# We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

6,900
Open access books available

185,000
International authors and editors

200M
Downloads

Our authors are among the

154
Countries delivered to

TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities




WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index in Web of Science™ Core Collection (BKCI)

## Interested in publishing with us?
## Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com



# 20

# Feedback Equivalence and Control of Mobile Robots Through a Scalable FPGA Architecture

G.P. Moustris[1], K.M. Deliparaschos[2] and S.G. Tzafestas[1]
[1]*National Technical University of Athens*
[2]*Cyprus University of Technology*
[1]*Greece*
[2]*Cyprus*

## 1. Introduction

The control of mobile robots is an intense research field, having produced a substantial volume of research literature in the past three decades. Mobile robots present a challenge in both theoretical control design as well as hardware implementation. From a theoretical point of view, mobile robots exert highly non-linear kinematics and dynamics, non-holonomy and complex operational environment. On the other hand, hardware designs strive for scalability, downsizing, computational power and low cost.

The main control problems in robot motion control can be classified in three general cases; stabilization to a point, trajectory tracking and path following. Point stabilization refers to the stabilization of the robot to a specific configuration in its state space (pose). For example, if the robot at $t = t_0$ is at $p_0 = (x_0, y_0, \theta_0)$, then find a suitable control law that steers it to a goal point $p_g$=($x_g$, $y_g$, $\theta_g$). Apparently $p_0$ must be an equilibrium point of the closed loop system, exerting asymptotic stability (although practical stability can also be sought for). In the path following (or path tracking) problem, the robot is presented with a reference path, either feasible or infeasible, and has to follow it by issuing the appropriate control commands. A path is defined as a geometric curve in the robot's application space. The trajectory tracking problem is similar, although there is a notable difference; the trajectory is a path with an associated timing law i.e. the robot has to be on specific points at specific time instances.

These three problems present challenges and difficulties exacerbated by the fact that the robot models are highly non-linear and non-holonomic (although robots that lift the non-holonomic rolling constraint do exist and are called *omni-directional robots*. However the most interesting mobile robots present this constraint). The non-holonomy means that there are constraints in the robot velocities e.g. the non-slipping condition, which forces the robot to move tangentially to its path or equivalently, the robot's heading is always collinear to its velocity vector (this can readily be attested by every driver who expects his car to move at the direction it is heading and not sideways i.e. slip). For a more motivating example of holonomy, consider a prototypical and pedagogical kimenatic model for motion analysis and control; the unicycle robot, described by the equations,

$$\Sigma : \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta \\ \sin\theta \\ 0 \end{bmatrix} v + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \omega \tag{1}$$

This model is linear on the inputs and describes a linear combination of vector fields which evolve on a 3D configuration space $M = \mathbb{R}^3 \times S^1$ (a three dimensional manifold). The robot is controlled by two inputs $v$ and $\omega$, expressing the linear and angular velocities respectively. The generalized velocities live on the manifold's tangent space $T_qM$ at each point q, thus the system's equations express the available directions of movement. The non-holonomic no-slipping constraint is expressed by the Pfaffian equation,

$$0 = \dot{x}\sin(\theta) - \dot{y}\cos(\theta) \tag{2}$$

which can be put into the general form,

$$G(q)\dot{q} = 0 \tag{3}$$

Here $q \in M$ is the state vector and $G(q) \in \mathbb{R}^{1\times 3}$ is the constraint matrix (although in this case is just a vector). Each row vector (covector) of G lives in the manifold's cotangent space $T_q^*M$ at q, which is the dual space of $T_qM$. Equation 3 describes restrictions on the available directions of movement. Apparently, since the velocities must satisfy (3), it is evident that they live in the null space of G. One can move from Eq.(3) to Eq.(1) by solving (3) with respect to the velocities $\dot{q}$. Since the system is underdetermined (note that G is a one by three "matrix"), two generalized velocities can vary freely, which are precisely the two inputs of (1). The non-holonomy of the system derives from the fact that Eq.(2) is not integrable i.e. does not express the total derivative of some function. By the Frobenius theorem, if $\Delta$ is the distribution spanned by the two vector fields of $\Sigma$, the system is holonomic if $\Delta$ is involutive under Lie bracketing, a condition that is not satisfied by (1).

Due to these challenges, the path following problem has been attacked by several researchers from many angles, ranging from classical control approaches (Altafini, 1999; Kamga & Rachid, 1997; Kanayama & Fahroo, 1997), to nonlinear control methodologies (Altafini, 2002; Egerstedt et al., 1998; Koh & Cho, 1994; Samson, 1995; Wit et al., 2004) to intelligent control strategies (Abdessemed et al., 2004; Antonelli et al., 2007; Baltes & Otte, 1999; Cao & Hall, 1998; Deliparaschos et al., 2007; El Hajjaji & Bentalba, 2003; Lee et al., 2003; Liu & Lewis, 1994; Maalouf et al., 2006; Moustris & Tzafestas, 2005; Rodriguez-Castano et al., 2000; Sanchez et al., 1997; Yang et al., 1998). Of course, boundaries often blend since various approaches are used simultaneously. Fuzzy logic path trackers have been used by several researchers (Abdessemed et al., 2004; Antonelli et al., 2007; Baltes & Otte, 1999; Cao & Hall, 1998; Deliparaschos et al., 2007; El Hajjaji & Bentalba, 2003; Jiangzhou et al., 1999; Lee et al., 2003; Liu & Lewis, 1994; Moustris & Tzafestas, 2011; 2005; Ollero et al., 1997; Raimondi & Ciancimino, 2008; Rodriguez-Castano et al., 2000; Sanchez et al., 1997) since fuzzy logic provides a more intuitive way for analysing and formulating the control actions, which bypasses most of the mathematical load needed to tackle such a highly nonlinear control problem. Furthermore, the fuzzy controller, which can be less complex in its implementation, is inherently robust to noise and parameter uncertainties.

The implementation of Fuzzy Logic Controllers (FLC) in software suffers 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 supporting fuzzy computation exist (Costa et al., 1997; Fortuna et al., 2003; Salapura, 2000). Software solutions running on these devices speed up fuzzy computations by at least one order of magnitude over standard processors, but are still not fast enough for some real-time applications. Thus, a dedicated hardware implementation must be used (Hung, 1995).

Due to the 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 System-on-a-Chip (SoC) realised 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.
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, re-programmability, and have a faster time-to-market since no fabrication (layout, masks, or other manufacturing steps) time is required, when compared to ASICs.
The use of FPGAs in robotic applications is noted in (Kongmunvattana & Chongstivatana, 1998; Leong & Tsoi, 2005; Li et al., 2003; Reynolds et al., 2001). A review of the application of FPGA's in robotic systems is provided be Leong and Tsoi in (Leong & Tsoi, 2005). A notable case study is the use of FPGA's in the Mars Pathfinder, Mars Surveyor '98, and Mars Surveyor '01 Lander crafts, analysed in (Reynolds et al., 2001).

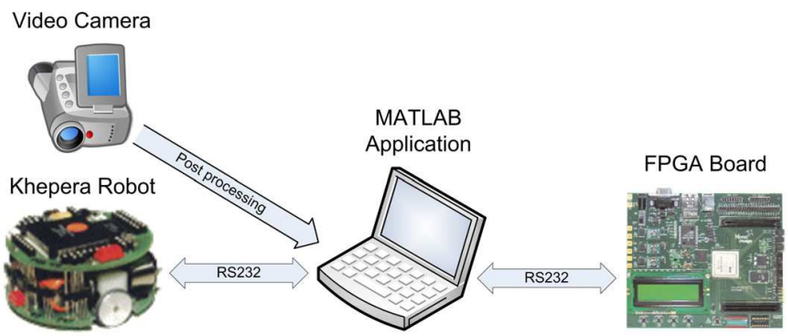



Fig. 1. Overview of the general system.

In this chapter we analyse a SoC implementation for the non-holonomic robot path tracking task using a fuzzy logic controller, along with a non-linear feedback-equivalence transformation which reduces path tracking to straight line tracking. The major components of the SoC are a parametrized Digital Fuzzy Logic Controller (DFLC) soft IP core Deliparaschos et al. (2006) Deliparaschos & Tzafestas (2006), implementing the fuzzy tracking algorithm, and Xilinx's Microblaze soft processor core as the top level flow controller. The system was tied to a differential drive robot and experiments were performed in order to asses the efficacy and performance of the overall control scheme. This was facilitated using an image analysis algorithm, presented in the following sections, which calculated the robot's position from a video stream captured using an overhead camera. The analysis was made off-line. The overall system setup can be seen in Fig 1.

## 2. Kinematics & odometry of the Khepera II robot

The mobile robot used in this work, is the Khepera II differential drive robot, described by the equations,

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \frac{r}{2}\cos\theta \\ \frac{r}{2}\sin\theta \\ \frac{r}{L} \end{bmatrix} u_r + \begin{bmatrix} \frac{r}{2}\cos\theta \\ \frac{r}{2}\sin\theta \\ -\frac{r}{L} \end{bmatrix} u_l \tag{4}$$

Here, $x, y$ are the coordinates of the middle point of the axis, $L$ is the axis length (the distance between the two wheels), $r$ is the wheel radius and $u_l, u_r$ the angular wheel velocities. A diagram of the model is seen in Fig.(2). Equations 4 can be transformed to a model more akin to the unicycle by first noting that the linear velocity of a point on the wheel's circumference is $v = r\omega$ ($\omega_i \triangleq u_i$). It can be easily shown that the linear velocity of the wheel's center equals the velocity of its circumference. Thus, denoting the centres' velocities as $v_r, v_l$, then,

$$\begin{aligned} v_r &= ru_r \\ v_l &= ru_l \end{aligned} \tag{5}$$

and substituting them into Eq.4, the system takes the form,

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta/2 \\ \sin\theta/2 \\ 1/L \end{bmatrix} v_l + \begin{bmatrix} \cos\theta/2 \\ \sin\theta/2 \\ -1/L \end{bmatrix} v_r \tag{6}$$

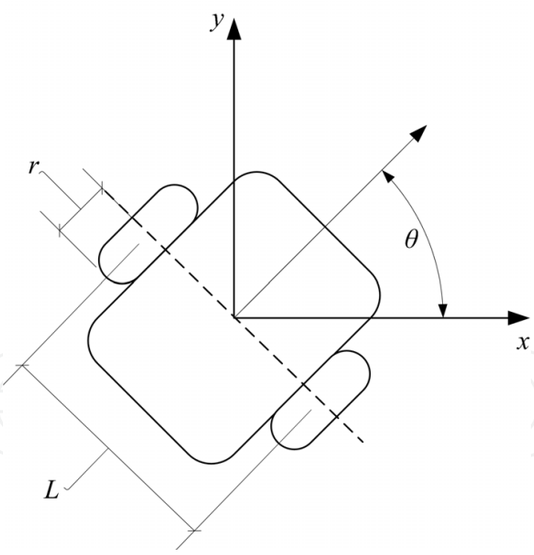



Fig. 2. Depiction of the generalized coordinated for the Differential Drive model

If we further apply a new input transformation,

$$\begin{aligned} u_s &= \frac{v_r + v_l}{2} \\ u_\theta &= \frac{v_r - v_l}{2} \end{aligned} \tag{7}$$

we get the familiar unicycle model, i.e.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta \\ \sin\theta \\ 0 \end{bmatrix} u_s + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} u_\theta \tag{8}$$

For the physical interpretation of the inputs $u_s, u_\theta$, consider that the robot performs a turn of radius R with respect to the axis' middle point (point *K*), centred at the point O (Fig 3). The point O is called the *Instantaneous Centre of Rotation* or ICR. If $\omega$ is the robot's angular velocity (actually the angular velocity of *K*), then its linear velocity is $v_s = \omega R$. It also holds that,

$$\begin{aligned} v_r &= \omega(R + L/2) \\ v_l &= \omega(R - L/2) \end{aligned} \tag{9}$$

By adding (9) together and solving for the linear velocity $v_s$, we get,

$$v_s = \frac{v_r + v_l}{2} \tag{10}$$

Subtracting (9) we come up with the robot's angular velocity $\omega$,

$$v_\theta = \frac{v_r - v_l}{L} = \omega \tag{11}$$

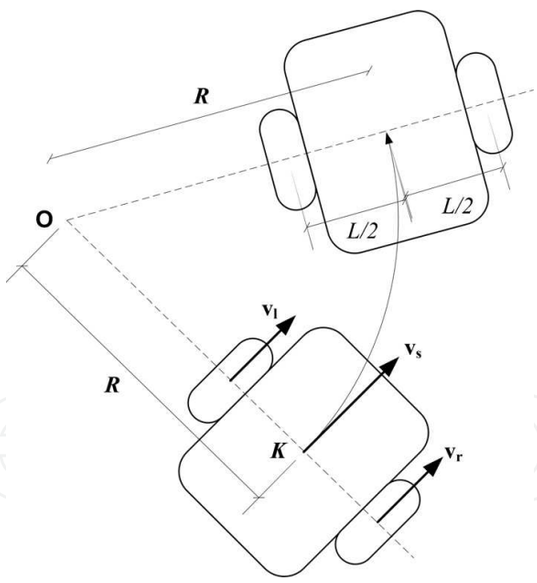



Fig. 3. Depiction of the velocities of the Differential Drive model

Observe that $v_s, v_\theta$ are actually $u_s, u_\theta$, thus the new input variables in Eq.7 are actually the robot's linear and angular velocities. What we have proved so far is that the unicycle (or the Dubins Car, which is the unicycle with a constant speed) is related to the Differential Drive by an input transformation, hence they are equivalent. This means that the Differential Drive can *emulate* these models. Consequently, we can develop a controller for either system and apply it to the others by using this transformation (this is feasible if the input transformation is actually *bijective*. If it is not, then the controller can by ported to one direction i.e. from

model A to model B, and not the other way around). As mentioned earlier, the Dubins Car is a simplified model where the velocity is considered constant. Furthermore, the path following problem involves the tracking of a purely geometric curve, where indeed the velocity is irrelevant. Hence, the model (8) can be transformed to a more suitable control form by using the curvature $\kappa \triangleq u_\kappa$. The curvature is related to the angular and linear velocities by the well known formula $\omega = \kappa v$, or, using our nomenclature,

$$u_\theta = u_\kappa u_s \tag{12}$$

Since the linear velocity $u_s = v$ is constant, by applying this transformation to the system (8), we get,

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ v \end{bmatrix} u_\kappa \tag{13}$$

This model is of control-affine form, with a non-vanishing drift term, where the only input is the curvature $u_\kappa$. By controlling the curvature in this model, we expect to control the actual system i.e. the Khepera robot, which is a differential drive and has two inputs. Thus, starting with the curvature, in order to calculate the true input vector to the robot, we need a second equation. This is of course the velocity equation $u_s = v$, which is considered known. By combining (10), (11), (12), the wheel velocities are calculated as,

$$\begin{aligned} v_r &= v(1 + u_\kappa L/2) \\ v_l &= v(1 - u_\kappa L/2) \end{aligned} \tag{14}$$

Equation 14 produces the linear wheel velocities of the robot, given its linear velocity and curvature. Since the linear velocity is constant, the only input is the curvature which is output by the fuzzy controller implemented on the FPGA.
In order to calculate its position, MATLAB queries the robot about its wheel encoder readings every 12.5 msec. The robot returns the 32bit encoder position, expressed in pulse units, with each unit corresponding to 0.08 mm. Consequently, by multiplying the units with 0.08 we get the *total length* each wheel has travelled since the beginning of the experiment. Now let $S_R(t), S_L(t)$ be the travel length of the right and left wheels at time $t$, and $S_R(t-1), S_L(t-1)$ be the length at $t-1$. We assume that in the interval $\Delta t$ the robot moves with a constant curvature, and thus traces an arc. This translates to constant wheel velocities (Eq. 14). Then, using (11) we have,

$$\omega = \frac{\Delta\theta}{\Delta t} = \frac{v_r - v_l}{L} = \frac{\Delta S_R - \Delta S_L}{\Delta t L} \Leftrightarrow \Delta\theta = \frac{\Delta S_R - \Delta S_L}{L} \tag{15}$$

If the robot's initial heading $\theta_0$, with respect to the world frame, is known, then at time $t$ it holds that,

$$\theta(t) = \theta_0 + \sum_{\tau=0}^{t} \Delta\theta(t) = \theta_0 + \sum_{\tau=0}^{t} \frac{\Delta S_R(\tau) - \Delta S_L(\tau)}{L} \tag{16}$$

Using 10, the length travelled by the point $K$ in $\Delta t$ is found to be,

$$v_s = \frac{\Delta S}{\Delta t} = \frac{v_r + v_l}{2} \Leftrightarrow \Delta S = \frac{\Delta S_R + \Delta S_L}{2} \tag{17}$$

To calculate the robot's $x, y$ position, we must solve the kinematics for the interval $\Delta t$, setting the input constant. The solution can be easily shown to be,

$$\begin{aligned} \Delta x(t) &= 2\frac{\Delta S}{\Delta\theta}\sin(\frac{\Delta\theta}{2})\cos(\theta_{t-1}+\frac{\Delta\theta}{2}) \\ \Delta y(t) &= 2\frac{\Delta S}{\Delta\theta}\sin(\frac{\Delta\theta}{2})\sin(\theta_{t-1}+\frac{\Delta\theta}{2}) \end{aligned} \tag{18}$$

In the case that the robot moves in a straight line, hence $\Delta\theta = 0$, taking the limit of 18 gives the equations,

$$\begin{aligned} \Delta x(t) &= \Delta S\cos(\theta_{t-1}+\frac{\Delta\theta}{2}) \\ \Delta y(t) &= \Delta S\sin(\theta_{t-1}+\frac{\Delta\theta}{2}) \end{aligned} \tag{19}$$

To get the absolute coordinates of the robot at $t$, Eq.19 must be integrated, leading to the odometric equations,

$$\begin{aligned} x(t) &= x_0 + \sum_{\tau=0}^{t}\Delta x(\tau) \\ y(t) &= y_0 + \sum_{\tau=0}^{t}\Delta y(\tau) \end{aligned} \tag{20}$$

Using the previous formulas (Eq.16 and Eq.20) we have succeeded in reconstructing the robot's state vector, i.e. the states $(x, y, \theta)$. Note that *time* is absent from the odometry equations. This has been chosen deliberately since it reduces the estimation error significantly. To consider this further, suppose that we get the wheel velocities from the robot and use the odometry equations involving the time $\Delta t$. The use of the velocities in the formulas inserts two types of errors; the first is the estimation of the velocities themselves from the robot. In the time between two queries to the robot, which is 12.5 msec, the velocity cannot be computed with adequate precision; the second error derives from the calculation of the interval $\Delta t$, which is inserted into the equations. This interval is not constant since there is always a small computational overhead in the software in order to setup and issue the command, communication delays etc. Furthermore, the queries to the robot are implemented in MATLAB using a *timer object*. The timer period however, is not guaranteed and is affected by the processes running on the computer at each instant. Thus, $\Delta t$ can vary from its nominal value, something which was also seen in actual experiments and must be minimized.

## 3. Strip-Wise Affine Map

The Strip-Wise Affine Map (SWAM) is the first step towards constructing a feedback equivalence relation which transforms the robot's model to a suitable form, under specific requirements. The equivalence relation however, exerts the useful property of *form invariance* on the mobile robot equations. The SWAM is defined for a robot model *and* a reference path, being applied to tracking control. To begin with, consider a reference polygonal path in the original physical domain, i.e. the actual domain where the robot dwells. Denote this physical domain as $D_p$ (w-plane) and the transformed canonical domain as $D_c$ (z-plane). Then, the strip-wise affine map is a homeomorphism $\Phi : D_c \rightarrow D_p$ that sends the real line of the canonical domain to the reference polygonal path in the physical domain. The SWAM is a piecewise linear homeomorphism between the two spaces (Groff, 2003; Gupta & Wenger,

1997) . It acts by inducing a strip decomposition on the planes and then applying an affine map between them. Note that the map acts on the entire domains, not just on a bounded region.

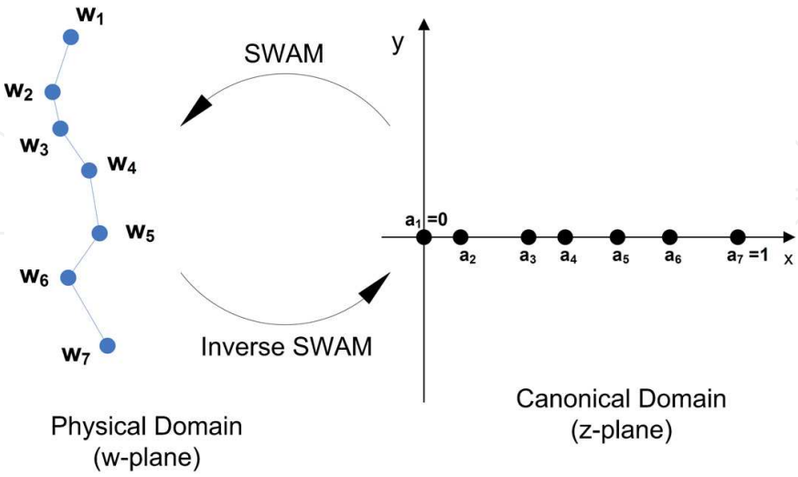



Fig. 4. Illustration of the Strip-Wise Affine Map

In more rigorous terms, let $A = \{w_1, w_2, \ldots, w_n\}, w_i \in \mathbb{C}$ be a polygonal chain on the *complex* physical domain. This represents the original reference path. Each vertex $w_i$ of the chain is projected to a point $a_i$ on the real axis in the canonical domain according to its normalized length,

$$a_i = \sum_{k=1}^{i} \frac{S_k}{S} \text{ , i} = 1,2,3,\ldots, \text{ n} \tag{21}$$

where $S_k = |w_k - w_{k-1}|, S = \sum_{k=1}^{n} S_k$ and $S_1 = 0$. The polygon edge from $w_{k-1}$ to $w_k$ is linearly projected onto $[a_{k-1}, a_k]$. The transformation of $[a_{k-1}, a_k]$ onto its respective polygon edge is done using the function

$$f_{k-1}(x) = w_{k-1} + S \cdot (x - a_{k-1}) \cdot e^{j \cdot \arg(w_k - w_{k-1})} \tag{22}$$

Each interval $[a_{k-1}, a_k]$ is transformed by a respective transformation $f_1, f_2, , f_{n-1}$. Now, consider the following rectangle pulse function on the canonical real axis,

$$\psi_k = \begin{cases} 1 \text{ , } x \in [a_k, a_{k+1}) \\ 0 \text{ , elsewhere} \end{cases} \tag{23}$$

The pulse is a complex function of $z = x + jy$ in the canonical domain. Each function $f_{k-1}$ is multiplied by the corresponding pulse and the products are summed to account for the general transformation that projects the interval [0,1) onto the polygonal chain,

$$f(x) = \sum_{k=1}^{n-1} f_k(x) \psi_k \tag{24}$$

Extension to the intervals (-∞,0) and [1,+ ∞) can be performed by appending an edge that begins from infinity ending at $w_1$, for the first case, and another edge starting from $w_n$ escaping

to infinity, on the second case. The edges have a direction of $\theta_{-\infty}$ and $\theta_{+\infty}$ respectively. The formulas that account for these branches are given by:

$$\begin{aligned} f_{-\infty}(x) &= (w_1 + S \cdot (x - a_1) \cdot e^{j \cdot \theta_{-\infty}})\psi_0, \\ f_{+\infty}(x) &= (w_n + S \cdot (x - a_n) \cdot e^{j \cdot \theta_{+\infty}})\psi_n \end{aligned} \tag{25}$$

Here $\psi_0$ is an open-left pulse with a single falling edge at x=$a_1$, and $\psi_n$ is the open-right pulse with a single rising edge at x=$a_n$. Combining the above and using the conventions $\theta_k = arg(w_{k+1} - w_k)$, $a_0 = -\infty$, $a_{n+1} = +\infty$, $w_0 \triangleq$ the point at infinity corresponding to $a_0$, $w_{n+1} \triangleq$ the point at infinity corresponding to $a_{n+1}$, $\theta_0 = arg(w_1 - w_0) = \theta_{-\infty}$, $\theta_n = arg(w_{n+1} - w_n) = \theta_{+\infty}$, the extended transformation takes the following form,

$$f(x) = \sum_{k=0}^{n} (w_k + S \cdot (x - a_k) \cdot e^{j \cdot \theta_k})\psi_k \tag{26}$$

where the functions $f_{-\infty}, f_{+\infty}$, correspond to k=0 and k=n respectively. In order to extend this transformation to the entire complex plane, let $z = x + jy$ be a complex variable in the canonical domain and consider the mapping,

$$\Phi(z) = y \cdot S \cdot e^{j \cdot \theta_s} + f(x) \tag{27}$$

where $\theta_s$ is the *shifting angle* in [-$\pi/2, \pi/2$].The complex variable $w = u + jv$ in the physical domain, is identified with the transformation $\Phi(z)$, i.e. $w = u + jv = \Phi(z)$. This transformation is the *direct strip-wise affine map* and produces a linear displacement of the polygon along the direction $\theta_s$. Each edge of the polygon produces an affine transformation that applies only in the "strip" that the edge sweeps as it is being translated. Thus, the transformation $\Phi(z)$ can be described as a "strip-wise affine" transformation. The invertibility of the map depends firstly on the geometry of the chain and secondly on the shifting angle. It can be shown (Moustris & Tzafestas, 2008) that necessary and sufficient conditions for the mapping to be invertible are that the chain must be a strictly monotone polygonal chain (Preparata & Supowit, 1981) and the shifting angle must not coincide with the angle of any of the chain's edges i.e. the chain must not be shifted along one of its edges. The inverse strip-wise affine map can be expressed in matrix form by treating $\Phi(z)$ as an $\mathbb{R}^2$ to $\mathbb{R}^2$ mapping since it cannot be solved analytically with respect to $z$. The inverse mapping equations can be calculated as,

$$\begin{bmatrix} x \\ y \end{bmatrix} = \mathrm{A}^{-1} \begin{bmatrix} u \\ v \end{bmatrix} - \begin{bmatrix} C/J - a_k \\ -D/J \end{bmatrix} \tag{28}$$

where $C = S \sum_{k=0}^{n} (w_k^R \sin\theta_s - w_k^I \cos\theta_s)\psi_k$, $D = S \sum_{k=0}^{n} (w_k^R \sin\theta_k - w_k^I \cos\theta_k)\psi_k$ and $J$ is the map Jacobian given by $J = S^2 \sum_{k=0}^{n} \sin(\theta_s - \theta_k)\psi_k$. $\mathrm{A}^{-1}$ is the inverse system matrix, i.e.

$$\mathrm{A}^{-1} = \begin{bmatrix} \sin\theta_s & -\cos\theta_s \\ -\sum_{k=0}^{n} \psi_k \sin\theta_k & \sum_{k=0}^{n} \psi_k \cos\theta_k \end{bmatrix} / S \sum_{k=0}^{n} \sin(\theta_s - \theta_k)\psi_k \tag{29}$$

Besides Eq.28, one also needs to know the activated pulse since the rectangle pulses are functions of the variable $x$, and thus (28) does not provide a complete solution to the inversion problem. If this information is provided, the sums in Eq.28 degenerate and the equation provides the inverse system. The activated pulse can be calculated algorithmically by doing

a point-in-strip test. Consider an axis orthogonal to the direction $\theta_s$ such that the projection of $w_1$ corresponds to 0. Furthermore let the projections of each $w_i$ onto that axis be denoted as $b_i$, and the projection of the current mapping point denoted as $b_c$. The projections of $w_i$ apparently partition the axis into consecutive line segments $[b_i b_{i+1}]$ which are into one-to-one correspondence with the edges of the polygonal chain. Then, in order to find the current pulse one needs to find the segment into which the point $b_c$ resides. This can be performed optimally by a binary search algorithm in $O(logn)$.

Since the SWAM transforms the robot's application space, it also affects its model's equations. Denoting all the state variables in the physical and canonical domains with a subscript of $p$ and $c$ respectively, then the $u$-$v$ plane (physical domain) is mapped to the $x$-$y$ plane (canonical domain), i.e. the state vector $q_p = [x_p, y_p, \theta_p]^T$ is transformed to $q'_p = [x_c, y_c, \theta_p]^T$ . The homeomorphism $\Phi$ defines an equivalence relation between the two states. Notice that the state $\theta_p$ remains unaffected. By introducing a new extended homeomorphism $\Psi$ that also maps the heading angle $\theta_p$, one can send the *canonical state-space* to the *physical state-space*, i.e. $q_p = \Psi(q_c)$. This transformation acts on all state variables and the new system state is $q_c = [x_c, y_c, \theta_c]^T$. The map is then defined by,

$$\begin{bmatrix} x_p \\ y_p \\ \theta_p \end{bmatrix} = \begin{bmatrix} y_c S \cos\theta_s + \mathrm{Re}(f(x_c)) \\ y_c S \sin\theta_s + \mathrm{Im}(f(x_c)) \\ \tan^{-1}\left(\frac{\sum_{\kappa=0}^{n} \sin\theta_\kappa \psi_\kappa + \sin\theta_s \tan\theta_c}{\sum_{\kappa=0}^{n} \cos\theta_\kappa \psi_\kappa + \cos\theta_s \tan\theta_c}\right) \end{bmatrix} = \Psi(q_c) \tag{30}$$

and the new system is,

$$\tilde{\Sigma}: \quad \begin{bmatrix} \dot{x}_c \\ \dot{y}_c \\ \dot{\theta}_c \end{bmatrix} = \begin{bmatrix} v_c \cos\theta_c \\ v_c \sin\theta_c \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ S^3 \gamma^3 J^{-1} v_c \end{bmatrix} \kappa_p \tag{31}$$

$J$ is the Jacobian of $\Phi$ and $\gamma = \sqrt{1 + \sin 2\theta_c \sum_{\kappa=0}^{n} \cos(\theta_s - \theta_\kappa)\psi_\kappa}$. The input $\kappa_p$ of the system remains unaffected. However, since it expresses the curvature of the physical system, it can also be transformed under $\Psi$. Thus by including the transformation of the input and extending the homeomorphism $\Psi$ to $\hat{\Psi} = (\Psi, \Omega)$ , where

$$\kappa_p = \Omega(\kappa_c, q_c) = S^{-3} J \gamma^{-3} \kappa_c$$

is the input map that sends the controls from the canonical input space to the controls in the physical input space, one gets the new extended system,

$$\Sigma_c: \quad \begin{bmatrix} \dot{x}_c \\ \dot{y}_c \\ \dot{\theta}_c \end{bmatrix} = \begin{bmatrix} v_c \cos\theta_c \\ v_c \sin\theta_c \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ v_c \end{bmatrix} \kappa_c \tag{32}$$

The systems $\Sigma_p$ and $\Sigma_c$ are feedback-equivalent (Gardner & Shadwick, 1987; 1990) since they are related by a state and input transformation. The input transformation is actually a feedback transformation of $\Sigma_c$ that feeds the states $q_c$ back to the input. A more careful look at Eq.(32) shows that it expresses the Dubins Car in the canonical domain, thus $\hat{\Psi}$ presents a kind of *form invariance* on the model.

Now, let $p_{ref}$ be a reference path in the physical domain and let $u_c(q_c, I_c, t)$ be a straight line tracking controller for $\Sigma_c$, where $I_c$ denotes the line segment $y_c = \{0/x_c \in [0,1]\}$ of the C-plane, i.e. the reference path in the canonical domain. This controller is transformed to a path tracking controller for $\Sigma_p$ under the equation,

$$u_p(q_p, p_{ref}, t) = \Omega(u_c(q_c, I_c, t), q_c) = \Omega(u_c(\Psi^{-1}(q_p), \Phi^{-1}(p_{ref}), t), \Psi^{-1}(q_p)) \quad (33)$$

However, since $\Sigma_c$ is the Dubins Car in the canonical domain, the straight line tracking controller $u_c(q_c, I_c, t)$ for $\Sigma_c$ is actually a straight line tracker for the Dubins Car, and in order to build a path tracker for the Dubins Car, one has but to build a straight line tracker and use Eq.(33) to promote it to a path tracker for strictly monotone polygonal chains. Furthermore, one could also use existing path trackers for the Dubins Car to track the straight line in the canonical domain. In this case, these controllers can be simplified since in general, straight line tracking is simpler than path tracking.

## 4. Fuzzy Logic Controller

The controller used in this work for the path following task, is based on a Fuzzy Logic Controller (FLC) developed by the authors, which has been deployed in previous experiments. The original Fuzzy Logic tracker is described in (Moustris & Tzafestas, 2005), and further modified in (Deliparaschos et al., 2007) in order to be implemented on a FPGA. Specifically the tracker is a zero-order Takagi-Sugeno FLC with the two inputs partitioned in nine triangular membership functions each, while the output is partitioned in five singletons (Fig. 5).

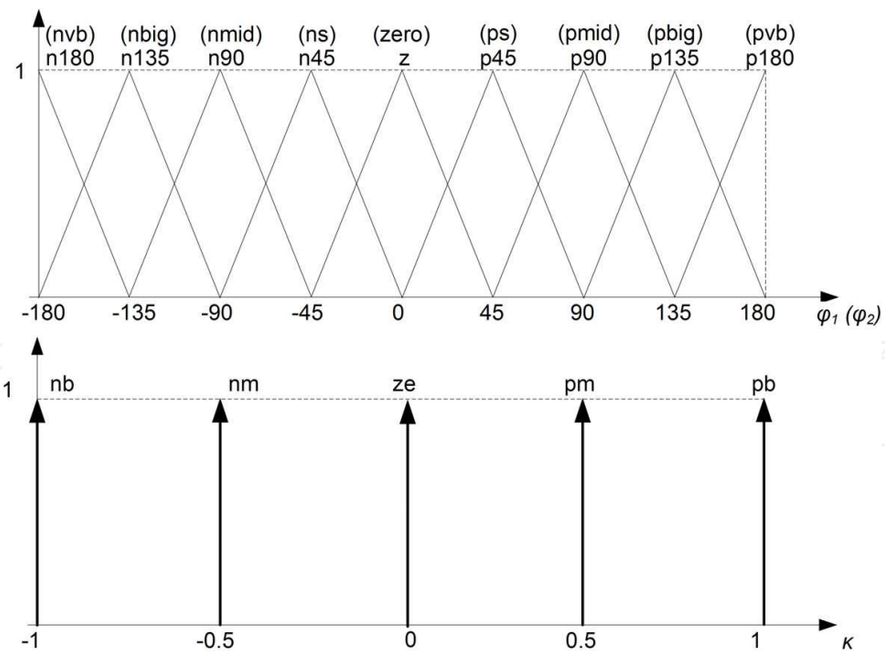



Fig. 5. Input and output membership functions of the fuzzy controller

The FL tracker uses two angles as inputs, and outputs the control input $u_\kappa$. It is assumed that the path is provided as a sequence of points on $\mathbb{R}^2$. The fuzzy rule base consists of 81 rules, which are presented in Table 1. The implication operator is the min operator.

| | pvb | pbig | pmid | ps | zero | ns | nmid | nbig | nvb |
|---|---|---|---|---|---|---|---|---|---|
| **p180** | pb | pb | pb | pm | ze | nm | nb | nb | pb |
| **p135** | pb | pb | pb | pb | pm | nm | nb | Pb | pb |
| **p90** | pb | pb | pb | pm | pm | pm | pb | pb | Pb |
| **p45** | pb | pb | pb | pm | pm | ze | nb | pb | pb |
| **z** | pb | pb | Pb | pm | ze | nm | nb | nb | nb |
| **n45** | nb | nb | pb | ze | nm | nb | nb | nb | nb |
| **n90** | nb | nb | nb | nm | nm | nb | nb | nb | nb |
| **n135** | nb | nb | pb | pm | nm | nb | nb | nb | nb |
| **n180** | pb | pb | pb | pm | zero | nm | nb | nb | pb |

Table 1. FLC Rule Base

In each control loop the closest path point is picked up and the two input angles are calculated. These angles are the angle $\phi_1$ of the closest point with respect to the current robot heading and the direction $\phi_2$ of the tangent of the path at that point, as depicted in Fig. 6a. Using the SWAM, we can move the tracking task to the canonical domain where the path is a straight line. In this case, the oriented straight line splits the plane into two half-planes, which present two general cases for the placement of the robot. Due to the symmetry of the cases only one will be analyzed. Consider that the robot resides in the positive half-plane (Fig. 6b) and that the distance from the closest path point P is D.

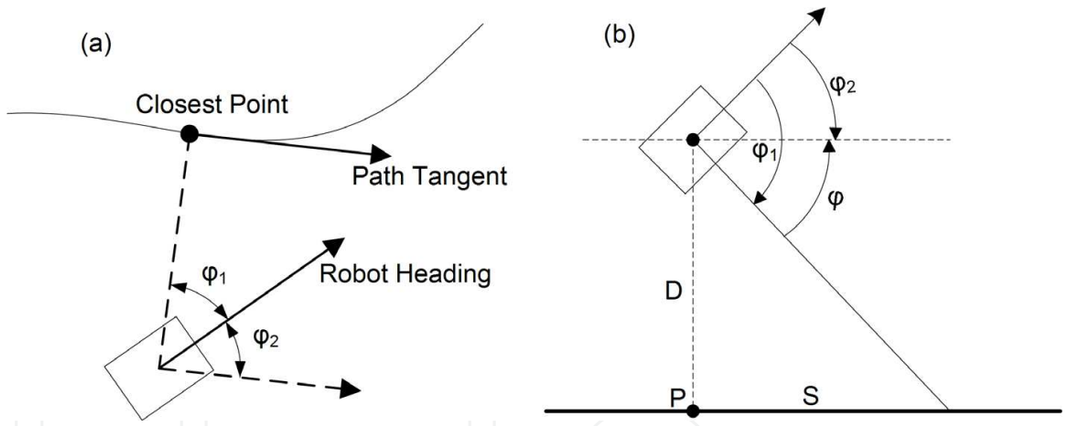



Fig. 6. Illustration of the controller inputs for path tracking in the general case (a) and in the straight line case (b).

Furthermore, one can consider that the robot tracks not the closest point P but the one found distance S ahead. The "sightline" of the robot to that point forms an angle $\phi$ with the path. In this situation, by varying the angle $\phi_2$ one can discern four cases for the relation between the angles $\phi$, $\phi_1$ and $\phi_2$ with the three of them being the same, namely,

$$\begin{aligned} \varphi_1 - \varphi_2 &= -\varphi \ , \varphi_2 \in [-\pi + \varphi, 0] \cup [0, \varphi] \cup [\varphi, \pi] \\ \varphi_1 - \varphi_2 &= 2\pi - \varphi \ , \varphi_2 \in [-\pi, -\pi + \varphi] \end{aligned} \tag{34}$$

When the robot resides in the positive half-plane, the angle $\phi$ is also positive. On the contrary, when it is in the negative half-plane, the angle changes sign although Eqs.(34) remain the same. With respect to the point being tracked, we discern two cases; either fixing the sightline, i.e. fixing the angle $\phi$, or fixing the look-ahead distance S, i.e. tracking the point that is distance

S ahead of the one nearest to the robot (point P). Of course, the nearest point P can be easily found since its coordinates are $(x_c, 0)$, where $x_c$ is the x-axis coordinate of the robot in the canonical space. In the case of a constant S, the angle $\phi$ varies from $[-\pi/2, \pi/2]$ and the tuple $(\phi_1, \phi_2)$ is constrained in a strip. This can also lead to a rule reduction in the FLC rule base since some rules are never activated ((Moustris & Tzafestas, 2011)). For the control of the robot, the FLC and the SWAM were calculated on-line using a dedicated FPGA SoC, as described in the next section.

## 5. DFLC & SoC architecture

This section discusses the System on Chip (SoC) implemented on an FPGA chip for the robot path tracking task using fuzzy logic. The SoC design was implemented on the Spartan-3 MB development kit (DS-KIT-3SMB1500) by Digilent Inc. 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 SoC is shown in Fig.7 and 8 respectively.

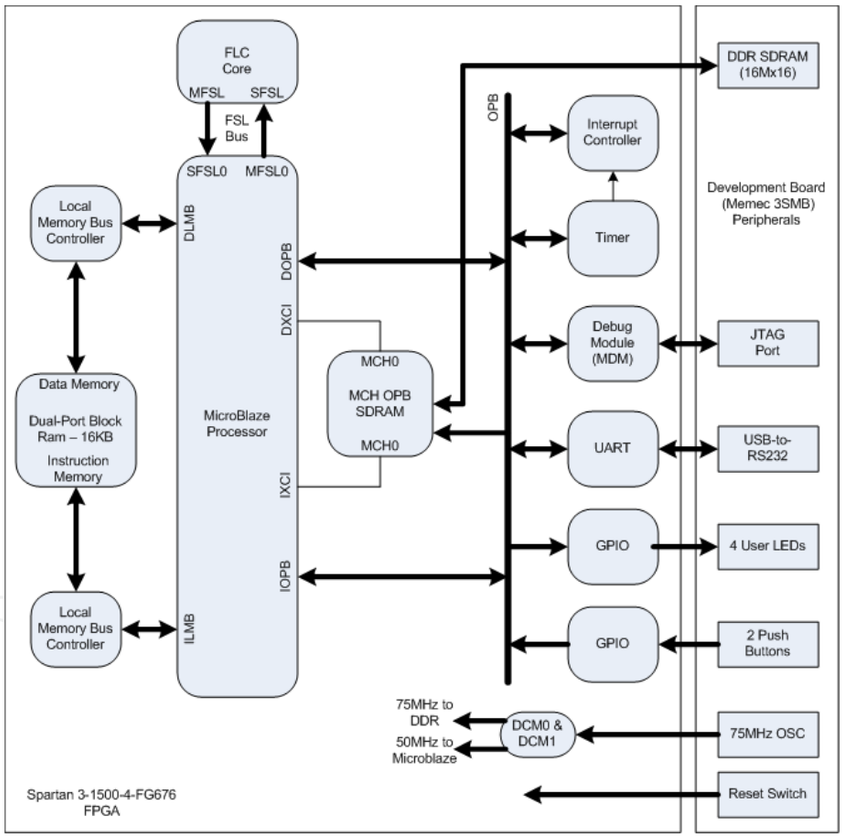



Fig. 7. Overview of the SoC's hardware architecture

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

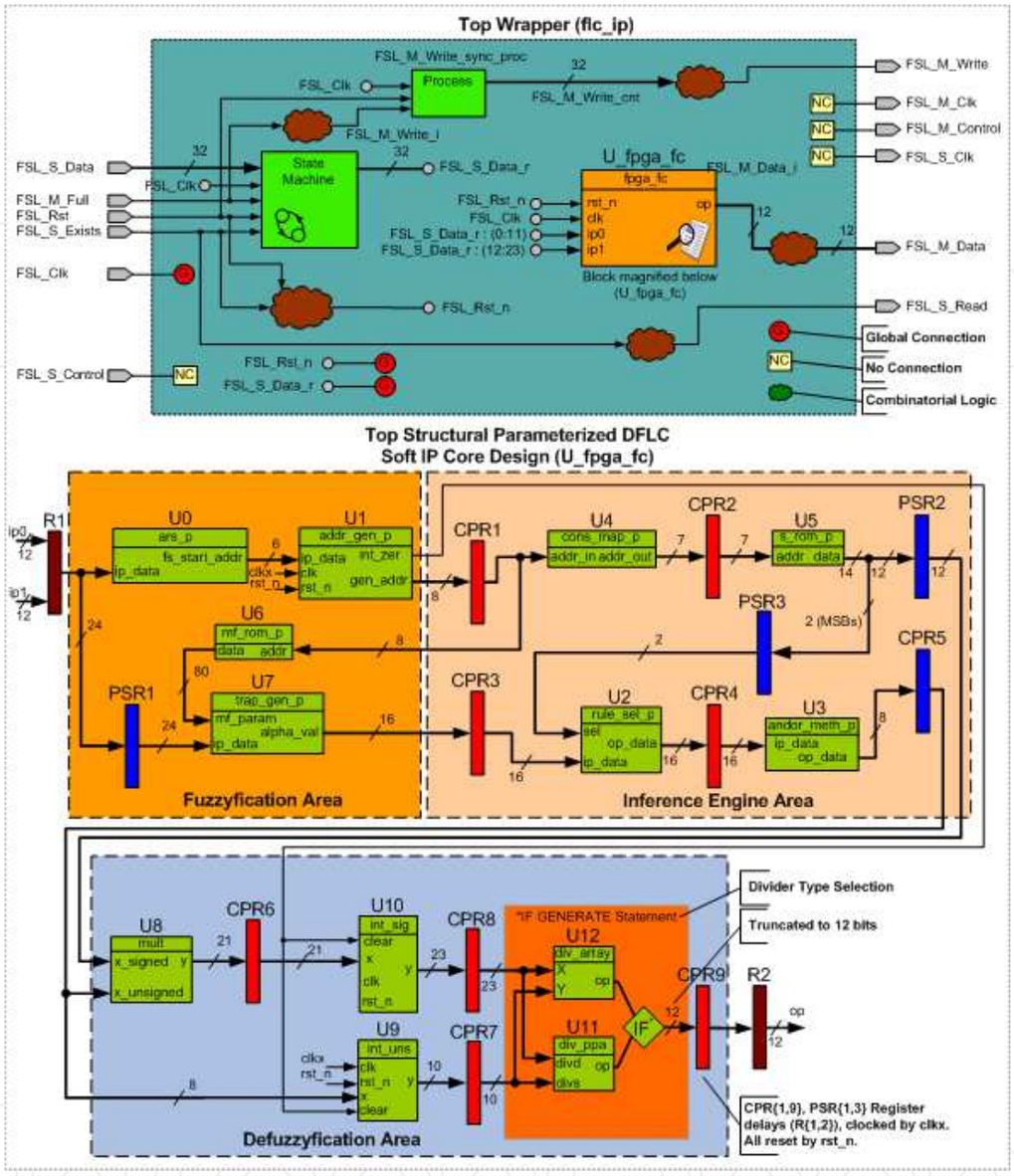



Fig. 8. Architecture of the DFLC IP core

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 (see Fig.9).

Today's FPGAs contain hundreds of powerful DSP slices with up to 4.7 Tera-MACS throughput; 2 million logic cells with clock speeds of up to 600MHz, and up to 2.4 Tera-bps high-speed on-chip bandwidth capable to outperform DSP and RISC processors by a factor of 100 to 1,000. 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 main unit of the SoC is a parametrized Digital Fuzzy Logic Controller (DFLC) soft IP core Deliparaschos et al. (2006)Deliparaschos & Tzafestas (2006) that implements the fuzzy

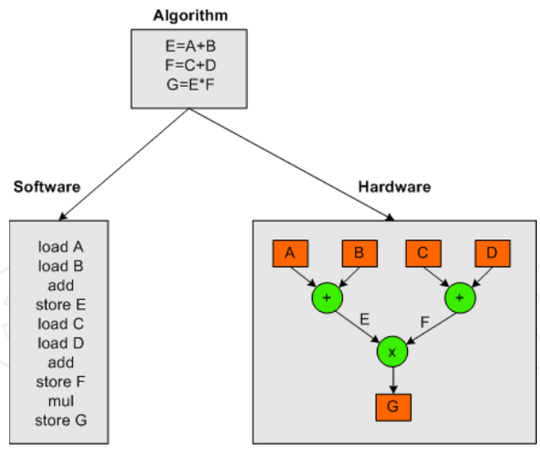



Fig. 9. Illustration of HW vs SW computation process

tracking algorithm, and a Xilinx's Microblaze soft processor core which acts as the top level flow controller. The FPGA board hosting the SoC controls the Kephera robot, used in the experiments of the tracking scheme.

In our application the DFLC facilitates scaling and can be configured for different number of inputs and outputs, number of triangular or trapezoidal fuzzy sets per input, number of singletons per output, antecedent method (t-norm, s-norm), divider type, and number of pipeline registers for the various components in the model. This parametrization enabled the creation of a generic DFLC 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 78.2960ns), since it processes one active rule per clock cycle. In its present form the SoC design achieves a core frequency speed of 51.1 MHz. To achieve this timing result, the latency of the chip architecture involves 9 pipeline stages each one requiring 19.574ns. The featured DFLC 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 FPGA SoC implements the autonomous control logic of the Kephera II robot. It receives odometry information from the robot and issues steering commands output by the FL tracker. The encoding and decoding of the information packets (i.e., encoding of steering control into data frames) is handled by the MATLAB application. Therefore the MATLAB application implements the actual framer/deframer for the I/O communication with the Kephera robot and downloads the tracking path to the SoC. The top-level program that supervises these tasks, treats synchronization and timing requirements, is written in C and executes in the Microblaze soft processor core. The SWAM algorithm is also implemented on the FPGA, in order to reduce the computation time.

The MATLAB application displays information about the robot's pose and speed, 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 frames. Another important function of the MATLAB application is to provide a path for the robot to track. The current work deals only

| Parameters (VHDL generics) | Value | Generic Description |
|---|---|---|
| ip_no | 2 | Number of inputs |
| ip_sz | 12 | Input bus resolution (bits) |
| op_no | 1 | Number of outputs |
| op_sz | 12 | Output bus resolution (bits) |
| MF_ip_no | 9 | Number of input MFs (same for all inputs) |
| dy_ip | 8 | Input MFs degree of truth resolution (bits) |
| MF_op_no | 5 | Number of output MFs (singletons) |
| MF_op_sz | 12 | Output MFs resolution (bits) |
| sel_op | 0 | Antecedent method connection: 0: min, 1: prod, 2: max, 3: probor |
| div_type | 1 | Divider Model: 0: restoring array, 1: LUT reciprocal approx. |
| **PSR** | | **Signal Path Route** |
| psr1_no | 1 | ip_set→psr1_no→trap_gen_p |
| psr2_no | 4 | s_rom→psr2_no→mult |
| psr3_no | 1 | s_rom→psr3_no→rul_sel_p |
| psr4_no | 1 | cpr5→psr→int_uns |
| **CPR** | | **Component (Entity) Name** |
| cpr1_no | 1 | addr_gen_p |
| cpr2_no | 1 | cons_map_p |
| cpr3_no | 3 | trap_gen_p |
| cpr4_no | 0 | rule_sel_p |
| cpr5_no | 2 | minmax_p |
| cpr6_no | 1 | mult |
| cpr7_no | 0 | int_uns |
| cpr8_no | 0 | int_sig |
| cpr9_no | 2 | div_array |

Table 2. DFLC soft IP core chosen parameters

with the path tracking task and not path planning. To compensate for this, the path is drawn in MATLAB, encoded properly and downloaded to the SoC. Then, the SoC begins the tracking control.

The Microblaze soft processor core is licensed as part of the Xilinx Embedded Development Kit (EDK) and 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 (Patterson & Hennessy, 1997)(Sailer et al., 1996). It features a 3-stage pipeline with most instruction completing in a single cycle. Both the instruction and data words are 32 bits. The core alone can obtain a speed of up to 100MHz on the Spartan 3 FPGA family. The Microblaze processor can connect to the OPB bus for access to a wide range of different modules, it can communicate via the LMB bus for a fast access to local memory, normally block RAM (BRAM) inside the FPGA.

Moreover, the Fast Simplex Link (FSL) offers the ability to connect user soft core IP's 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 extending the processor's execution unit with custom soft core co-processors. In this work the DFLP IP core is playing the role of such

a co-processor and is connected to the Microblaze via the FSL bus. The architecture of the present SoC consists mainly of the DFLP that communicates with the Microblaze Processor through the Fast Simplex Bus (FSL), the utilized block RAMs (BRAM) through the LMB bus, and other peripherals such as the general purpose input/output ports (GPIO), and UART modules via the OPB bus. Here, the DFLP incorporates the fuzzy tracking algorithm, whereas the Microblaze processor mainly executes the C code for the flow control.
The parametrized zero-order TSK type Fuzzy Logic Controller (FLC) core exchanges data with the MicroBlaze processor via the FSL bus. The scope of the FLC core is to serve as high-speed fuzzy inference co-processor to the Microblaze. The DFLC core was implemented with the following parameters (see Table3).

| Property | Value |
|---|---|
| Inputs | 2 |
| Input resolution | 12 |
| Outputs | 1 |
| Output resolution | 12 bit |
| Antecedent Membership Functions (MF's) | 9 Triangular MF's |
| Degree of Truth resolution | 8 bit |
| Consequent MF's | 5 Singletons |
| MF resolution | 8 bit |
| Number of fuzzy inference rules | 81 |
| Rule activation method | MIN |
| Aggregation method | SUM |
| Implication method | PROD |
| MF overlapping degree | 2 |
| Defuzzification method | Weighted average |

Table 3. DFLC core characteristics

Besides these two main cores and buses, the design consists of 16 KB local memory, 32 MB DDR, timer, interrupt controller, a UART, a debug peripheral (MDM) and a couple of General Purpose Inputs/Outputs (GPIOs). A Multi-CHannel (MCH) On-chip Peripheral Bus (OPB) Double Data Rate (DDR) Synchronous DRAM (SDRAM) memory controller (MCH OPB DDR with support for asynchronous DDR clock) is used in this design. This allows the MicroBlaze system to run at a lower speed of 51 MHz, which is more reasonable for Spartan-3, while the DDR is running at 75 MHz, which is the minimum required frequency for the Micron DDR chip. The on-chip Digital Clock Manager (DCM) is used to create the various clock frequencies and phases required to make this system work, all based on the 75 MHz oscillator on the 3SMB board. The FLC core runs at the same speed as the OPB and MicroBlaze, which is 51 MHz. Based on the place and route report, the design occupies 4174 out of 13312 slices of the Xilinx Spartan 3 FPGA (XC3S1500-4FG676).

## 6. Hardware/software co-design

On the beginning of the co-design process one starts with an architecture independent description of the system functionality. Since the description of the system functionality is independent of the HW and SW, several system modelling representations may be used, such as finite state machines (FSMs) for example. The modelled system can then be described by means of a high level language, which is next compiled into an internal representation

such as a data control flow description. This description which serves as a unified system representation allows to perform HW/SW functional partitioning. After the completion of the partitioning, the HW and SW blocks are synthesized and evaluation is then performed. If the evaluation does not meet the required objectives, another HW/SW partition is generated and evaluated (Rozenblit & Buchenrieder, 1996)(Kumar, 1995).
A general HW/SW co-design schema followed in this SoC implementation is illustrated in Fig.10).

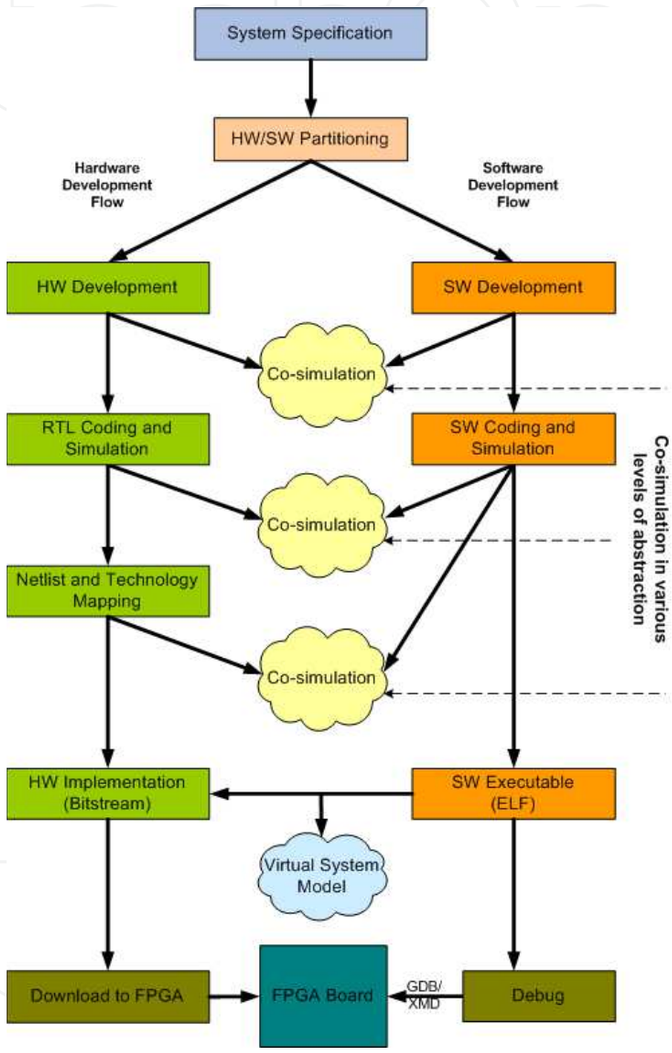



Fig. 10. HW/SW Co-design Flow

The DFLC core implementation follows a sequential design manner (see Fig.11) (Navabi, 1998). The starting point of the design process was the functional modelling of the fuzzy controller in a high level description (i.e., MATLAB/Simulink). This serves a two purpose role, first to evaluate the model and second to generate a set of test vectors for RTL and timing verification. The model was coded in register transfer level (RTL) with the use of hardware description language VHDL. Extensive use of VHDL generic and generate statements was used through out the coding of the different blocks, in order to achieve a parameterized DFLC

core. The DFLC core is scalable in terms of the number of inputs/bus resolution, number of input/output fuzzy sets per input and membership resolution. More specifically A VHDL package stores the above generic parameters together with the number of necessary pipeline stages for each block. An RTL simulation was performed to ensure the correct functionality of the fuzzy controller. The DFLC core was independently synthesized with Synopsys Synplify logic synthesis tool (as it produced better synthesis results and meet timing constraints), whereas the rest of the SoC cores were synthesised with Xilinx synthesis tool XST. The Xilinx EDK studio was used for the integration flow of different SoC cores (i.e., DFLC, Microblaze, etc) and Xilinx ISE tool for the placement and routing of the SoC on the FPGA. More analytically, the place and route tool accepts the input netlist file (.edf), previously created by the synthesis tool and goes through the following steps. First, the translation program translates the input netlist together with the design constraints to a database file. After the translation program has run successfully, the logical design is mapped to the Xilinx FPGA device. Lastly, the the mapped design is placed and routed onto the chosen FPGA family and a device configuration file (bitstream) is created. Xilinx's SDK used for C programming and debugging the SoC's Microblaze soft processor. RTL and timing simulation to verify the correct functionality was handled with the use of Mentor's Modelsim simulator.

## 7. FPGA design and performance evaluation

The Component Pipeline Registers (CPR) blocks in Fig.8 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 Deliparaschos & Tzafestas (2006).

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.8 while the chosen (generic) parameters (VHDL package definition file) for the parameterized DFLC IP (U_fpga_fc) and its characteristics are summarized in Table 2 and Table 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 is 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 SoC design (including the DFLC) occupies 4,494 (16%) LUTs, 15 Block Multipliers (MULT18X18s), and 18 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 DFLC core itself occupies 1303 or 4% LUTs, 8 Block Multipliers, 12 64x1 ROMs (ROM64X1) and 54 256x1 ROMs (ROM256X1). The SoC achieves a minimum clock operating period of 19.574ns or a maximum frequency of $\sim$51.1 MHz respectively (the DFLC with the chosen parameters reports a frequency of 85MHz when implemented alone).

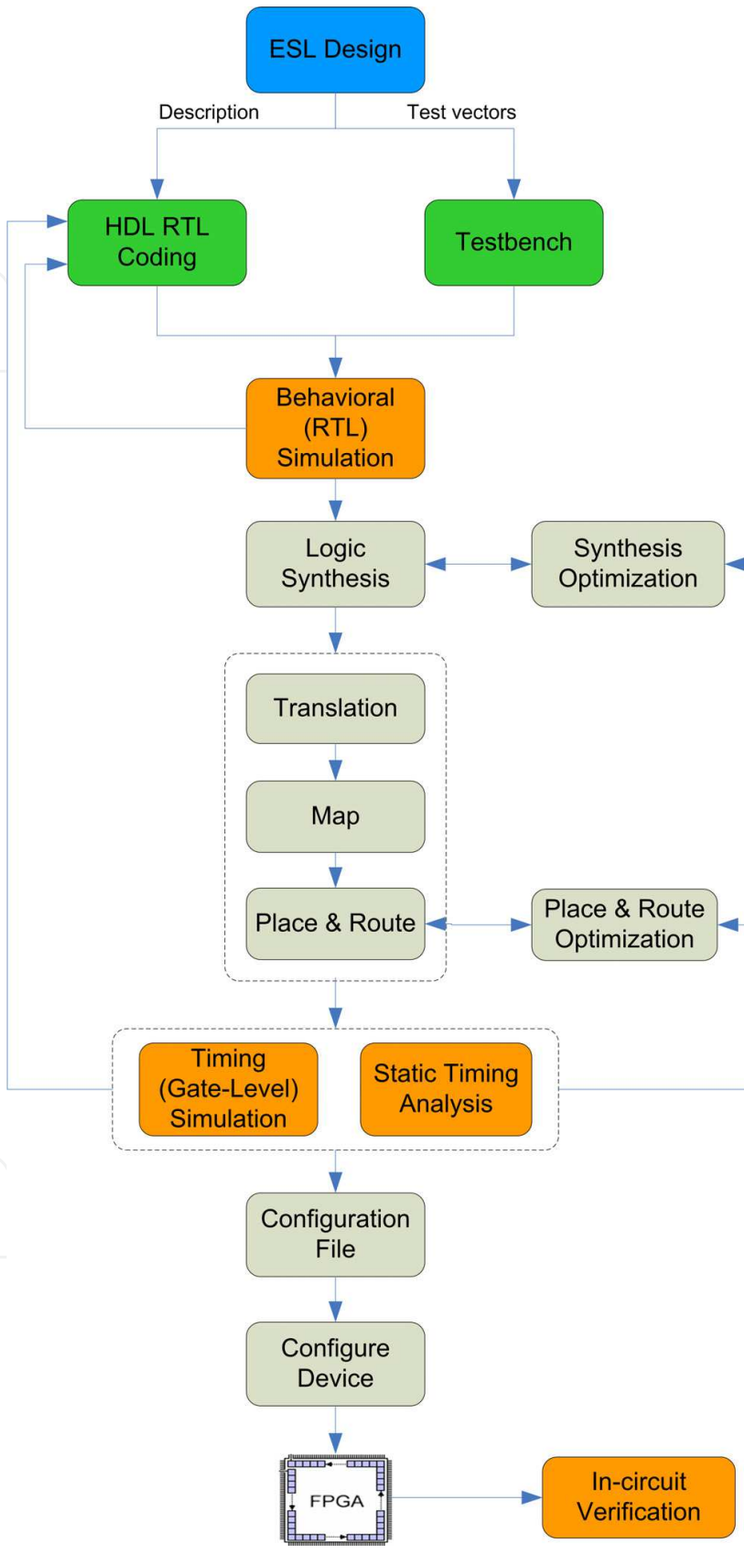



Fig. 11. HW/SW Hardware design flow

| **Logic Utilization** | | |
|---|---|---|
| Number of Slice Flip Flops | 3,288 out of 26,624 | 12% |
| Number of 4 input LUTs | 4,494 out of 26,624 | 16% |
| **Logic Distribution** | | |
| Number of occupied Slices | 4,174 out of 13,312 | 31% |
| Number of Slices containing only related logic | 4,174 out of 4,174 | 100% |
| Number of Slices containing unrelated logic | 0 out of 4,174 | 0% |
| Total Number of 4 input LUTs | 5,893 out of 26,624 | 22% |
| Number used as logic | 4,494 | |
| Number used as a route-thru | 165 | |
| Number used for Dual Port RAMs | 432 | |
| (Two LUTs used per Dual Port RAM) | | |
| Number used as 16x1 ROMs | 432 | |
| Number used as Shift registers | 370 | |
| Number of bonded IOBs | 62 out of 487 | 12% |
| IOB Flip Flops | 94 | |
| IOB Dual-Data Rate Flops | 23 | |
| Number of Block RAMs | 18 out of 32 | 56% |
| Number of MULT18X18s | 15 out of 32 | 46% |
| Number of GCLKs | 6 out of 8 | 75% |
| Number of DCMs | 2 out of 4 | 50% |
| Number of BSCANs | 1 out of 1 | 100% |
| Total equivalent gate count for design | 1,394,323 | |
| Additional JTAG gate count for IOBs | 2,976 | |

Table 4. SoC design summary

## 8. Experimental results

The experiments consist of tracking predefined paths and analysing the displacement error. The paths are drawn by hand in the MATLAB application and downloaded to the FPGA. Then the control algorithm running on the board, calculates the steering command (curvature), relays it to MATLAB, which in turn passes it to the robot. Conversely, the MATLAB application receives odometric data from the robot which are then relayed to the FPGA. Essentially, the MATLAB application acts as an intermediary between the board and the robot, transforming commands and data to a suitable form for each party. Note also that the actual odometry is being performed by MATLAB (estimation of the robot's pose $(x, y, \theta)$ using the data from the robot's encoders). A key detail in the above process is that odometry provides an estimation of the actual pose. Thus in order to analyse the efficacy of the tracking scheme, we need to know the actual pose of the robot. Position detection of the robot is achieved using a camera hanging above the robot's activity terrain and utilizing a video tracking algorithm which extracts the robot's trajectory in post-processing. This algorithm tracks a red LED placed at the center of the robot.

The video tracking algorithm uses the high contrast of the LED with its surrounding space. Specifically, each video frame is transformed from the $RGB$ color space to the generalized $rgb$

space. This space expresses the percentage of each color at each pixel, i.e.,

$$\begin{aligned} r &= \frac{R+G+B}{R} \\ g &= \frac{R+G+B}{G} \\ b &= \frac{R+G+B}{B} \end{aligned} \tag{35}$$

It is the hyper-space of the so-called $rg$ chromaticity space, which consists of the first two equations. Following the transformation, in order to enhance the contrast, each image pixel is raised to the 3rd power,

$$(r', g', b')_{(u,v)} = (r^3, g^3, b^3)_{(u,v)} \tag{36}$$

The new image is then re-transformed according to the $rgb$ transform, essentially computing the color percentage of the percentage. Then, we apply a thresholding on the $r$ channel, producing a binary image. The threshold was empirically set to "0.6". This procedure produces the "patch" of pixels corresponding to the red LED. The next step is, of course, to calculate a single pixel value from this patch, and thus get the robot's position. To this end, we calculate the median $(row, col)$ value of the patch's pixels. This algorithm is applied to the first video frame, and is rather slow since the image dimensions are large (1280×720 pixels). In order to speed up the process, the algorithm processes an image region of interest in each consecutive frame. This ROI is a 17×17 pixel square, centred at the point extracted from the previous frame. The square dimensions are appropriate since the robot is not expected to have moved far between frames. The precision of the algorithm is about ±2 pixels, translating to 2.4 mm.

Previous to all runs, the camera was calibrated using the Camera Calibration Toolbox by J.Y. Bouguet, extracting the camera's intrinsic and extrinsic parameters. Two runs were performed; one tracking a straight line and one tracking a curved path (snapshots of the two videos are seen in Fig. 12). For the first run, the reference, odometry and camera paths are presented in Fig. 13(UP). The minimum distance versus the path length of the odometry and the camera paths are shown in Fig.13(DOWN). Likewise, for the second experiment the results are presented in Fig14.

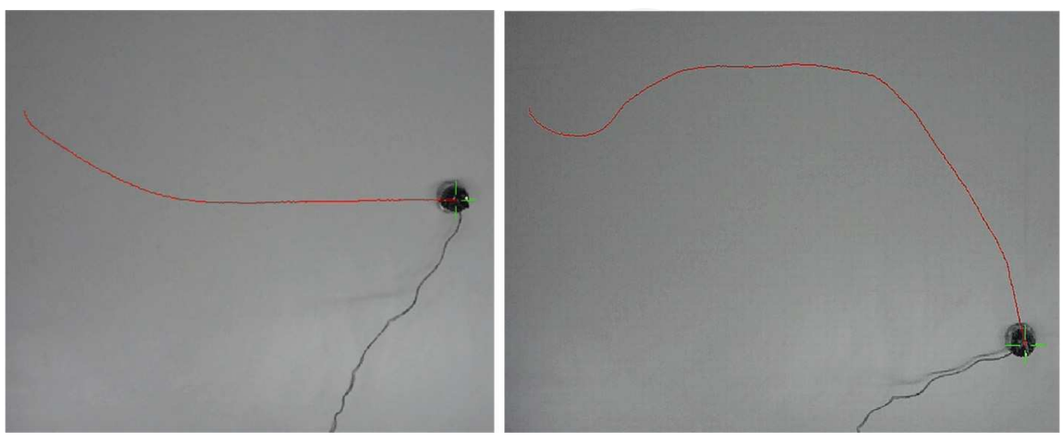


Fig. 12. Snapshots of the first (LEFT) and the second (RIGHT) experiments. The red line is the robot's path calculated off-line from the video camera.

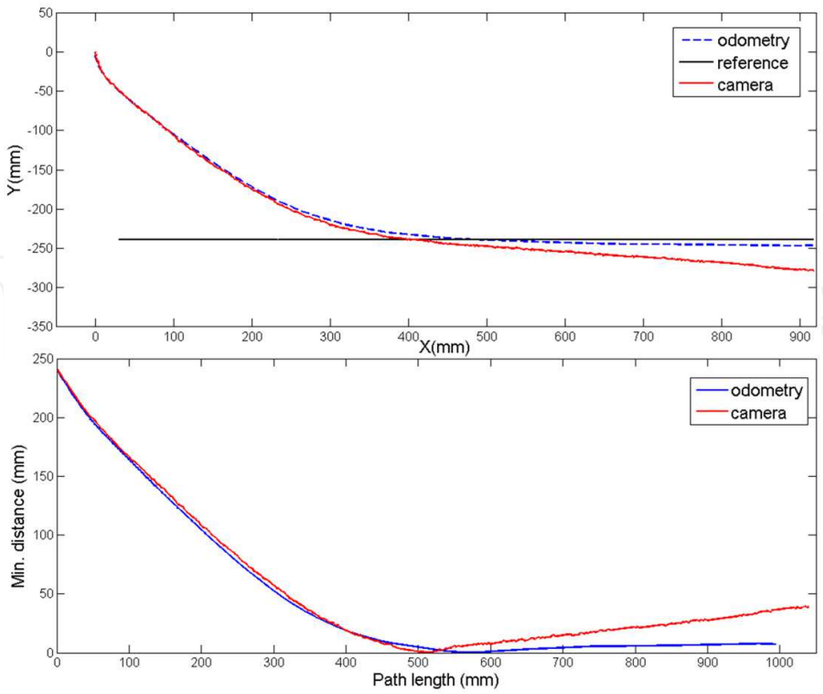



Fig. 13. (UP) The odometry (blue), camera (red) and reference (dashed black) paths for the first experiment. (DOWN) Minimum distance of the odometry and camera paths to the reference path versus path length experiment.

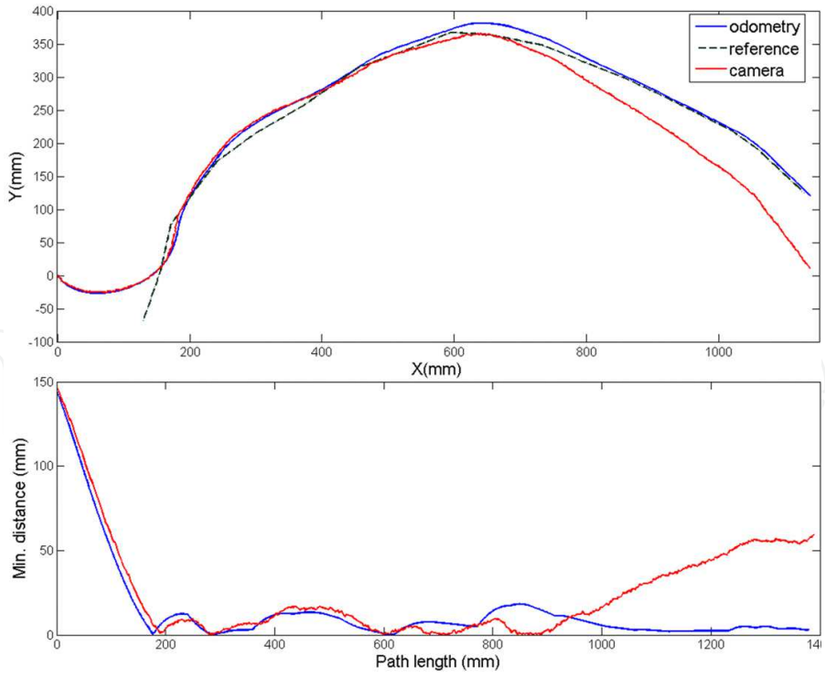



Fig. 14. (UP) The odometry (blue), camera (red) and reference (dashed black) paths for the second experiment. (DOWN) Minimum distance of the odometry and camera paths to the reference path versus path length.

As one can see, the performance of the tracking scheme is satisfactory maintaining the minimum distance to the reference path at about 50mm in the worst case. However, by taking a closer look at Figures 13 and 14, it is clear that the performance degradation is attributed not to the algorithm *per se* but to the odometry. The error accumulation of odometric data forces the robot to diverge from the actual path. But the actual odometry solution is very close to the reference path, meaning that based solely on odometry (as is the case in these experiments), the tracker maintains the robot very close to the reference path (the minimum distance is below 10mm in both cases). This implies that if a better localization technique is used, our tracking scheme would perform with more accuracy.

## 9. Conclusions

In this chapter we have analysed and demonstrated the applicability of the strip-wise affine transform in the path tracking task for mobile robots. The transformation was translated to hardware and implemented into an FPGA chip with the use of VHDL and advanced EDA software. The scalability of the fuzzy controller core allowed easy parameter adaptation of the theoretic fuzzy tracker model. The experiments show that the tracking scheme performs satisfactory but is degraded by the accumulation of errors of the odometry used in estimating the robots position.

## 10. References

Abdessemed, F., Benmahammed, K. & Monacelli, E. (2004). A fuzzy-based reactive controller for a non-holonomic mobile robot, *Robotics and Autonomous Systems* 47(1): 31–46.

Altafini, C. (1999). A Path-Tracking criterion for an LHD articulated vehicle, *The International Journal of Robotics Research* 18(5): 435–441.

Altafini, C. (2002). Following a path of varying curvature as an output regulation problem, *Automatic Control, IEEE Transactions on* 47(9): 1551–1556.

Antonelli, G., Chiaverini, S. & Fusco, G. (2007). A Fuzzy-Logic-Based approach for mobile robot path tracking, *Fuzzy Systems, IEEE Transactions on* 15(2): 211–221.

Baltes, J. & Otte, R. (1999). A fuzzy logic controller for car-like mobile robots, *Computational Intelligence in Robotics and Automation, 1999. CIRA '99. Proceedings. 1999 IEEE International Symposium on*, Monterey, CA , USA, pp. 89–94.

Cao, M. & Hall, E. L. (1998). Fuzzy logic control for an automated guided vehicle, *Intelligent Robots and Computer Vision XVII: Algorithms, Techniques, and Active Vision* 3522(1): 303–312.

Costa, A., Gloria, A. D., Giudici, F. & Olivieri, M. (1997). Fuzzy logic microcontroller, *IEEE Micro* 17(1): 66–74.

Deliparaschos, K. M., Nenedakis, F. I. & Tzafestas, S. G. (2006). Design and implementation of a fast digital fuzzy logic controller using FPGA technology, *Journal of Intelligent and Robotics Systems* 45(1): 77–96.

Deliparaschos, K. M. & Tzafestas, S. G. (2006). A parameterized T-S digital fuzzy logic processor: soft core VLSI design and FPGA implementation, *International Journal of Factory Automation, Robotics and Soft Computing* 3: 7–15.

Deliparaschos, K., Moustris, G. & Tzafestas, S. (2007). Autonomous SoC for fuzzy robot path tracking, *Proceedings of the European Control Conference 2007*, Kos, Greece.

Egerstedt, M., Hu, X. & Stotsky, A. (1998). Control of a car-like robot using a dynamic model, *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*, Vol. 4, pp. 3273–3278 vol.4.

El Hajjaji, A. & Bentalba, S. (2003). Fuzzy path tracking control for automatic steering of vehicles, *Robotics and Autonomous Systems* 43(4): 203–213.

Fortuna, L., Presti, M. L., Vinci, C. & Cucuccio, A. (2003). Recent trends in fuzzy control of electrical drives: an industry point of view, *Proceedings of the 2003 International Symposium on Circuits and Systems*, Vol. 3, pp. 459–461.

Gardner, R. B. & Shadwick, W. F. (1987). Feedback equivalence of control systems, *Systems & Control Letters* 8(5): 463–465.

Gardner, R. B. & Shadwick, W. F. (1990). Feedback equivalence for general control systems, *Systems & Control Letters* 15(1): 15–23.

Groff, R. E. (2003). *Piecewise Linear Homeomorphisms for Approximation of Invertible Maps*, PhD thesis, The University of Michigan.

Gupta, H. & Wenger, R. (1997). Constructing piecewise linear homeomorphisms of simple polygons, *J. Algorithms* 22(1): 142–157.

Hung, D. L. (1995). Dedicated digital fuzzy hardware, *IEEE Micro* 15(4): 31–39.

Jiangzhou, L., Sekhavat, S. & Laugier, C. (1999). Fuzzy variable-structure control for nonholonomic vehicle path tracking, *Intelligent Transportation Systems, 1999. Proceedings. 1999 IEEE/IEEJ/JSAI International Conference on*, pp. 465–470.

Kamga, A. & Rachid, A. (1997). A simple path tracking controller for car-like mobile robots, *ECC97 Proc.*

Kanayama, Y. & Fahroo, F. (1997). A new line tracking method for nonholonomic vehicles, *Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on*, Vol. 4, pp. 2908–2913 vol.4.

Koh, K. & Cho, H. (1994). A path tracking control system for autonomous mobile robots: an experimental investigation, *Mechatronics* 4(8): 799–820.

Kongmunvattana, A. & Chongstivatana, P. (1998). A FPGA-based behavioral control system for a mobile robot, *Circuits and Systems, 1998. IEEE APCCAS 1998. The 1998 IEEE Asia-Pacific Conference on*, pp. 759–762.

Kumar, S. (1995). *A unified representation for hardware/software codesign*, PhD thesis, University of Virginia. UMI Order No. GAX96-00485.

Lee, T., Lam, H., Leung, F. & Tam, P. (2003). A practical fuzzy logic controller for the path tracking of wheeled mobile robots, *Control Systems Magazine, IEEE* 23(2): 60– 65.

Leong, P. & Tsoi, K. (2005). Field programmable gate array technology for robotics applications, *Robotics and Biomimetics (ROBIO). 2005 IEEE International Conference on*, pp. 295–298.

Li, T., Chang, S. & Chen, Y. (2003). Implementation of human-like driving skills by autonomous fuzzy behavior control on an FPGA-based car-like mobile robot, *Industrial Electronics, IEEE Transactions on* 50(5): 867– 880.

Liu, K. & Lewis, F. (1994). Fuzzy logic-based navigation controller for an autonomous mobile robot, *Systems, Man, and Cybernetics, 1994. 'Humans, Information and Technology'., 1994 IEEE International Conference on*, Vol. 2, pp. 1782–1789 vol.2.

Maalouf, E., Saad, M. & Saliah, H. (2006). A higher level path tracking controller for a four-wheel differentially steered mobile robot, *Robotics and Autonomous Systems* 54(1): 23–33.

Moustris, G. P. & Tzafestas, S. G. (2011). Switching fuzzy tracking control for mobile robots under curvature constraints, *Control Engineering Practice* 19(1): 45–53.

Moustris, G. & Tzafestas, S. (2005). A robust fuzzy logic path tracker for non-holonomic mobile robots., *International Journal on Artificial Intelligence Tools* 14(6): 935–966.

Moustris, G. & Tzafestas, S. (2008). Reducing a class of polygonal path tracking to straight line tracking via nonlinear strip-wise affine transformation, *Mathematics and Computers in Simulation* 79(2): 133–148.

Navabi, Z. (1998). *VHDL: analysis and modeling of digital systems*, McGraw-Hill Professional.

Ollero, A., Garcia-Cerezo, A., Martinez, J. L. & Mandow, A. (1997). Fuzzy tracking methods for mobile robots, *in* M. Jamshidi, A. Titli, L. Zadeh & S. Boverie (eds), *Applications of fuzzy logic: Towards high machine intelligence quotient systems*, Prentice-Hall, New Jersey.

Patterson, D. A. & Hennessy, J. L. (1997). *Computer Organization and Design: The Hardware/Software Interface*, 2 edn, Morgan Kaufmann.

Preparata, F. P. & Supowit, K. J. (1981). Testing a simple polygon for monotonicity, *Info. Proc. Lett.* 12(4): 161–164.

Raimondi, F. & Ciancimino, L. (2008). Intelligent neuro-fuzzy dynamic path following for car-like vehicle, *Advanced Motion Control, 2008. AMC '08. 10th IEEE International Workshop on*, pp. 744–750.

Reynolds, R., Smith, P., Bell, L. & Keller, H. (2001). The design of mars lander cameras for mars pathfinder, mars surveyor '98 and mars surveyor '01, *Instrumentation and Measurement, IEEE Transactions on* 50(1): 63–71.

Rodriguez-Castano, A., Heredia, G. & Ollero, A. (2000). Fuzzy path tracking and position estimation of autonomous vehicles using differential GPS, *Mathware Soft Comput* 7(3): 257–264.

Rozenblit, J. & Buchenrieder, K. (1996). *Codesign: Computer-aided Software/Hardware Engineering*, I.E.E.E.Press.

Sailer, P. M., Sailer, P. M. & Kaeli, D. R. (1996). *The DLX Instruction Set Architecture Handbook*, 1st edn, Morgan Kaufmann Publishers Inc.

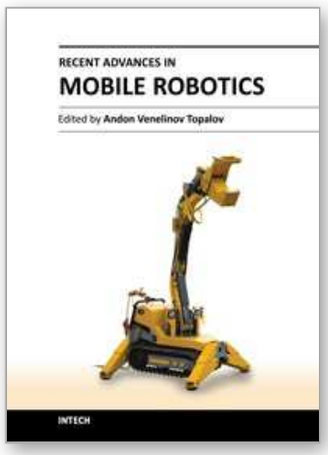
Salapura, V. (2000). A fuzzy RISC processor, *IEEE Transactions on Fuzzy Systems* 8(6): 781–790.

Samson, C. (1995). Control of chained systems application to path following and time-varying point-stabilization of mobile robots, *Automatic Control, IEEE Transactions on* 40(1): 64–77.

Sanchez, O., Ollero, A. & Heredia, G. (1997). Adaptive fuzzy control for automatic path tracking of outdoor mobile robots. application to romeo 3R, *Fuzzy Systems, 1997., Proceedings of the Sixth IEEE International Conference on*, Vol. 1, pp. 593–599 vol.1.

Wit, J., Crane, C. D. & Armstrong, D. (2004). Autonomous ground vehicle path tracking, *J. Robot. Syst.* 21(8): 439–449.

Yang, X., He, K., Guo, M. & Zhang, B. (1998). An intelligent predictive control approach to path tracking problem of autonomous mobile robot, *Systems, Man, and Cybernetics, 1998. 1998 IEEE International Conference on*, Vol. 4, pp. 3301–3306 vol.4.




**Recent Advances in Mobile Robotics**

Edited by Dr. Andon Topalov

ISBN 978-953-307-909-7
Hard cover, 452 pages
**Publisher** InTech
**Published online** 14, December, 2011
**Published in print edition** December, 2011

Mobile robots are the focus of a great deal of current research in robotics. Mobile robotics is a young, multidisciplinary field involving knowledge from many areas, including electrical, electronic and mechanical engineering, computer, cognitive and social sciences. Being engaged in the design of automated systems, it lies at the intersection of artificial intelligence, computational vision, and robotics. Thanks to the numerous researchers sharing their goals, visions and results within the community, mobile robotics is becoming a very rich and stimulating area. The book Recent Advances in Mobile Robotics addresses the topic by integrating contributions from many researchers around the globe. It emphasizes the computational methods of programming mobile robots, rather than the methods of constructing the hardware. Its content reflects different complementary aspects of theory and practice, which have recently taken place. We believe that it will serve as a valuable handbook to those who work in research and development of mobile robots.

**How to reference**

In order to correctly reference this scholarly work, feel free to copy and paste the following:

G.P. Moustris, K.M. Deliparaschos and S.G. Tzafestas (2011). Feedback Equivalence and Control of Mobile Robots Through a Scalable FPGA Architecture, Recent Advances in Mobile Robotics, Dr. Andon Topalov (Ed.), ISBN: 978-953-307-909-7, InTech, Available from: http://www.intechopen.com/books/recent-advances-in-mobile-robotics/feedback-equivalence-and-control-of-mobile-robots-through-a-scalable-fpga-architecture

INTECH
open science | open minds

**InTech Europe**
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

**InTech China**
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

© 2011 The Author(s). Licensee IntechOpen. This is an open access article distributed under the terms of the Creative Commons Attribution 3.0 License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

IntechOpen

IntechOpen