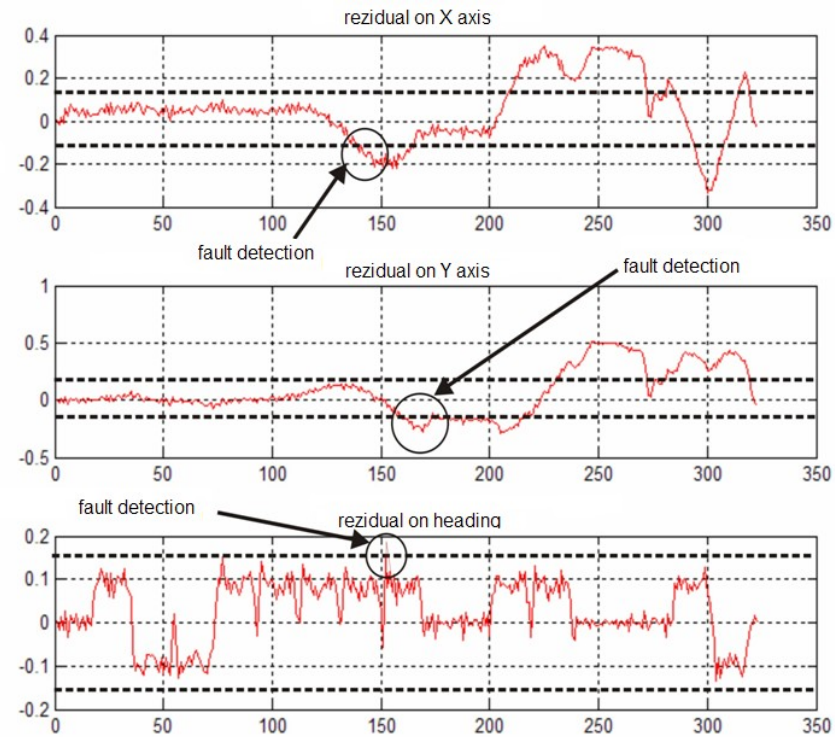
measurement vector gives a specific behavior in the presence of the fault and the filter should predict a slightly equal value for the measurement vector based on the altered model. Next a short analysis on the behavior of the system is described in the first fault injected context. So the error vector components (Xe, Ye, heading error) are described next to emphasize the nominal (fault free) behavior and the faulty one for the demo trajectory.







As one can see after injecting the fault the behavior of the robot is altered and it can easily be observed that is relatively easy to discriminate a fault by analyzing the residual. Relatively to the normal behavior the error gets higher values now that the fault was injected and the fault detection module has to signal the appearance of a fault. The residual (generated by the EKF0) is next depicted and as one can see at sample 100 the fault is injected and the residual gets over the preset thresholds and that is the moment when the fault is detected.



The nominal filter generated residual was depicted to extract the proper features of the implemented method. After detecting the fault this residual is compared with the other filters residuals and as it was mentioned earlier the smallest residual in fact emphasizes the fault that occurred because that specific filter predicts a measurement vector that has close values to the real measurement vector affected by the fault. The control reconfiguration is a future work idea and the current application offers full support to implement this new feature. The injection framework is based on simple signaling some events to the extent that for demo purposes the Java client application user can select an injection moment in the robot operation and inject the fault at that specific moment. To overcome the fact that I couldn't implement the faults in hardware I've selected to

implement them in software, and also as presented the fault detection and identification method is based on analytical redundancy. So a specific fault is injected by altering specific odometry information with the modified parameters (wheels radii) in computing linear and angular speed used to determine the robots' pose. So the measurement vector is now reflecting the injected fault behavior as it was if the real (physical) fault occurred.
The presented approach is based on an extensive literature survey regarding Sliding mode implementations for trajectory tracking mobile robot control and fault diagnosis using EKF filters mainly used in flight control or chemical/batch processes. The innovative aspect in this approach is the regarding the flexibility of the software application and the specific mechanisms implemented to gain proper results in controlling and diagnosing the self made differential mobile robot. Some representative images of the robot in all the building stages are next depicted.