Fuzzy logic path tracking control for autonomous non-holonomic mobile robots: Design of System on a Chip

S.G. Tzafestas, K.M. Deliparaschos, G.P. Moustris *

Intelligent Automation Systems Research Group, School of Electrical and Computer Engineering, National Technical University of Athens, Zographou Campus, Athens, GR 157 73, Greece

A R T I C L E   I N F O

Article history:
Received 29 March 2009
Received in revised form 26 March 2010
Accepted 31 March 2010
Available online 22 April 2010

Keywords:
Path tracking
Fuzzy logic
Mobile robots
Digital Fuzzy Logic Controller (DFLC)
System-on-a-Chip (SoC)

A B S T R A C T

This paper presents a System on Chip (SoC) for the path following task of autonomous non-holonomic mobile robots. The SoC consists of a parameterized Digital Fuzzy Logic Controller (DFLC) core and a flow control algorithm that runs under the Xilinx Microblaze soft processor core. The fuzzy controller supports a fuzzy path tracking algorithm introduced by the authors. The FPGA board hosting the SoC was attached to an actual differential-drive Pioneer 3-DX8 robot, which was used in field experiments in order to assess the overall performance of the tracking scheme. Moreover, quantization problems and limitations imposed by the system configuration are also discussed.

© 2010 Elsevier B.V. All rights reserved.

1. Introduction

This work presents a System on Chip (SoC) implementation for the robot path tracking task using fuzzy logic. The major components the SoC is composed of, are a parameterized Digital Fuzzy Logic Controller (DFLC) soft IP core [1,2], implementing the fuzzy tracking algorithm, and a Xilinx Microblaze soft processor core as the top level flow controller. The FPGA board hosting the SoC was mounted to an actual differential-drive Pioneer 3-DX8 robot, which was used in experiments of the tracking scheme.

FPGAs provide several advantages over single processor hardware, on the one hand, and Application Specific Integrated Circuits (ASIC) on the other. FPGA chips are field-upgradable and do not require the time and expense involved with ASIC redesign. Being reconfigurable, FPGA chips are able to keep up with future modifications that might be necessary. They offer a simpler design cycle, reprogrammability, and have a faster time-to-market, since no fabrication (layout, masks, or other manufacturing steps) time is required, when compared to ASICs.

A design on an FPGA could be thought as a “hard” implementation of program execution. The Processor based systems often involve several layers of abstraction to help schedule tasks and share resources among multiple processes. The driver layer controls hardware resources and the operating system manages memory and processor bandwidth. Any given processor core can execute only one instruction at a time, and processor based systems are continually at risk of time critical tasks preempting one another. FPGAs on the other hand do not use operating systems, and minimize reliability concerns with true parallel execution and deterministic hardware dedicated to every task. Today’s FPGAs contain hundreds of powerful DSP slices reaching frequencies up to 500 MHz outperforming DSP and RISC processors by a factor of 100 to 1000. Taking advantage of hardware parallelism, FPGAs exceed the computing power of digital signal processors (DSPs) by breaking the paradigm of sequential execution and accomplishing more per clock cycle.

The use of field programmable gate arrays in robotic applications is noted in [3–6] by several other researchers. A review of the application of FPGAs in robotic systems is provided be Leong and Tsoi in [3]. A notable case study is the use of FPGAs in the Mars Pathfinder, Mars Surveyor ‘98, and Mars Surveyor ‘01 Lander crafts, analyzed in [6].

Due to an increased number of calculations necessary for the path tracking control, a high performance processing system to efficiently handle the task is required. By using a SoC realized on an FPGA device, we utilize the hardware/software re-configurability of the FPGA to satisfy the needs of fuzzy logic path tracking for autonomous robots for high-performance onboard processing and flexible hardware for different tasks. Software Fuzzy Logic Controller (FLC) implementations suffer from speed limitations
due to the sequential program execution and the fact that standard processors do not directly support many fuzzy operations (i.e., minimum or maximum). In an effort to reduce the lack of fuzzy operations, several modified architectures of standard processors to support fuzzy computation exist [7–9]. Software solutions running on these devices speed up fuzzy computation by at least one order of magnitude over standard processors, but are still not fast enough for some real-time applications; a dedicated hardware implementation [10] must be used.

In our application the DFCL facilitates scaling and can be configured for different numbers of inputs and outputs, numbers of triangular or trapezoidal fuzzy sets per input, numbers of singletons per output, antecedent method (t-norm, s-norm), divider type, and numbers of pipeline registers for the various components in the model. This parameterization enabled the creation of a generic DFCL soft IP core that was used to produce a fuzzy controller of different specifications without the need of redesigning the IP from the beginning. The fuzzy logic controller architecture assumes overlap of two fuzzy sets among adjoining fuzzy sets, and requires $2^n$ ($n$ is the number of inputs) clock cycles at the core frequency speed in order to sample the input data (input sample rate of 56.34 ns), since it processes one active rule per clock cycle. In its present form the SoC design achieves a core frequency speed of 71 MHz. To achieve this timing result, the latency of the chip architecture involves 9 pipeline stages each one requiring 14.085 ns. The featured DFCL IP is based on a simple algorithm similar to the zero-order Takagi–Sugeno inference scheme and the weighted average defuzzification method. By using the chosen parameters of Table 2, it employs two 12-bit inputs and one 12-bit output, 9 triangular membership functions (MFs) per input and 5 singleton MFs at the output with 8-bit and 12-bit degree of truth resolution respectively. The rule base consists of 81 rules.

The fuzzy tracking algorithm used, is based on a previous fuzzy path tracker developed by the authors [11]. The fuzzy logic (FL) tracker has undergone some alterations due to the hardware restrictions posed by the DFCL soft IP core. While the original fuzzy logic controller (FLC) was of the Mamdani-type with Gaussian membership functions, the one deployed here is of the Takagi–Sugeno zero-order type FLC with triangular membership functions and an overlap of two between adjacent membership functions. Besides the FLC, the “spatial window” technique used in the previous paper [11] has also been incorporated in the tracking scheme.

2. Overview of the system

Four cooperative functional units properly tied together integrate the system. An overview can be seen in Fig. 1, and an actual picture of the system in Fig. 13.

The SoC implements the autonomous control logic of the P3 robot. It receives odometry information from the robot and issues steering commands outputted by the FL tracker. The SoC realizes several other tasks besides the steering control; namely it decodes the information packets sent by the robot, which include the pose estimation done by the robot, the status of the motors, sonar readings etc, and encodes the steering commands in a data frame that is accepted by the robot. In other words, the SoC implements a codec for the I/O communication with the P3 robot. Furthermore, it also relays some critical information to a MATLAB monitoring application that has been developed. The top-level program that attends to all these tasks is written in C and executed by the Microblaze soft processor core. This top-level program also treats synchronization and timing requirements.

A MATLAB application was developed for monitoring and initialization purposes. It communicates with the FPGA board through a bridged USB connection and displays information about the robot’s pose and speed, as estimated by the robot’s odometry, as well as some other data used for the path tracking control. It also calculates the robot’s position relative to the world and the local coordinate systems. Another important function of the application is to provide a path for the robot to track. Given that there is no path planning routine implemented in this work, the path is drawn in the GUI by hand as a sequence of points. Consequently the program uses a linear interpolation scheme to produce all the data samples of the path under a fixed sampling spacing, i.e., the distance between two sample points on the path is constant. The application allows choosing the number of interpolation points. The aforementioned interpolation routine was chosen after field observations on different interpolation schemes such as polynomial, cubic and linear and produced the best results.

The test platform on which the SoC was tested is the ActivMedia P3-DX8 robot [12]. The robot uses 1 mm resolution for the position estimation and $1^\circ$ angle resolution for the heading. The kinematics of the robot are emulated to a bounded curvature steering vehicle and not that of a differential drive one, i.e., there is an imposed constraint on the maximum curvature it can turn with. This has been introduced because the fuzzy tracking algorithm was intended for the Dubin’s Car model [13] where there is a minimum turning radius constraint on the robot and only for forward motion. As will be explained in a later section, the curvature constraint along with the one degree resolution of the P3 robot presents a quantization problem to the curvature. The robot connects to the FPGA board through a serial cable to send and receive framed data. ActivMedia uses its own data framing protocol handled by the robot’s microcontroller. The data sent from the robot are called Server Information Packets (SIP packets) while the received data are called Command Packets. More information on the data framing protocol can be found in the robot’s manual [12, pp.33–36]. The experiments showed clearly that even though the FL tracker performs well, its actual performance is severely degraded by the accumulation of odometry errors over time. Several calibration tests have been carried out in order to improve odometry localization but, as it was expected, position estimation through odometry proved inefficient.

3. Fuzzy logic path tracking algorithm

The tracking problem can be formulated in the following way: let $\Sigma$ be a system described by Eq. (1).

$$\begin{align*}
\dot{x} &= f(t, x, u) \\
y &= h(t, x, u)
\end{align*}$$

where $x \in \mathbb{R}^n$ is the state vector, $u \in \mathbb{R}^m$ is the input vector and $y \in \mathbb{R}^p$ is the output vector. Let $x^r(t)$ be a feasible reference trajectory in the state space that satisfies Eq(1). This solution corresponds to a reference input $u^r(t)$, i.e., $x^r = f(t, x^r, u^r)$. Find a feedback law $u = u(t, x, x^r, u^r)$ such that $\lim_{t \to \infty} (x(t) - x^r(t)) = 0$. The path tracking problem can be formulated in the same manner except that the goal is to track the image of the reference trajectory $x^r(t)$.

In such a setting, the image admits a reparametrization according to some
The path is provided as a sequence of points in the direction of the closest point with respect to the current robot heading and the curvature $\kappa$ uses two angles as inputs while it provides as output the curvature $\kappa$.

A technique that has been also introduced by the authors in [11], called “spatial window”, that enhances the path tracking control, has been incorporated into the control algorithm. This technique is based on the idea of having a spatial window on the path rather than just a single point. In this arrangement a window of the path is used in order to calculate the steering command, thus introducing a “perspective” to the controller. The spatial window is defined by three parameters: the window order, which is the number of points used in the window, the window step, which is the number of points skipped for each active point, and the window offset, which is the number of points, counting from the closest, that moves the window forward on the path. Every point of the window is presented to the controller and an appropriate curvature is calculated for each one. Consequently, there are $n$ computed curvatures for each control loop, where $n$ is the window order. From these curvatures a final output curvature must be calculated. The simplest method, which was exclusively used in this paper, is the mean value of the curvatures. The spatial window technique provides a smoother path tracking control and an overall better performance.

The FLC presents several advantages over classical analytic tracking controllers which make it an ideal candidate for the problem at hand. Firstly, it can be used to track different classes of reference paths and does not depend on actual path characteristics. For example, the controller can be applied to continuously differentiable reference paths, polygonal paths or just points. The last application is the most interesting since the path is encoded as a collection of points, which reduces the computational complexity of the problem. Secondly, the FLC presents robustness with respect to localization uncertainty as well as path encoding errors (see [11] for an analysis and extensive simulation results). Furthermore, the FLC does not need fine tuning and can be used off-the-shelf. This is important as many controllers depend on gains or parameters that must be tuned in order to work with specific systems and...
uses an internal representation of the coordinate variables of type is a rather limited range. To overcome this obstacle, the program effective range of \(-15892\) to \(15892\) mm, i.e., \(-16\) to \(16\) m, which is a rather limited range. To overcome this obstacle, the program uses an internal representation of the coordinate variables of type double, which maximize the range of the localization. In this arrangement, the SIP packet’s \(x\), \(y\) data are treated as \(\Delta x\), \(\Delta y\) with the registration point being reset every time an overflow occurs. A detailed description of this work-around is depicted in the flowchart of Fig. 5(b).

After the calculation of odometry data, the program execution passes to the algorithm responsible for the path tracking control, seen in the flowchart of Fig. 5(a). This algorithm implements the spatial window technique of the path tracker. It picks the closest path point to the robot, calculates the two controller inputs of point \(j\) of the window and calls the DFLC IP for each set of inputs. Upon completion it outputs the mean of all computed curvatures. The algorithm halts the robot if it detects that it is close enough to the path end so that a full spatial window cannot be used on the path.

In order to find the closest path point, the square of the Euclidean norm is used due to the fact that the square root calculation of the standard Euclidean distance is computationally costly, hence it is avoided in this way. Moreover the variables used to compute this norm are of type long int (64-bit integer) instead of type double because the code execution is much faster this way when executed in the Microblaze IP core without the use of the FPU unit. This is of crucial importance since the main algorithm bottleneck takes place in this operation. If one considers the fact that the downloaded path may consist of hundreds of sample points, the algorithm must cycle through all of them in order to find the closest one. The latter can delay the execution long enough so that the packets sent by the robot are not flushed fast enough to the variable that contains the data, resulting in fragmented SIP packets. This leads to the loss of synchronization between the FPGA and the robot because no entire SIP packet can be reconstructed from the FPGA.

Following the calculation of the steering curvature from the previous routine, the steering commands must be computed. Since the Pioneer robot does not have an explicit command regarding the curvature, a turn of predefined curvature must be implicitly issued through the combination of two other commands; one regarding the linear velocity and one regarding the angular velocity. The
The actual curvature $\kappa$ outputted by the DFLC is an integer ranging from $-2^{11} \leq \kappa \leq 2^{11} - 1$ or $-2048$ to $2047$ (12-bit resolution). In a strict sense, the positive curvature range should be divided by 2047, but to avoid a sign check we divide by 2048. This introduces an 1 lsb error on the positive range, which is insignificant compared to the quantization error produced by the robot itself [see next section]. The curvature is normalized to $[-1, 1]$ and multiplied by a user-defined maximum curvature $\kappa_{\text{max}}$ since the differential drive does not have bounds on the turning radius. In this way, one can control the minimum turning radius that can be steered by the P3 robot. Thus, Eq. (3) is transformed to Eq. (4),

$$\omega' = \frac{\kappa}{2048} \cdot \frac{180}{1000 \cdot \pi} \cdot \omega' \cdot \frac{180}{1000 \cdot \pi} = \frac{\omega'}{\omega'} \cdot \frac{180}{1000 \cdot \pi} \cdot \frac{180}{1000 \cdot \pi} \pm 1 \text{ lsb}. \quad (4)$$

As soon as the robot's velocity is extracted from the SIP packet and the angular velocity is calculated from Eq. (4), the two motion commands are transmitted back to the robot. The resulting trajectory has an imposed bound on the curvature and the differential P3 robot emulates a simple cart. As mentioned previously, the velocity control can be decoupled from the path tracking task and can be controlled from a speed controller independently. In other applications, such as kinodynamic tracking where the dynamics of the robot are taken into account, one must also incorporate the velocity into the control loop. However in this work the velocity control is not necessary for the tracking problem, under the assumption that the speed of the robot is small enough such that the dynamics do not affect the kinematic behavior. In an actual car-like robot the curvature constraints and the actual motion are mechanically imposed, not emulated in the manner described here. The way the curvature is being emulated in this work presents a problem that is analyzed next.

5. Quantization

The P3 robot makes use of 16-bit integers to encode the SIP packet data. Accordingly, it also uses 16-bit integers to encode command arguments in a command packet. However, the range of the angular velocity command is $[0, 300]$ deg/s with one deg/sec/bit. Since $\omega'$ can take only integer values with one deg/s resolution, this imposes a quantization on the curvature as well. Solving Eq. (3) for $\kappa$ (or $R = 1/\kappa$, the turning radius in meters), one gets Eq. (5),

$$R = \frac{\omega'}{\omega'} \cdot \frac{180}{1000 \cdot \pi} \cdot \frac{180}{1000 \cdot \pi} \pm 1 \text{ lsb}. \quad (5)$$
Fig. 5. Flow-charts of (a) the spatial window routine and (b) the routine that fixes encoder overflows.

Plotting the radius $R$ versus the quantized angular velocity $\omega'$ for $v' = 100$ mm/s (see Fig. 6) one can see that the resolution towards small curvatures (large radii) is poor going from $R = 5.73$ ($\omega' = 1$) to $R = 2.86$ ($\omega' = 2$) to $R = 1.91$ ($\omega' = 3$), while the resolution towards large curvatures (small radii) is high. This has a severe impact on the way the controller can follow the path. When the robot is on the path i.e. $\phi_1, \phi_2 \approx 0$, the FLC issues commands of small curvature (large radius) to avoid “nervous” steering, e.g. oscillations. If the resolution in that range is poor, as described above, the control is degraded since the curvature commands are clipped to the available resolution levels. Furthermore, since there is a constraint on the minimum turning radius, all values of Fig. 6 below the lower bound $R_{\text{min}}$, which was set to 1 m in our case studies, are unattainable by the robot. For $v' = 100$ mm/s, there are only six radii over 1 m, i.e., only six curvatures the robot can turn with. Since $R$ decreases monotonically with respect to $\omega'$ and the latter starts from 1 deg/s, one can find the number of available quantization levels (Eq. (6)) by substituting $R = 1$ in Eq. (5) and solving for $\omega'$ i.e.,

$$L_{\text{num}} = \left\lfloor \frac{v' \cdot \frac{180}{1000} \cdot \frac{1}{\pi}}{10} \right\rfloor,$$

where $\lfloor \rfloor$ is the floor function.

By inspection of Eqs. (5) and (6) one can see that by increasing the velocity $v'$, the maximum turning radius increases linearly along with the resolution in that range, i.e., the $L_{\text{num}}$. Thus, the obvious solution for the increase of the curvature resolution is to increase the speed. Of course, this raises the problem of finding an appropriate robot speed since a low speed reduces the curvature resolution but a high speed might result in an unresponsive system.

To estimate an acceptable error level between the curvature computed by the FLC and the actual curvature the robot follows, we draw the maximum relative error versus all available speeds over all available inputs (Eq. (7)),

$$100 \cdot \max (\frac{\kappa_{\text{FLC}} - \kappa_{\text{ACTUAL}}}{\kappa_{\text{ACTUAL}}} \cdot \psi_1, \psi_2).$$

In this way one can see the maximum possible relative error for each speed between the actual and the desired curvature. This error is illustrated in Fig. 7. The minimum turning radius was set to one meter ($\kappa_{\text{max}} = 10^{-3}$). It is clear that as the speed increases the error decreases. An acceptable trade-off speed seems to be 1000 mm/s (or 1 m/s) where the error drops below 1.745%. For this speed, the available quantization levels, as derived from (6) are $L_{\text{num}} = 57$. As a result, the robot’s speed was set to 1000 mm/s in all field experiments.
6. SoC hardware architecture

The SoC design presented in this work was implemented on the Spartan-3 MB development kit (DS-KIT-3SMB1500) by Memec Design. The Spartan-3 MB system board utilizes the 1.5 million-gate Xilinx Spartan-3 device (XC3S1500-4FG676) in the 676-pin fine-grid array package. A high level and a detailed architecture view of the proposed SoC is shown in Figs. 8 and 9 respectively.

The Component Pipeline Registers (CPR) blocks in Fig. 9 indicate the number of pipeline stages for each component; the Path Synchronization Registers (PSR) blocks point to registers used for synchronizing the data paths, while the “U” blocks represent the different components of the DFLC [2].

The U_fpga_fc component is embedded in the flc_ip top structural entity wrapper which is compliant with the FSL standard and provides all the necessary peripheral logic to the DFLC soft IP core in order to send/receive data to/from the FSL bus. The flc_ip wrapper architecture is shown in Fig. 10 while the chosen (generic) parameters (VHDL package definition file) for the parameterized DFLC IP (U_fpga_fc) and its parameters are summarized in Tables 2 and 3 respectively.

The U_fpga_fc alone was synthesized using Synplify Pro synthesizer tool, while the rest of the design components were synthesized with Xilinx Synthesis Tool (XST) through the EDK Platform Studio. The produced .edf file for the U_fpga_fc has been seeing by the flc_ip wrapper as a blackbox during the XST flow. The placement and routing of the SoC design into the FPGA was done through the EDK by calling the Xilinx ISE tool.

According to the device utilization report from the place and route tool (see Table 4), the DFLC IP itself occupies 1600 (6%) LUTs, 4 Block Multipliers (MULTX18s), 12 (ROM64X1) Block RAMs, and 56 (ROM256X1) Block RAMs. The implemented design uses two Digital Clock Manager (DCM) Modules (DCM_0 for the system clock and DCM_1 for clocking the external DDR RAM) that produce the different clocks in the FPGA. The SoC achieves a system clock operating frequency of 14.140ns or ~71 MHz. A system block diagram of the implemented SoC with an explanation of the different symbols used, is shown in Fig. 10.

The Microblaze soft processor core [14] is licensed as part of the Xilinx Embedded Development Kit (EDK). The processor is a soft core, meaning that it is implemented using general logic primitives rather than a hard dedicated block in the FPGA. The Microblaze is based on a RISC architecture which is very similar to the DLX architecture described in [15]. It features a 3-stage pipeline with most instructions completing in a single cycle. Both the instruction and data words are 32-bit. The core alone can achieve a speed of up to 100 MHz on the Spartan 3 FPGA family. The Microblaze processor can connect to the On-chip Peripheral Bus (OPB) for access to a wide range of different modules, and can communicate via the Local Memory Bus (LMB) for a fast access to local memory (normally block RAMs (BRAMs) inside the FPGA).

Moreover, the Fast Simplex Link (FSL) bus system offers the ability to connect user soft IP cores acting as co-processors to accelerate time critical algorithms. The FSL channels are dedicated unidirectional point-to-point data streaming interfaces. Each FSL channel provides a low latency interface to the processor pipeline allowing the extension of the processor’s execution unit with custom soft core co-processors. In this paper the DFLC IP core plays the role of such a co-processor and is connected to the Microblaze via the FSL bus [16].

The architecture of the present SoC consists mainly of the DFLC that communicates with the Microblaze Processor through the FSL bus, the utilized BRAMs through the LMB bus, and other peripherals such as the general purpose input/output ports (GPIO), and UART modules via the OPB. The DFLC hosts the fuzzy tracking algorithm, whereas the Microblaze processor mainly executes the C code for the flow control.

7. Experiments

In this section the results of two outdoor experiments of the system are presented. The experiments took place inside the
Table 3
DFLC characteristics.

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fuzzy Inference System (FIS) type</td>
<td>Takagi–Sugeno zero-order type</td>
</tr>
<tr>
<td>Inputs</td>
<td>2</td>
</tr>
<tr>
<td>Input resolution</td>
<td>12-bit</td>
</tr>
<tr>
<td>Outputs</td>
<td>1</td>
</tr>
<tr>
<td>Output resolution</td>
<td>12-bit</td>
</tr>
<tr>
<td>Antecedent MF’s</td>
<td>9 Triangular shaped per fuzzy set</td>
</tr>
<tr>
<td>Antecedent MF’s degree of truth resolution</td>
<td>8-bit</td>
</tr>
<tr>
<td>Consequent MF’s</td>
<td>5 Singleton type</td>
</tr>
<tr>
<td>Consequent MF’s resolution</td>
<td>12-bit</td>
</tr>
<tr>
<td>Number of fuzzy inference rules</td>
<td>81</td>
</tr>
<tr>
<td>AND method</td>
<td>MIN (T-norm minimum operator)</td>
</tr>
<tr>
<td>Implication method</td>
<td>PROD (T-norm product operator)</td>
</tr>
<tr>
<td>MF overlapping degree</td>
<td>2</td>
</tr>
<tr>
<td>Defuzzification method</td>
<td>Weighted average</td>
</tr>
</tbody>
</table>

Fig. 9. DFLC soft IP core architecture [1].

NTUA campus. The goal was to assess the overall efficiency of the system and particularly the fuzzy tracker. The experiments consist of tracking two prescribed paths. The first is a straight line path and the second is an S-shaped path. In order to log the actual position of the robot during the runs, a DGPS antenna and receiver was 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 1 cm + 1 ppm for a baseline under 10 km. The occupation is 1 s, i.e. a positional sample is taken each second. The robot’s speed was set to 1 m/s in both cases. The results of the two experiments can be seen in Fig. 11, for both the straight run and the S-shape run. In the straight run experiment the robot was set to follow a 25 m straight path. The depiction in Fig. 11 does not present the entire run, but rather the segment where the GPS solution is of the highest quality (quality factor, $Q = 1$) since in order to assess the path tracker’s performance we need a high precision position estimation. This must not be confused with the position estimation module that the tracker uses, which in this case is derived from odometry data. The GPS is used in order to see the actual position of the robot. 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. The initial offset of the robot to the path was about 130 cm while the heading error with respect to the closest point was approximately 0 deg.

The second experiment presents the tracking of an S-shaped path. The same conditions regarding the GPS data also apply to this run. The path has a length of approximately 25 m. All GPS data with $Q > 1$ have been discarded. In the case the initial robot offset was about 165 cm and the heading error 40 deg. It should be noted that the S-shaped path is not actually a feasible reference path since the curvature derivative is discontinuous at the polygon vertices. However if the discontinuity is small, then the robot is expected to provide an accurate tracking. In order to infer a quantitative
Table 4
FPGA device utilization summary.

<table>
<thead>
<tr>
<th>Resource</th>
<th>Used</th>
<th>Available</th>
<th>Utilization (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BSCANS</td>
<td>1</td>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>BUFGMUXs</td>
<td>6</td>
<td>8</td>
<td>75</td>
</tr>
<tr>
<td>DCMs</td>
<td>2</td>
<td>4</td>
<td>50</td>
</tr>
<tr>
<td>External IOBs</td>
<td>121</td>
<td>487</td>
<td>24</td>
</tr>
<tr>
<td>LOCed IOBs</td>
<td>120</td>
<td>121</td>
<td>99</td>
</tr>
<tr>
<td>MULT18X18s</td>
<td>11</td>
<td>32</td>
<td>34</td>
</tr>
<tr>
<td>RAMB16s</td>
<td>16</td>
<td>32</td>
<td>50</td>
</tr>
<tr>
<td>Slices</td>
<td>4021</td>
<td>13,312</td>
<td>30</td>
</tr>
<tr>
<td>SLICEMs</td>
<td>668</td>
<td>6,656</td>
<td>10</td>
</tr>
<tr>
<td>Total LUTs</td>
<td>5956</td>
<td>26,624</td>
<td>22</td>
</tr>
</tbody>
</table>

Fig. 12. Minimum distance in meters, to the reference path of the GPS and odometry solutions, versus the normalized length.

Fig. 13. Depiction of the actual system.

interpretation of the tracking experiments, the minimum distance of the GPS and Odometry paths to the reference path has been calculated. However, since the paths are just a collection of points on the plane, albeit a rather sparse one since the GPS solution provides a fix approximately every meter, the paths have been up-sampled using a linear interpolation scheme. Furthermore a direct comparison of the GPS and Odometry solutions is unfeasible because the sampling times are not synchronized; hence one cannot compare the minimum distances of the two paths at specific time instances. Instead the distance is plotted versus the normalized length of each path with respect to the maximum length of the two (Fig. 12).

As one can see, when the robot’s pose is “close” the path’s pose, the distance is kept under 40 cm (20 cm for straight line tracking). Furthermore it can be seen that the odometry path is very close the reference path. Since the localization used by the robot is the odometry solution, if a higher precision position estimation is used with the path tracker, such as a Real-Time Kinematic DGPS data feed that provides positional data to the path tracker in real-time, the tracker will perform better. This is part of the future work of the authors. Moreover, the FPGA can easily incorporate data from other sensors and provide additional output. The inherent design of field programmable gate arrays allow for great scalability. By writing codecs similar to that of the SIP deframer in the Microblaze environment, data from more external sources can be easily manipulated.

8. Conclusion

This paper presents a novel SoC, that successfully utilizes and interfaces the parametric DFLC core with the Microblaze core, for the path following task of autonomous non-holonomic mobile robots. The scalability of the fuzzy logic controller design [2] easily allowed adapting it to the fuzzy tracker model [11] without the need of recording the core. 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, namely the quantization of available steering commands and the existence of a dead zone. 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.

References


K.M. Deliparaschos received his B.Eng. Hons degree in Electronics Engineering from De Montfort University, Leicester, UK, the M.Sc. degree in Mechatronics from De Montfort University with the collaboration of National Technical University of Athens (NTUA), and completed his Ph.D. degree at NTUA. He is currently a member of the Intelligent Automation Systems research group (IAS) at the School of Electrical & Computer Engineering of NTUA. His main research interests focuses on intelligent control systems and robotics, digital system design for high-performance FPGA architectures, VLSI systems, core design and SoC integration, and hardware/software co-design.

G.P. Moustris received his M.Eng. in Electrical & Computer Engineering from the Aristotle University, Thessaloniki, Greece and completed his Ph.D. in computational intelligence and robotics at the National Technical University of Athens (NTUA), Athens, Greece. He has been with the NTUA since 2003 and is currently a member of the Intelligent Automation Systems research group (IAS) at the School of Electrical & Computer Engineering. His fields of interest include mobile robotics, computational and artificial intelligence, non-holonomic systems and non-linear control.