Autonomous SoC for Fuzzy Robot Path Tracking

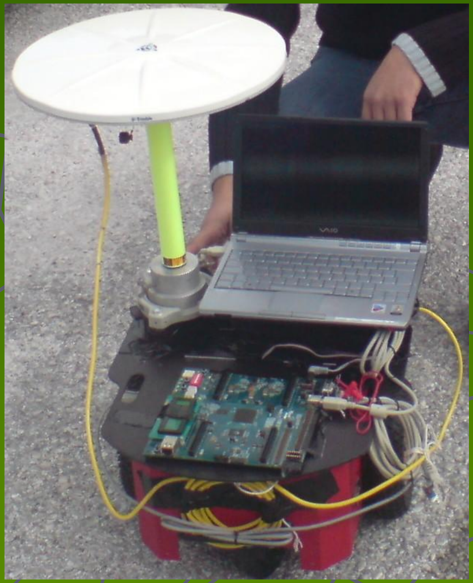
Kyriakos M. Deliparaschos, George P. Moustris, Spyros G. Tzafestas

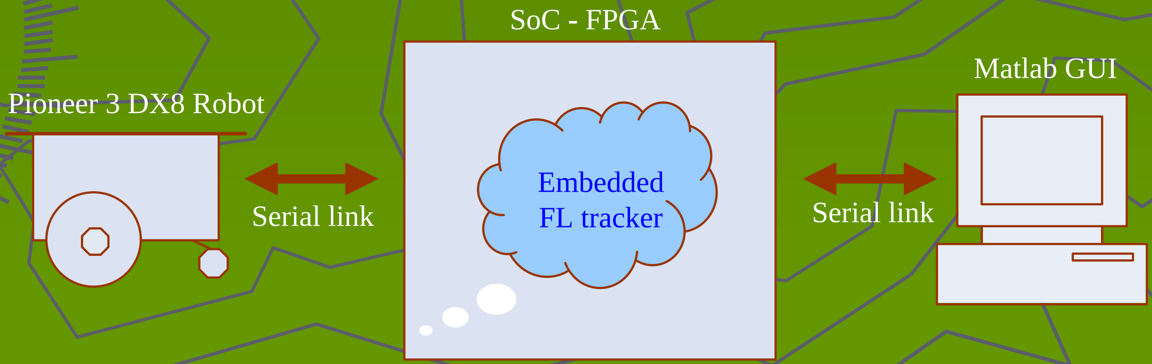
School of Electrical and Computer Engineering
National Technical University of Athens
Intelligent Robotics and Automation Laboratory

e-mails: kdelip@mail.ntua.gr
gmoustri@central.ntua.gr
tzafesta@softlab.ntua.gr

Overview of the System

- Robot platform:
  ActivMedia P3-DX8
  - 1 mm resolution for the position estimation and 1° angle resolution for the heading
  - Kinematics emulated to a bounded curvature vehicle
- SoC FPGA:
  Xilinx XC3S1500-4FG676C Spartan-3 FPGA
  - Parameterized Digital Fuzzy Logic processor (DFLP) Intellectual Property (IP) core implementing the Fuzzy Tracking algorithm
  - Xilinx Microblaze™ soft processor core as the top-level flow controller
- Matlab GUI:
  Visualization/Monitoring running on Laptop





Path Tracking

- Let F be a nonlinear system of the form

$$\dot{p} = f(t, p, u)$$

where *p* is the state vector and *u* the input. If p_{ref} is a feasible reference path in the state space which corresponds to a feasible reference input u_{ref} then find an appropriate state feedback law $u(p, p_{ref}, u_{ref}, t)$ such that

$$\lim_{t\to\infty}(p - p_{ref}) = 0$$

Path Tracking

- Former problem known as trajectory tracking because reference trajectory is parameterized by time
- If p_{ref} is a geometric reference path (no temporal parameterization) then we get the "path tracking problem" {e.g. $p_{ref}=(x,y)$ where (x,y) are the Cartesian coordinates of the robot}
- For the Dubin's Car model:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta \\ \sin\theta \\ \kappa \end{bmatrix} v$$

$p_{ref}=(x_{ref}, y_{ref})$, where (x,y) is the middle point of the robot axis.

- Speed is constant
- Control input is the *curvature* κ

Constraints

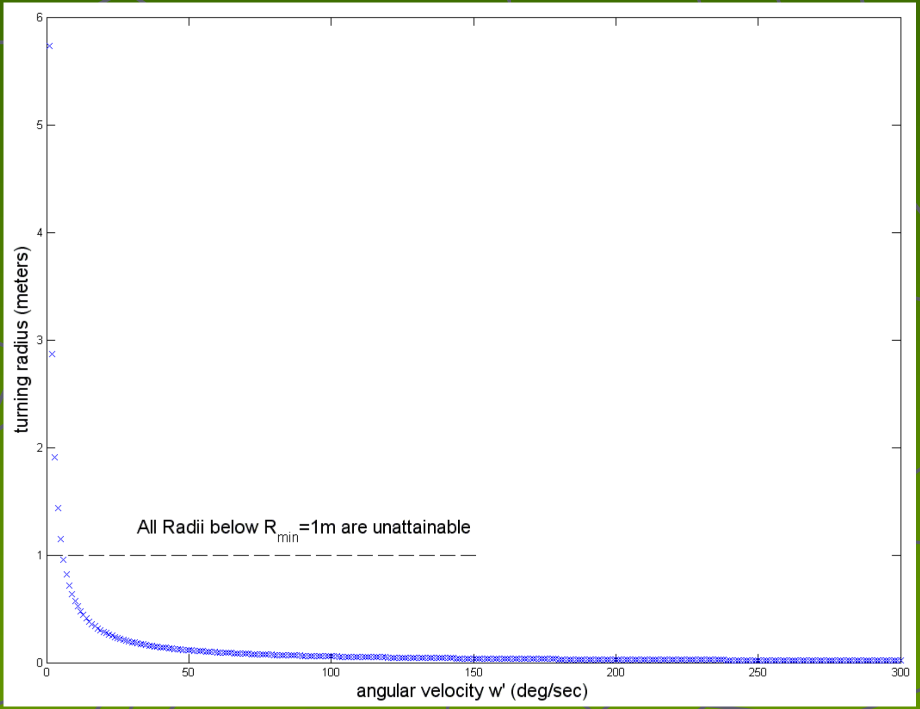
- Non-holonomic: *rolling without slipping*
- Rigidity: *robot dimensions remain fixed*
- Input bounds: *robot cannot turn while stopped (bounded κ)*
- Quantization: *robot quantizes states and inputs*
  - No explicit command for curvature. Curvature command is issued through the control of the linear and angular velocities
  - Angular velocity quantized to 1 deg/sec/bit resolution
  - Linear Velocity quantized to 1 mm/sec/bit resolution
  - RESULT → Turning radius: $\kappa^{-1}=R=c\ v/\omega$ is quantized

Quantization

- Quantization presents a problem for the control input. If the curvature is bounded to |κ|≤1 m-1, a velocity of v=100 mm/sec results to 6 available
input commands
- Bounding constraint reduces further the available levels
- Performance severely degraded. Results to oscillations
- Putting R=1 and solving w.r.t ω we get the available quantization levels:

$$L_{num} = \left\lfloor \upsilon \cdot \frac{180}{1000 \cdot \pi} \right\rfloor$$

- Linear dependence on v. Obvious solution is to increase velocity but if v is large, dynamics might come into play



Setting the Speed

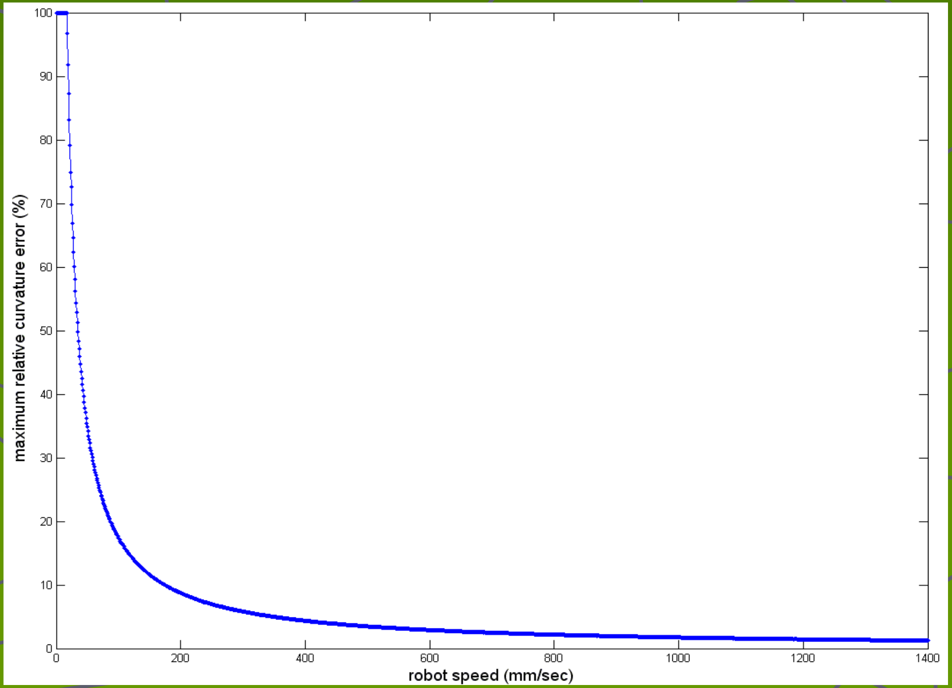
To estimate an acceptable error level between the curvature computed by the DFLP and the actual curvature the robot follows, we draw the maximum relative error versus all available speeds over all available inputs i.e.,

$$100 \cdot \max(\kappa_{FLC} - \kappa_{ACTUAL}) / \kappa_{ACTUAL}, \forall \varphi_1, \varphi_2$$

In this way one can see the maximum possible relative error for each speed between the actual and the desired curvature.

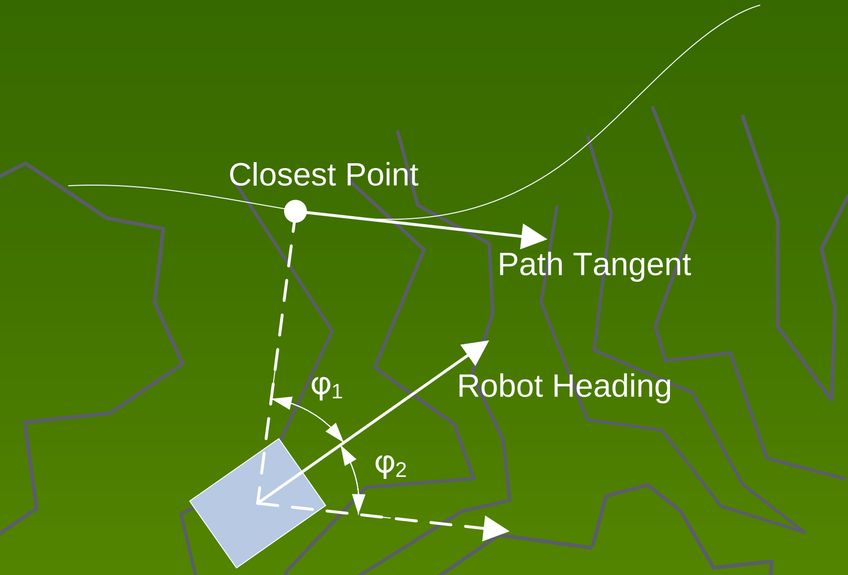
An acceptable trade-off speed seems to be 1000 mm/sec (1m/sec) where the error drops below 1.745%.

For this speed, the available quantization levels are *Lnum*=57. As a result, the robot's speed was set to 1000 mm/sec in all field experiments.



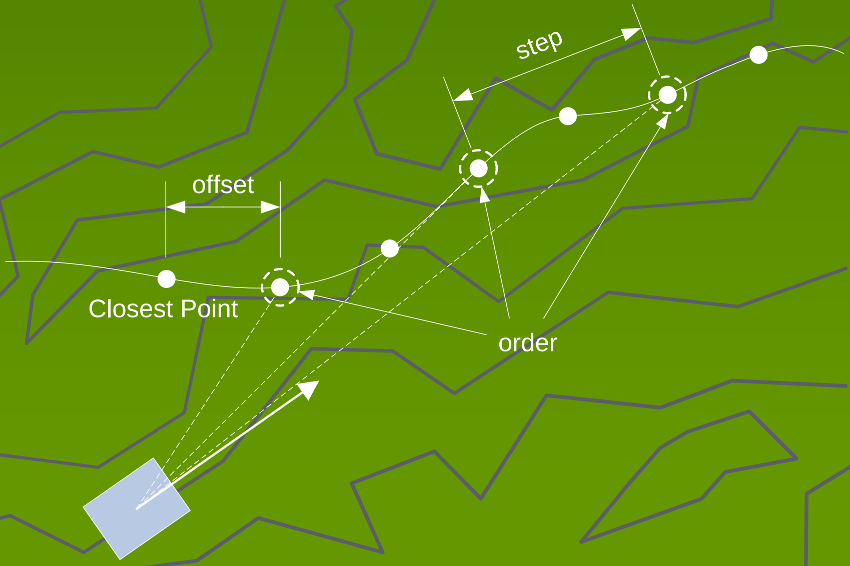
Control Strategy

- Path sampled under fixed sampling spacing *Δs*
- Inputs: Angle error φ_1
  Heading error φ_2
- Output: Curvature *κ*



- Spatial Window
  - Offset (Start at "*offset*" points from closest)
  - Order (iterate over "*order*" path points)
  - Step (skip "*step*" points at each iteration)
- Output: Mean of curvatures

$$\kappa = \frac{\kappa_1 + \kappa_2 + \ldots + \kappa_n}{n}$$



FPGA Development Platform Specifications

- The SoC Application was implemented on a Spartan-3 Development Platform

Xilinx XC3S1500-4FG676C Spartan-3 FPGA
16 M x 16 DDR memory, 2 M x 16 flash memory
Platform Flash ISP PROMs
10/100 Ethernet PHY, USB 2.0 and RS232
2 7-segment LED displays
4 User LEDs, 2 Push Buttons, 8 Pos. Dip Switches
On-board clock oscillator
JTAG configuration port
75 MHz Clock Oscillator
2 x 16 Character LCD
Two P160 expansion slots
System ACE/User I/O Header
LVDS tx/rx interface

Fuzzy Path Tracker (DFLP IP Core)

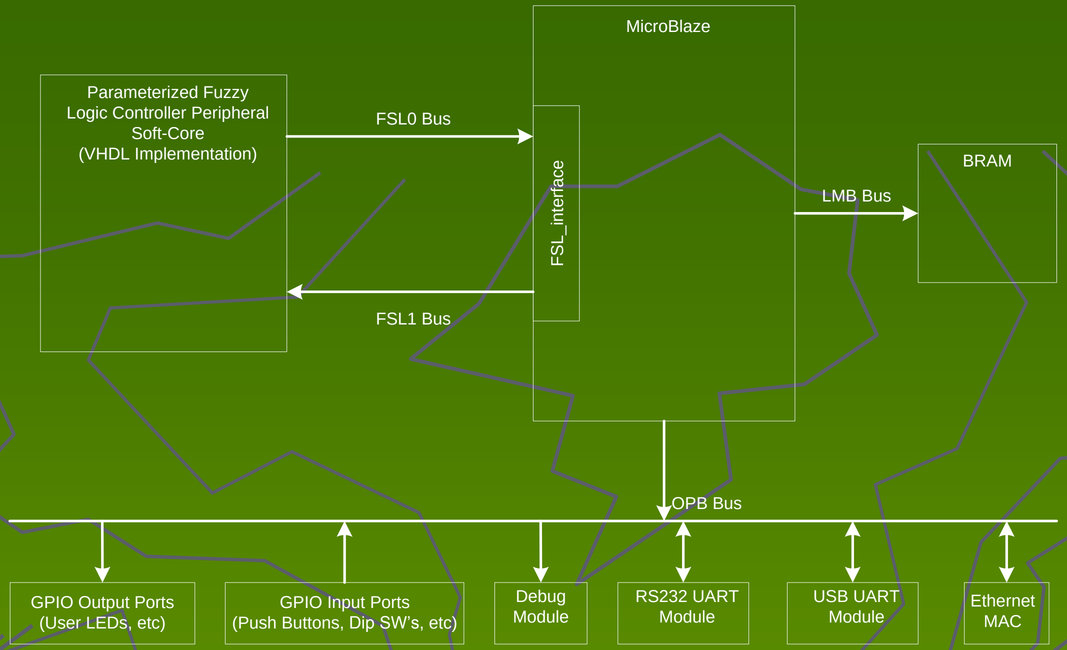
The DFLP IP Core is fully parameterized. The selected architecture assumes overlap of two fuzzy sets between adjacent fuzzy sets and requires 2^n clock cycles (input data processing rate), where *n* is the number of inputs, since it processes one active rule per clock cycle.

Characteristics

- TS Zero-Order Fuzzy Logic Controller
- 2 Inputs, 1 Output
- 9 Triangular MFs per Input
- 5 Singleton Output MFs
- 81 Fuzzy Rules
- Implication method: Product
- Defuzzification method: Weighted average

Fuzzy Inference System (FIS) type	Takagi-Sugeno zero-order type
Inputs	2
Input resolution	12 bit
Outputs	1
Output resolution	12 bit
Antecedent Membership Functions (MF's)	9 Triangular or Trapezoidal shaped per fuzzy set
Antecedent MF Degree of Truth (α value) resolution width	8 bit
Consequent MF's	81 Singleton type
Consequent MF resolution	8 bit
Max. no. of fuzzy inference rules	81 (no. of fuzzy sets $^{\text{no. of inputs}}$)
AND method	MIN (T-norm operator implemented by minimum)
Implication method	PROD (product operator)
MF overlapping degree	2
Defuzzification method	Weighted average

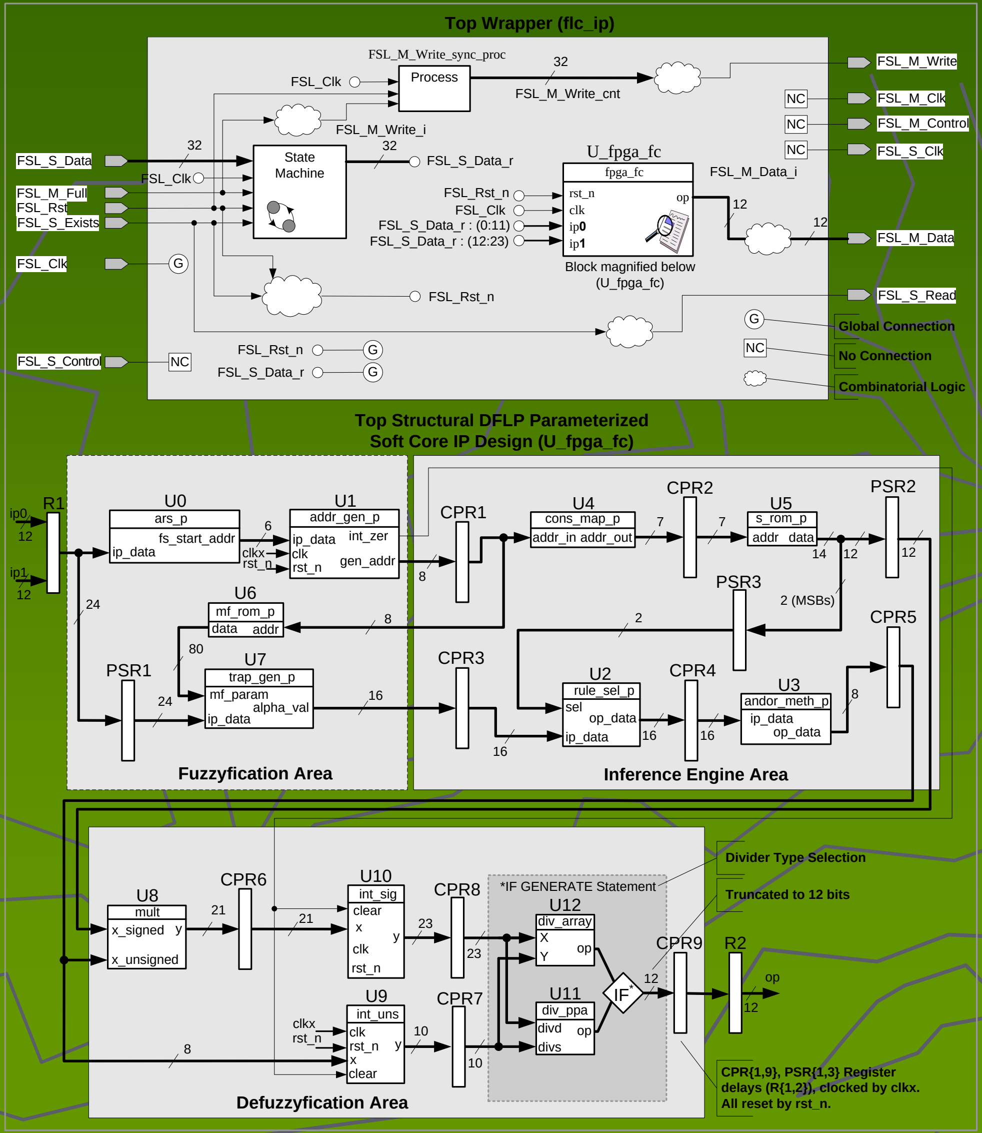
SoC Architecture - High Level Diagram


MicroBlaze
Parameterized Fuzzy Logic Controller Peripheral Soft-Core (VHDL Implementation)
FSL0 Bus
FSL1 Bus
FSL_interface
LMB Bus
BRAM
OPB Bus
GPIO Output Ports (User LEDs, etc)
GPIO Input Ports (Push Buttons, Dip SW's, etc)
Debug Module
RS232 UART Module
USB UART Module
Ethernet MAC

- Microblaze™ is licensed as part of the Xilinx Embedded Development Kit (EDK) and is based on RISC architecture
- It is a soft core, meaning that it is implemented using general logic primitives rather than a hard dedicated block on the FPGA
- The DFLP IP core is connected to the Microblaze™ via the FSL bus
- The processor connects to the OPB bus for access to a wide range of different modules, and communicates via the LMB bus for a fast access to BRAM inside the FPGA

SoC Architecture – Detailed Diagram

- Component pipeline registers (CPR)
- Path synchronization registers (PSR)
- ‘U’ blocks represent the different components of the DFLP IP core
- U_fpga_fc component is embedded in flc_ip top structural entity wrapper
- Two Digital Clock Manager (DCM) Modules (DCM_0 and DCM_1) used for the different clocks production
- SoC achieves a system clock (DCM_0) operating frequency of 14.140ns or ~71 MHz, while DCM_1 is mainly used for clocking the external DDR RAM



FPGA Design Resources

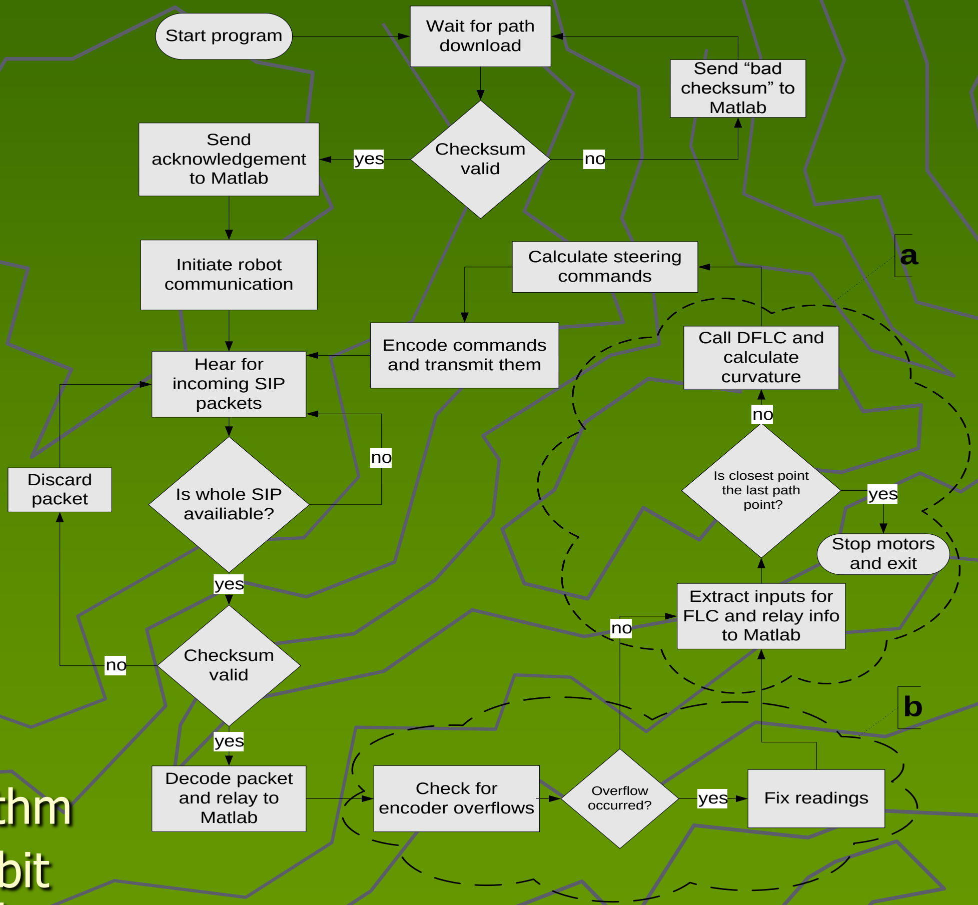
- U_fpga_fc alone was synthesized using Synplify Pro synthesizer tool
- The rest of the design components were synthesized by Xilinx Synthesis Tool (XST) through the EDK Platform Studio
- The DFLP IP core alone for the selected parameters on this application occupies 1600 (6%) LUTs, 4 Block Multipliers (MULT18X18s), 12 64x1 ROMs (ROM64X1), and 56 256x1 ROMs (ROM256X1)

Resource	Used	Available	Utilization
BSCANs	1	1	100%
BUFGMUXs	6	8	75%
DCMs	2	4	50%
External IOBs	121	487	24%
LOCed IOBs	120	121	99%
MULT18X18s	11	32	34%
RAMB16s	16	32	50%
Slices	4021	13312	30%
SLICEMs	668	6656	10%
Total LUTs:	5,956	26,624	22%

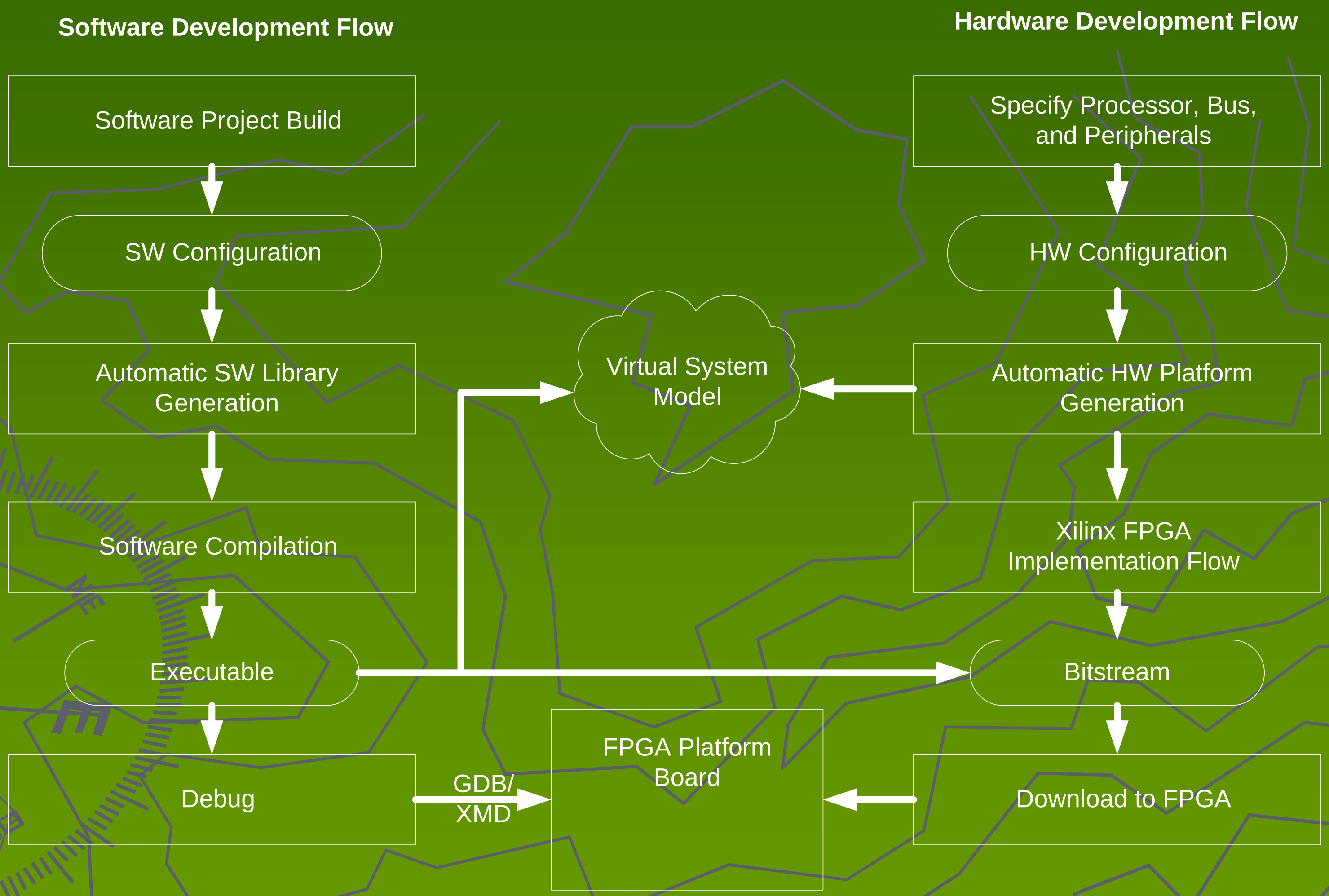
Top-Level Control Program

Written in C and executed in the Microblaze™ soft processor core

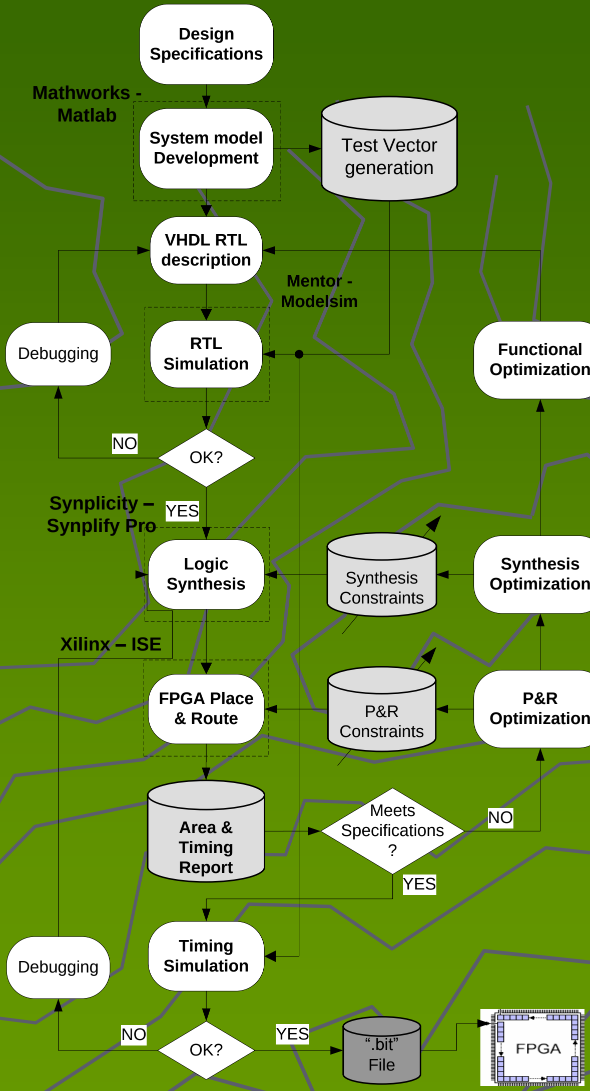
- Implements the autonomous control logic of the P3 robot
- Receives odometry information from the robot and issues steering commands outputted by the FL tracker
- Treats synchronization and timing requirements
- For the communication with the outside world (Robot, Laptop) uses two I/O channels, one serial and one Serial2USB bridge, both having 16-byte input and output buffers
    - (a) Path tracking control algorithm
    - (b) "fix" algorithm to avoid 16-bit integer coordinates overflow when their range is exceeded


Start program
Wait for path download
Send "bad checksum" to Matlab
Send acknowledgement to Matlab
yes
Checksum valid
no
Initiate robot communication
Calculate steering commands
a
Encode commands and transmit them
Call DFLC and calculate curvature
Hear for incoming SIP packets
no
no
Discard packet
Is whole SIP availiable?
Is closest point the last path point?
yes
Stop motors and exit
yes
Extract inputs for FLC and relay info to Matlab
no
Checksum valid
no
yes
b
Decode packet and relay to Matlab
Check for encoder overflows
Overflow occurred?
yes
Fix readings

Hardware/Software Co-design Flow


Software Development Flow
Hardware Development Flow
Software Project Build
SW Configuration
Automatic SW Library Generation
Software Compilation
Executable
Debug
GDB/ XMD
Virtual System Model
FPGA Platform Board
Specify Processor, Bus, and Peripherals
HW Configuration
Automatic HW Platform Generation
Xilinx FPGA Implementation Flow
Bitstream
Download to FPGA

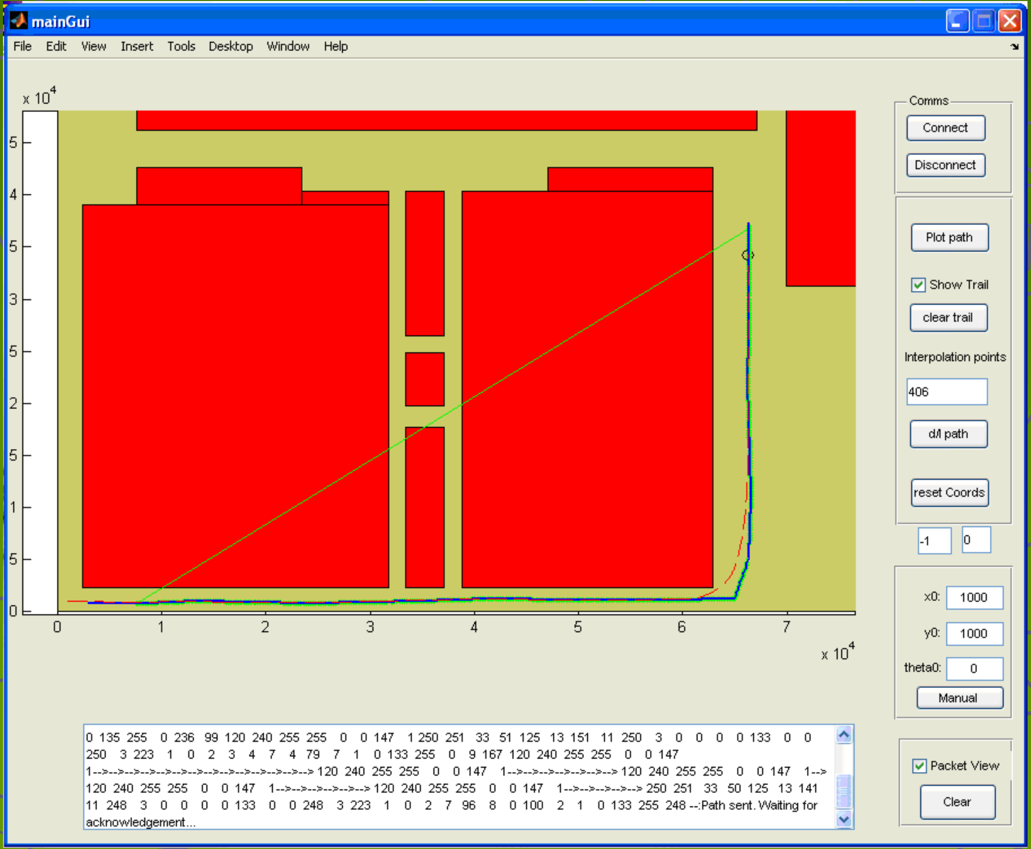
DFLP IP Core design Flow



Matlab GUI Application

A Matlab program was developed for monitoring and initialization purposes. Matlab is connected to the FPGA through a bridged USB connection. It receives and analyzes data relayed by the SoC, mainly the SIP packets that the robot sends. The program decodes the SIP packets and extracts odometry information. It also incorporates the same routine used in the SoC for catching and fixing encoder overflows

Snapshot of the GUI after an experiment. The solid line represents the desired path while the dashed line the actual path. The map illustrates part of the 2nd floor of the Electrical & Computer Engineering faculty of NTUA. All units are in millimeters.



Interpolation points: 406

-1 0

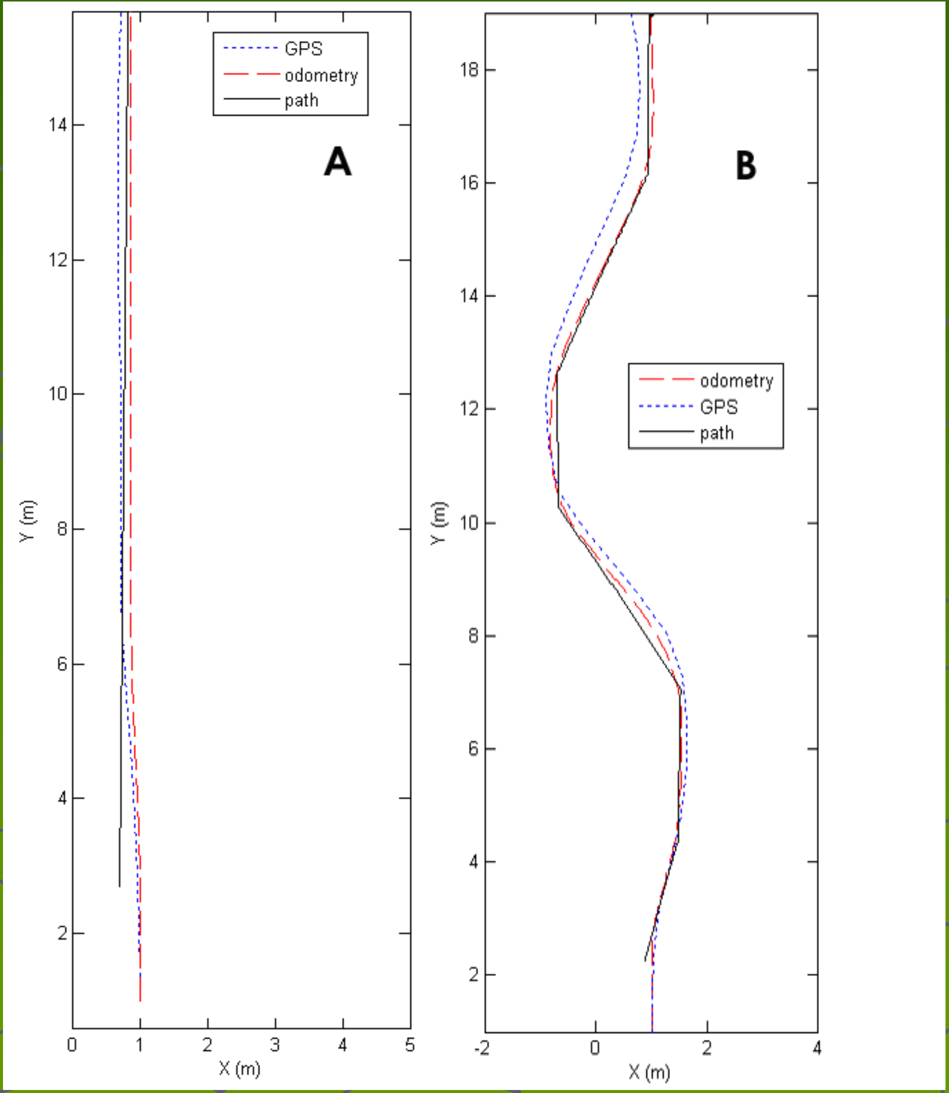
x0: 1000
y0: 1000
theta0: 0

✓ Show Trail
✓ Packet View

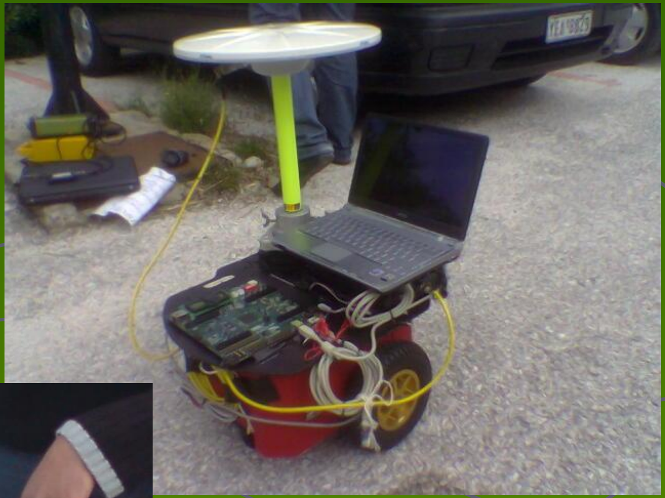
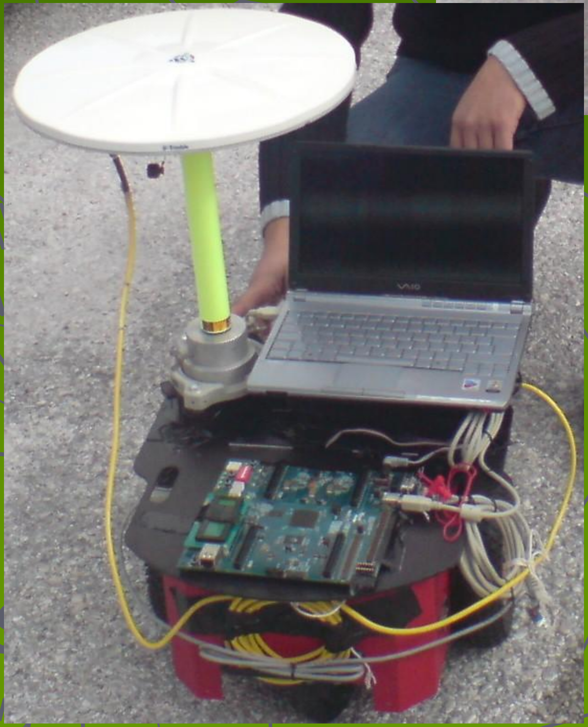
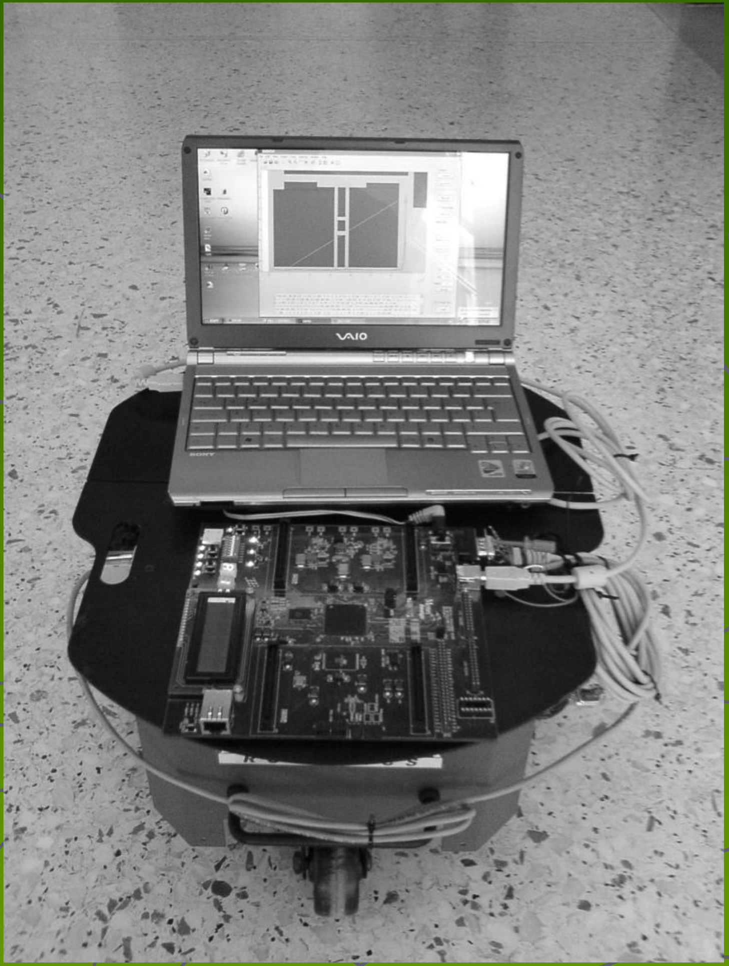
```
0 135 255   0 236  99 120 240 255 255   0   0 147   1 250 251  33  51 125  13 151  11 250   3   0   0   0   0 133   0   0
250   3 223   1   0   2   3   4   7   4  79   7   1   0 133 255   0   9 167 120 240 255 255   0   0 147
1-->-->-->-->-->-->-->-->-->-->-->-->-->--> 120 240 255 255   0   0 147   1-->-->-->-->-->-->--> 120 240 255 255   0   0 147   1-->
120 240 255 255   0   0 147   1-->-->-->-->-->--> 120 240 255 255   0   0 147   1-->-->-->-->--> 250 251  33  50 125  13 141
11 248   3   0   0   0   0 133   0   0 248   3 223   1   0   2   7  96   8   0 100   2   1   0 133 255 248 --:Path sent. Waiting for
acknowledgement...
```

Experiments

- Experiments took place inside the NTUA campus. The goal was to assess the overall efficiency of the system and particularly the fuzzy tracker. In order to log the actual position of the robot during the runs, a DGPS antenna and receiver were mounted onto it. The DGPS system used is the Trimble 4700 GPS receiver. The GPS was set to Kinematic Survey mode where the path is solved in post-processing. In this mode the horizontal precision is ±1cm+1ppm for a baseline under 10Km
- Localization was done through odometry, not DGPS. The DGPS was used only to get the actual position. Thus a degraded GPS solution is useless and positional data of a Q factor greater than 1 (with 1 being the best and 6 the worst) have been discarded



Showcase

Conclusions

- A novel SoC for the path following task of autonomous non-holonomic mobile robots was presented.
- The latency of the control is very small although it is bounded by the response of the controlled system i.e. the robot.
- Simulations and field experiments showed that the fuzzy tracking algorithm, introduced by the authors, and the overall system performance is satisfactory even under the limitations presented by the real system, e.g. the quantization of available steering commands. This is due to the high robustness exhibited by the fuzzy tracking algorithm along with the “smoothing” behavior of the spatial window technique inserted in the control loop.