I. Troch, F. Breitenecker, eds. ISBN 978-3-901608-35-3

MODELS OF COMPUTATION FOR REACTIVE CONTROL OF AUTONOMOUS MOBILE ROBOTS

G. Rigatos[1]
[1] Unit of Industrial Automation, Industrial Systems Institute, Greece

Corresponding author: G. Rigatos, Unit of Industrial Automation, Industrial Systems Institute, Stadiou str., 26504, Rion Patras, Greece
grigat@isi.gr

Abstract. The paper studies computation models for tasks performed by autonomous mobile robots. Such tasks can be accomplished by reactive control algorithms. The evolution of reactive control systems can be described using different models of computation which have as distinguishing feature the abstraction level of time. Thus, three computation models are defined: the untimed model, the synchronous model and the timed model. It is shown that the clocked-synchronous model of computation is more appropriate for describing the controller for a parallel parking task.

1 Introduction

The motivation of this paper is to develop computation models for tasks performed by autonomous mobile robots. Such models can help to analyze the functioning of the associated control algorithms and to study their stability properties [1]. Reactive control has been successfully applied to autonomous mobile robots and has enabled robotic vehicles to perform various tasks, such as parallel parking or motion with desirable speed in uncertain environments [2-4]. Reactive systems receive inputs, react to them by computing outputs and wait for the next inputs to arrive. Reactive systems correspond to finite state machines which in turn can be represented with the use of Petri nets [5-8]. The evolution of reactive control systems can be described using different models of computation which have as distinguishing feature the abstraction level of time. Thus, three computation models are defined: the untimed model, the synchronous model and the timed model [5].

First, the *untimed model* of computation is considered. This adopts the simplest timing approach, in which processes are modeled as state machines which are connected to each others via signals. Signals transport data values which do not carry any time information but preserve their order of emission. Values that are emitted first are assumed to be received first by the receiving process. Second the *synchronous model* of computation is considered. This can be based on partition of time either into time slots or into clock cycles. The *perfectly synchronous* model assumes that no time advances during the evaluation of a process. Consequently, the results of a computation on input values are already available in the same cycle. The *clocked-synchronous* model assumes that every simulation step of a process takes one cycle. Hence the reaction of a process to an input becomes effective in the next cycle. Third, the *timed model* of computation is examined which assigns a time stamp to each value communicated between processes. This allows to model time-related issues in great detail, but it complicates the model and the task of analysis and simulation.

It will be shown that the clocked-synchronous model of computation is more appropriate for describing the controller for the parallel parking task. Results on the efficiency of the proposed control algorithm have been obtained in the case of a 4-wheel mobile robot (Coroware CB-WA). This research work will be continued with the detailed presentation of the experimental implementation of the reactive parking controller on the four wheel robot.

The structure of the paper is as follows: In Section 2 the use of finite state machines and Petri Nets in the modeling of computation processes is analyzed. In Section 3 the basic elements of the various models of computation (MoC) are presented and the untimed model of computation is explained. In Section 4 the synchronous model of computation is introduced. In Section 5 the timed model of computation is presented. In Section 6 the modeling of a parallel parking controller by a finite state machine is analyzed and the associated untimed, synchronous and timed models of computation are studied. Finally, in Section 7 concluding remarks are stated.

2 Finite state machines and Petri Nets models

2.1 System modeling with the use of deterministic automata

Automata have been used to model the dynamics of Discrete Event Systems (DES). In that case automata describe dynamical systems the behavior of which cannot be completely represented by differential equations, because of the existence of asynchronous events that affect the system's state. Such systems are usually met in manufacturing, robotics and in intelligent autonomous vehicles. Usually a DES is described by a finite automaton, which is defined by the five-tuple [9-12]

$$M = (\Phi, B, \delta, S, F) \tag{1}$$

where Φ is a set of discrete states, B is the set of events that enable the transition between states, $\delta : \Phi \times B \rightarrow \Phi$ is the transitions mapping, and $q_0 \in \Phi$ is the initial state. An example of a DES is depicted in Fig. 1 where $\Phi = \{E, F, P\}$, $B = \{\alpha_1, \alpha_2, \alpha_3, \beta_1, \beta_2, \beta_3\}$, $S = E$, S contains the arcs that appear in Fig. 1. At time instant k the system can be in one of the nodes depicted in the above graph, for instance node i, thus the state of the system is the vector $\phi^k = [0, \cdots, 1, \cdots, 0]$.

The transition between the various states of the automaton are enabled through the events $a_i,\ i = 1,2,3$, and $\beta_i,\ i = 1,2$. Each event is associated with a transition matrix which in the case of the automaton shown in Fig. 1 becomes [9]

$$\alpha_1 = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \beta_1 = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \alpha_2 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}$$
$$\beta_2 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}, \alpha_3 = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \beta_3 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix} \tag{2}$$

Thus, if the system is initially is state $F = [0,1,0]$ and event β_1 appears then the next state of the DES is state $E = [1,0,0]$ as shown in the following calculation

$$\begin{pmatrix} 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \end{pmatrix} \tag{3}$$

The properties of reachability, cyclic behavior and asymptotic stability are of importance for the efficient performance of the computations modeled by the DES [12].

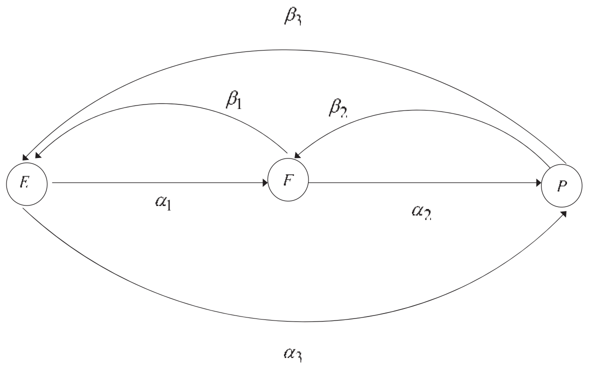

Figure 1: Modelling of a discrete event system with the use of an automaton

2.2 System modelling with the use of fuzzy automata

The need for non-deterministic finite state machines in the description of robotic and autonomous vehicle controllers is justified as follows [5]: First it may be unknown what the exact effect of a particular event is. The modeling of the breaking behavior of a vehicle can be used as an example. When the driver steps on the break with a given force, it will not always have the same effect and decrease the velocity by the same amount. The precise effect depends on many factors, such as the age of the vehicle, the friction of the tires and the condition of the road. If one includes all possibilities then a stochastic model of the car's behavior is obtained. A second situation where nondeterminism is useful arises when an exact model of the system's behavior may be complicated and untractable. Since one has to deal with all possibilities when designing or analyzing a vehicle it may be superfluous to include all details of cause and effect in the model.

Crisp finite state machines are not adequate for applications in which the states and the transitions of a system are always somewhat uncertain. Subjective human observation, judgement and interpretation invariably play a significant role in describing the status of a state, usually not crisp. To overcome these limitations fuzzy finite state machines have been proposed [7]. In fuzzy finite state machines the possibility of being at a state of the automaton depicted in Fig. 1, at a time instant k, is denoted by a fuzzy membership function. Assume that the states of the automaton are $\phi_1, \phi_2, \cdots, \phi_n$, and the associated membership functions at time instant k are $[\mu_1^k, \mu_2^k, \cdots, \mu_n^k]$. Then at time instant k the fuzzy finite state machine is at a state defined by the vector $g^k = [\mu_1^k, \cdots, \mu_n^k]$. The transition matrix between the state of the fuzzy state machine at time instant k and its state at time instant $k+1$ is given by

$$\alpha = [\alpha_{ij}]_{n\times n} = \begin{pmatrix} \alpha_{11} & \cdots & \alpha_{1n} \\ \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots \\ \alpha_{n1} & \cdots & \alpha_{nn} \end{pmatrix} \tag{4}$$

A fuzzy discrete event system is represented by a fuzzy automaton where [9,12]: (i) the possibility to find the automaton at a certain state at time instant k is given by a fuzzy membership function, (ii) the possibility of a transition between states to take place is also given by a fuzzy membership function, Assume that the states of the automaton depicted in Fig. 1 are fuzzy and the associated membership functions are

$$\tilde{\phi} = [0.4, 0.8, 0] \tag{5}$$

It is also assumed that the fuzzy event $\tilde{\alpha}_1$ is given by the transition matrix

$$\tilde{\alpha}_1 = \begin{pmatrix} 0.1 & 0.9 & 0.1 \\ 0.2 & 0.1 & 0.2 \\ 0 & 0.1 & 0.1 \end{pmatrix} \tag{6}$$

i.e. again it is most likely to have a transition from state E to state F. Then using the max-product inference one has

$$\tilde{\phi} \circ \tilde{\alpha}_1 = [0.4, 0.8, 0] \circ \begin{pmatrix} 0.1 & 0.9 & 0.1 \\ 0.1 & 0.1 & 0.1 \\ 0 & 0.1 & 0.1 \end{pmatrix} = \begin{pmatrix} 0.08 & 0.36 & 0.08 \end{pmatrix} \tag{7}$$

A typical definition of a fuzzy automaton is the five-tuple $\tilde{M} = (\tilde{\Phi}, \tilde{B}, \tilde{\delta}, \tilde{s}, \tilde{F})$, where

- $\tilde{\Phi}$ is the finite set of fuzzy states. A membership value $\mu(\phi_i) \in [0,1]$ is assigned to each state.
- $\tilde{B}$ is the set of inputs where each input has a membership function $\mu(b_i) \in [0,1]$.
- $\tilde{\delta} : \Phi \times B \rightarrow \Phi$ is the set of fuzzy transitions, where a membership function $\mu(\tilde{\delta}) \in [0,1]$ is associated with each transition from state ϕ_i to state ϕ_j.
- $\tilde{s}$ is the fuzzy start state.
- $\tilde{F} \subset \tilde{\Phi}$ is the set of fuzzy final states.

2.3 Modeling of event-driven systems with the use of Petri-Nets

Models based on finite state machines focus on the state of a system and the observable input-output behavior. They are not well suited to studying the interaction of concurrently active parts of a system and the combined realization behavior of distributed parallel systems. To address issues of concurrency and synchronization state machines are generalized into a model of communicating, concurrent state machines. Petri Nets make suitable the study of concurrency in finite state machines. The means of communication is a token, which does not contain any data. A Petri Net is described by the five-tuple

$$PN = (P, T, A, W, M_0) \tag{8}$$

where

$P = \{p_1, p_2, \cdots, p_n\}$ is a finite set of place
$T = \{t_1, t_2, \cdots, t_m\}$ is a finite set of transitions
$A \subset (P \times T) \cup (T \times P)$ is a set of arcs that connect positions to transitions
$W : A \rightarrow R^+$ is an arc weight function (represented with numbers that label the arcs)
$M_0 : P \rightarrow N$ is the initial marking (initial number of tokens in places)

The Petri Net structure is $N = \{P, T, F, W\}$ so $PN = (M, W_0)$. The expression $PN = (N, M_0)$ is usually referred as general Petri Net. It should also be noted that marked graphs and state machines are special cases of Petri Nets. A Petri-Net model that describes the non-deterministic velocity controller of a mobile robot is shown in Fig. 2 [12].

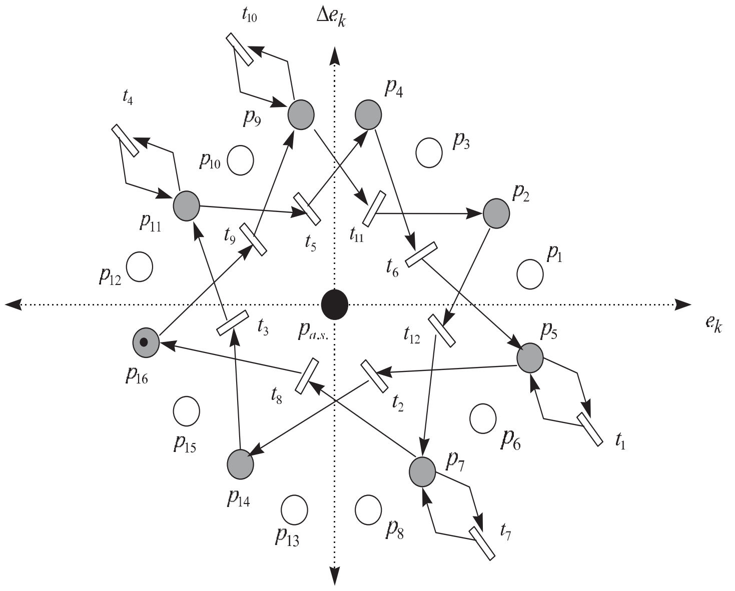

Figure 2: A Petri Net model with initial marking

A qualitative description for the Petri Net of Fig. 2 is as follows: The vector $[e_k, \Delta e_k]$ describes the velocity error of a mobile robot ($e_k = v_k - v_k^d$: difference between the current value of the velocity v_k and the desirable value v_k^d). The velocity is controlled by incremental changes (acceleration or deceleration during a computation cycle) which is denoted by the transition t_i of the state machine. Such a dynamic model has a monotonous input-output behaviour ($\dot{v} = a(t)$, or $v_{k+1} = v_k + a$, i.e. the longer acceleration is applied to the model the larger the vehicle's speed becomes). Using an appropriate sequence of control events (accelerations/decelerations) as defined by the transitions of the Petri Net of Fig. 2, and by diminishing the acceleration a every time the sign of $e_k \cdot e_{k-1}$ changes, one can succeed to keep the error state vector in the quarter-plane $e_k \Delta e_k < 0$, which means that finally the velocity error converges to 0.

3 Basic terms in models of computation

3.1 Models of Computation (MoC)

The computation programs which substantiate certain tasks in autonomous vehicle operation are denoted with the term *process*. In all models of computation *events* are the elementary units of information exchange between processes, or between the internal states of a process. *Processes* receive or consume events, and they send or emit events. The medium through which events are communicated from one process to the other is called *signal*. A signal can be potentially an infinite sequence of events. The activity of each process is divided into *evaluation cycles*. In each evaluation cycle a process receives inputs and produces outputs. A process is a software that performs specific computations and is usually represented by a *finite state machine*, or equivalently by a *Petri Net*. A process partitions its inputs and outputs into subsequences which correspond to evaluation cycles. For instance in the untimed and synchronous models of computation only one event takes place during an evaluation cycle while in the timed model of computation multiple events can take place during the evaluation cycle. *Process constructors* are parametrizable templates that instantiate processes. Through a process constructor one can define the structure of the state machine that represents the performed computation, as well as the number of inputs and outputs. A *Model of Computation* (MoC) is a process, or set of processes or process networks that can be generated by certain process constructors [5].

A MoC is primarily characterized by the abstraction level of time it uses. Thus one has (i) the *untimed* MoC in which no time measurement is associated with the arrival of input events and the production of output events by the process, (ii) the *synchronous* MoC in which only one input event appears at a specific time slot and only one output event is also recorded at a certain time slot [13,14], (iii) the *timed* model of computation in which several input and output events can appear at a specific time slot.

3.2 Events and signals in models of computation

Untimed events $\dot{E}$ are just values without further information, $\dot{E} = V$. *Synchronous events* $\bar{E}$ include a pseudovalue $\sqcup$ (denoted as "absent") in addition to the normal values. Hence $\bar{E} = V \cup \{\sqcup\}$. *Timed events* $\hat{E}$ are identical to synchronous events. Intuitively, timed events occur at much finer granularity than synchronous events, for instance at a nanoseconds time period. In contrast synchronous events are assigned to abstract time slots or clock cycles. This model of events and time can accommodate discrete time models.

Signals are sequences of events. Sequences are ordered, and one uses subscripts such as e_i to denote the i-th event in a signal. For example, a signal may be written as $< e_0, e_1, e_2 >$. In general, signals can be finite or infinite sequences of events, and S is the set of all signals. One can distinguish between three different kinds of singals. $\dot{S}$, $\bar{S}$ and $\hat{S}$ denote the untimed, synchronous and timed signal sets, and $\dot{s}$, $\bar{s}$ and $\hat{s}$ designate individual untimed, synchronous and timed signals (see Fig. 3).

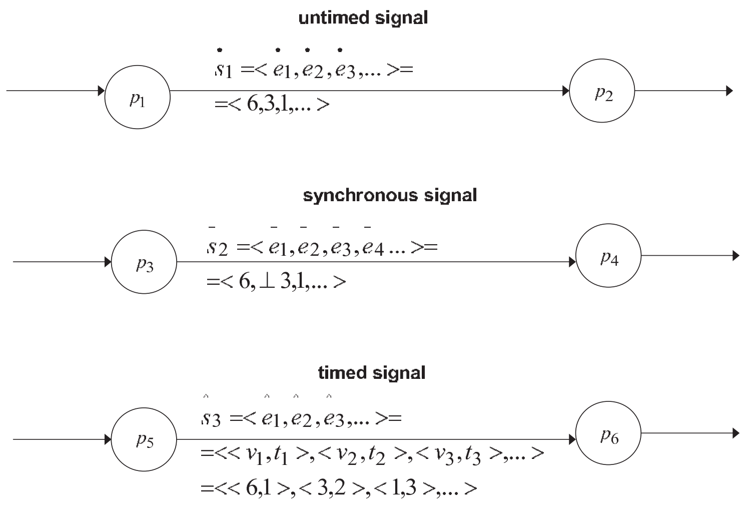
untimed signal
$\dot{s}_1 =< \dot{e}_1, \dot{e}_2, \dot{e}_3, ... >=$
$=< 6,3,1,... >$
p_1
p_2
synchronous signal
$\bar{s}_2 =< \bar{e}_1, \bar{e}_2, \bar{e}_3, \bar{e}_4 ... >=$
$=< 6, \perp 3,1,... >$
p_3
p_4
timed signal
$\hat{s}_3 =< \hat{e}_1, \hat{e}_2, \hat{e}_3, ... >=$
$=<< v_1, t_1 >, < v_2, t_2 >, < v_3, t_3 >, ... >$
$=<< 6,1 >, < 3,2 >, < 1,3 >, ... >$
p_5
p_6

Figure 3: Processes p_1 and p_2 are connected by the untimed signal $\dot{s}_1$. Processes p_3 and p_4 are connected by the synchronous signal $\bar{s}_2$. Processes p_5 and p_6 are connected by the timed signal $\hat{s}_3$.

3.3 Process properties

Important properties of processes are monotonicity and continuity. A process $p:\ S \rightarrow S$ is monotonic if

$$s_1 \subseteq s_2 \Rightarrow p(s_1) \subseteq s_2 p(s_2) \tag{9}$$

Monotonicity means that receiving more input can only provoke a process to generate more output, but does not change the already emitted output. This means that a process can start computing before all the input is available because new input will only add to the previously created output, but not change it.

Continuity, guarantees that infinite output signals can be gradually approximated. A monotonic process p is continuous if

$$p(\sqcup C) = \sqcup p(C)\ (:= \sqcup \{p(s) | s \in C\}) \tag{10}$$

for every chain $C \subseteq S$. Moreover, a process with two or more inputs, i.e. receiving measurements from two or more sensors is sequential if the inputs are not sequentially processed. This means that if the one sensor stops providing measurements the controller's functioning will be blocked. Finally, it is noted that processes can be combined into a process network through parallel composition, sequential composition and feedback operators.

3.4 The untimed Model of Computation

An untimed model of computation (MoC) is a 2-tuple $MoC = (C, O)$, where C is the set of the constructors of the process, each of which, when given constructor specific parameters instantiates the process. O is a set of process composition operators (if needed to combine processes into a process network), each of which when given processes as arguments instantiates a new process. The created process is a state machine which functions according to the basic assumption of the untimed computation, i.e. that no time measurement is associated with the arrival of input events or the production of output events by the process.

4 Synchronous models of computation

4.1 Perfect synchrony

Synchronous models of computation divide the time axis into slots. The evaluation cycle of the process lasts exactly one time slot. There are two synchronous MoC. In the *perfectly synchronous* MoC the output events of a process occur in the same time slot as the corresponding input events. This leads to interesting situations when an output event of a process becomes also input event of the same process in the same time slot and in this way it contributes to its own creation. This situation corresponds to a set of recursive equations.

The *perfect synchrony hypothesis* assumes that neither computation, nor communication takes time. In a perfectly synchronous model, inputs arrive in a particular order, and even though they do not carry information about the concrete time instances when this happens (unlike the timed model of computation), one can observe that event u_1 arrives at the system before u_2.

Process P reacts to the events u_1 and v_1 immediately, by computing outputs $u_1^{''}$ and $v_1^{''}$ and then it waits for the next inputs. Thus one can observe a sequence of cycles (read inputs, compute outputs). Since, neither computation nor communication takes time, the outputs occur at exactly the same time as the inputs. Thus $u_1, v_1, u_1^{''}, v_1^{''}$ and all corresponding intermediate events $u_1^{'}, v_1^{'}$ occur simultaneously. Thus, the time instances of the occurrence of the output events are completely determined by the time instances of the occurrence of the input events. *Reactive systems* receive inputs, react to them by computing outputs and wait for the next inputs to arrive.

Synchronous processes have two specific characteristics. First, all synchronous processes consume and produce exactly one event one event on each input or output in each evaluation cycle. Second, events can take the special value $\sqcup$ which denotes the absence of an event. In this way one can define synchronous events $\bar{E}$ and synchronous signals $\bar{S}$. The synchronous process process constructors substantiate processes which are able to deal with synchronous events and signals.

In the *perfectly synchromous MoC* the process constructor creates a process that operates according to the perfect synchrony hypothesis, i.e that neither computation, not communication takes time and thus for a certain input event occurring at the *i*-th time slot the associated output event takes also place at the same time slot.

4.2 The clocked-synchronous model of computation

In the *clocked synchronous MoC* every process incurs a delay from an input to an output event. The delay is equivalent to the duration of the evaluation cycle. Consequently, feedback loops loose their difficulty because there is always a delay of at least one evaluation cycle in each loop and events cannot affect their own generation.

On the other hand, in the *clocked synchronous MoC* the clocked synchronous hypothesis holds: there is a global clock signal controlling the start of each computation in the system. Communication, takes no time, and computation takes one clock cycle. To describe clocked synchronous processes a delay function Δ is considered, which delays each input by one cycle, as shown in Fig. 4.

In the *clocked synchromous MoC* the process constructor creates a process that operates according to the clocked synchronous hypothesis, i.e that communication takes no time while computation takes one clock cycle and thus for a certain input event occurring at the *i*-th time slot the associated output event takes also place at the next time slot. A delay Δ between input and output events is thus recorded.

5 Timed models of computation

5.1 Basic properties in the timed MoC

The *timed MoC* is an elaboration of the synchronous MoC. The differences form the synchronous MoC are the following: (i) the partition of the time axis is much finer, and events may occur at each nanosecond or picosecond rather than every clock cycle (each clock cycle may contain several events). However, the physical time unit is not part of the MoC, (ii) Processes can receive as input and emit as output any number of events during an evaluation (clock) cycle. From this point of view the timed MoC is closer to the untimed than to the synchronous MoC, (iii) processes must comply with the causality constraint, which means that those output events which are a reaction to input events cannot occur before these input events. When a process consumes a number of input events in a given evaluation cycle, the first output event of this evaluation cycle is emitted not earlier than the latest input event of this cycle. As a consequence a delay period is associated with each evaluation cycle of the process.

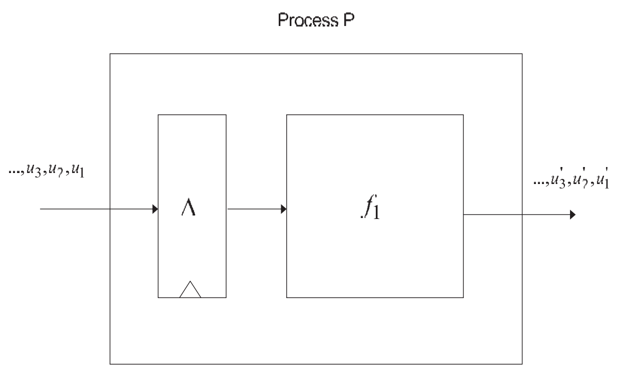

Figure 4: A simple clocked synchronous process P

Using a timed MoC reflects the intention of capturing the timing behavior of physical entities accurately without simplifying assumptions. The only simplification concerns the accuracy of the time representation (unit of the physical time).

5.2 The causality constraint in the timed MoC

A process constructor for the timed MoC creates a process which functions according to the constraint that output events cannot occur before the input events of the same evaluation cycle. This is achieved by enforcing an equal number of input and output events for each evaluation cycle and by keeping a sequence of absent events. Since the signals also represent the progression of time, the appearance of absent events at the outputs corresponds to an initial delay of the process in reacting to the inputs. Moreover, each event is annotated with its own time tag which annotates the time of its occurrence (with reference to a global or local timer).

6 Simulation and experimental tests

6.1 Description of the parking algorithm

Parallel parking is the placement of a vehicle in parallel to its moving direction in a confined space which is very little wider than the vehicle's dimensions. The mobile robot can be described as a four-wheel vehicle, the front wheels of which are used to turn its direction in an angle up to 45^o in both directions. The vehicle model is shown in Fig. 5.

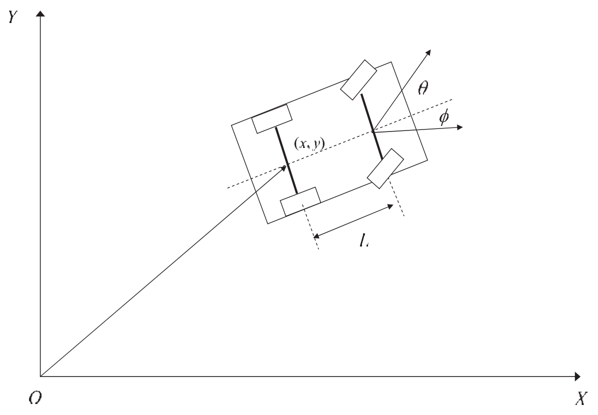

Figure 5: The model of the robotic vehicle

The position of the vehicle is described by the coordinates (x,y) of the center of its rear axis and its direction is given by the angle ϕ between the x-axis and the axis of the direction of the vehicle. The steering angle θ and the speed u are considered to be the inputs of the system. The continuous-time equations that describe the vehicle's motion are the following:

$$\begin{aligned} \dot{x} &= cos(\theta)cos(\phi)v \\ \dot{\psi} &= cos(\theta)sin(\phi)v \\ \dot{\phi} &= sin(\theta)\tfrac{v}{l} \end{aligned} \tag{11}$$

The parking space is described by a rectangle of dimensions α and β (see Fig. 2). The proposed parallel parking algorithm can be decomposed into the following steps [2]:

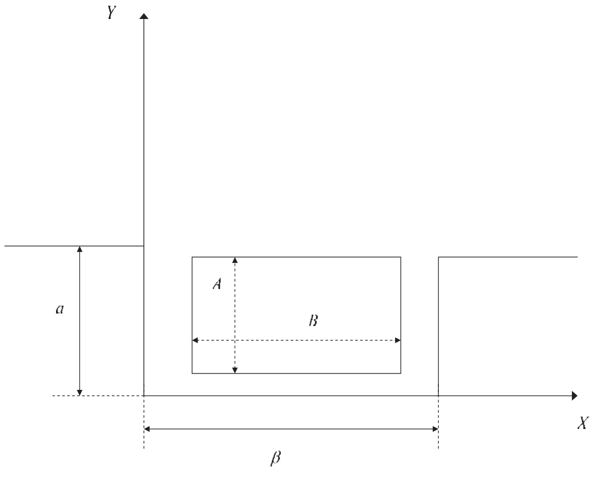

Figure 6: Parking place and vehicle

Step 1: The vehicle is set parallel and ahead of the parking area.

Step 2: The vehicle backtracks to a certain point and then the front wheels are turned so that the vehicle moves towards the parking area until it reaches a certain angle of approach.

Step 3: As long as the vehicle stays inside the boundaries of the parking place, the vehicle continues to backtrack.

Step 4: When the vehicle reaches the boundaries, the front wheels are turned the other way round and the direction of movement also changes.

Step 5: Step 3 is repeated until the vehicle is found in a direction parallel to the desired one, but inside the limitations of the parking place. The parking is then completed.

The parking place is defined by a rectangle as shown in Fig.2. It is assumed that the car is initially on the right of the parking place with $y_0 = h$ and $x_0 = 0$. The parking manoeuvre is parameterized by ϕ_{ref} and x_{ref}, which are given by:

$$\begin{gathered} sin(\phi_{ref}) = \tfrac{a}{B} \\ x_{ref} = \tfrac{1}{tan(\phi_ref)}(h + \tfrac{\alpha}{2} - \tfrac{L}{tan(\theta)} - Acos(\phi_{ref})) + \tfrac{1}{sin(\phi_ref)}(\tfrac{A}{2} + \tfrac{L}{tan(\theta)}) \end{gathered} \tag{12}$$

In the above equation

α stands for the width of the parking area, β stands for the length of the parking area, A stands for the width of the car, B stands for the total length of the car, h describes the initial y position of the center of the rear axis of the car (i.e. $y_0 = h$), L is the length of the wheel-base of the car, ϕ_{ref} is the initial angle approach of the parking area which is defined by the transversal axis of the vehicle and the x-axis, x_{ref} is the starting x position of the center of the rear axis of the car.

The manoeuvres' sequence is demonstrated in Fig. 7. The equations defining x_{ref} and ϕ_{ref} can be interpreted as follows: the aim is for the car to enter the parking place with the highest angle possible, such that the right front part of the car remains within the parking place. x_{ref} is selected such that, after manoeuvres (K_0), (K_1) and (K_2) are completed, the vehicle reaches the left border of the parking area with its left rear corner, in order to have as much as room as possible to re-orient.

The parking manoeuvres consist of the following steps: the vehicle backs up (K_0) until the back of the car reaches x_{ref}, it turns right (K_1) until the orientation overshoots ϕ_{ref}, then it backs up (K_2) until the rear of the car touches the left or the lower bound of the parking area. Finally, the vehicle re-orients by repeating the following sequence: (i) if the rear part touches the border of the parking place (K_3), it drives forward and turns right (K_4), (ii) if the front part touches the border (K_5), or if the left rear corner of the car reaches the upper limit of the parking place, the vehicle backs up (K_6). As soon as the vehicle becomes parallel to the desirable axis $\phi_d = 0^o$, the parking manoeuvres stop.

6.2 Modeling of the parking controller as a finite state machine

To implement a change of the angle of the steering wheel an increment $\pm\Delta\theta$ is considered, while to implement a change of the vehicle's velocity an increment $\pm\Delta v$ can be considered. The vehicle is set parallel and ahead of the parking place with its rear far ahead from point x_{ref}. The vehicle backtracks to the point x_{ref} and then it turns the front wheels so as to move towards the parking place, until it reaches the angle of approach ϕ_{ref}. The velocity v of the vehicle is kept constant during the parking manoeuvres and its sign is changed each time the vehicle changes its direction of motion. Thus, the control input is taken to be the angle of the steering wheel. The error $e_k = \phi_k - \phi_d$ is defined and its derivative Δe_k is also considered. The parking control manoeuvres can be summarized in the following rules:

R_1: IF $sgn(e_k \Delta e_k) < 0$ AND the previous control action was to increase θ THEN keep on increasing θ.

R_2: IF $sgn(e_k \Delta e_k) < 0$ AND the previous control action was to decrease θ THEN keep on decreasing.

R_3: IF $sgn(e_k \Delta e_k) > 0$ AND the previous control action was to increase θ THEN set $\theta = 0^o$ and decrease θ.

R_4: IF $sgn(e_k \Delta e_k) > 0$ AND the previous control action was to decrease θ, THEN set $\theta = 0^o$ and increase θ.

R_5: IF the vehicle reaches the FRONT or the INSIDE boundary, THEN increase $\Delta\theta$ and change the sign of velocity $sgn(v_k)$.

R_6: IF the vehicle reaches the REAR or the OUTSIDE boundary, THEN increase $\Delta\theta$ and change the sign of velocity $sgn(v_k)$.

The proposed control algorithm is modeled as a finite state machine and can be represented with the use of a Petri Net. At a next stage, using an appropriate model of abstraction the associated untimed computation model can be derived. The finite state machine that models the control algorithm consists of the following places and transitions:

$$\begin{array}{lllll}
p_1: & e_k > 0 & \text{and} & \Delta e_k > 0 & \text{and the last control action is increase} \\
p_2: & e_k > 0 & \text{and} & \Delta e_k > 0 & \text{and the last control action is decrease} \\
p_3: & e_k > 0 & \text{and} & \Delta e_k < 0 & \text{and the last control action is increase} \\
p_4: & e_k > 0 & \text{and} & \Delta e_k < 0 & \text{and the last control action is decrease} \\
p_5: & e_k < 0 & \text{and} & \Delta e_k > 0 & \text{and the last control action is increase} \\
p_6: & e_k < 0 & \text{and} & \Delta e_k > 0 & \text{and the last control action is decrease} \\
p_7: & e_k < 0 & \text{and} & \Delta e_k < 0 & \text{and the last control action is increase} \\
p_8: & e_k < 0 & \text{and} & \Delta e_k < 0 & \text{and the last control action is decrease}
\end{array} \tag{13}$$

The transitions associated with the above states are:

$$\begin{array}{llllll}
t_1: & \text{decrease} & t_5: & \text{increase} & t_9: & \text{increase} \\
t_2: & \text{decrease} & t_6: & \text{decrease} & t_{10}: & \text{decrease} \\
t_3: & \text{increase} & t_7: & \text{decrease} & t_{11}: & \text{increase} \\
t_4: & \text{decrease} & t_8: & \text{increase} & t_{12}: & \text{decrease}
\end{array} \tag{14}$$

The Petri-Net diagram given in Fig. 8 describes the dynamics of the car-controller ensemble and can be initialized randomly at any one of the states $p_1 - p_4$ or $p_5 - p_8$. This means that the initial position error e_k and the velocity error Δe_k of the car, as well as the initial control action $\alpha_i\{increase, decrease\}$ can be chosen arbitrarily.

A token is placed at the starting place and at every iteration of the control algorithm (i.e. each time a new control action α_i is applied) the token moves to a different place. The transition between different places is enabled through the transitions (control actions) $t_1, t_2, t_3, t_4, t_9, t_{10}$ and $t_5, t_6, t_7, t_8, t_{11}, t_{12}$. There are also inactive transitions such as t_3, t_4, t_5 and t_6.

6.3 Convergence analysis of the parking control algorithm

The following notation will be used: (i) the deviation from the desirable angle is denoted by the error function $e = \phi - \phi_d$, (ii) the two control actions are denoted as α_1=increase, and α_2=decrease.

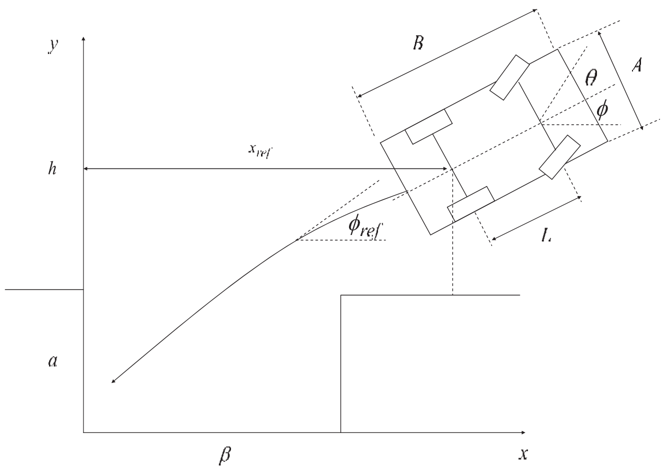

Figure 7: sequence of maneuvers for reactive parallel parking, performed by the autonomous mobile robot

Lemma 1: The error function $e = f(\alpha_i)$ is monotone as long as the same control action is applied.

Proof: A time step equal to 1 clock cycle is assumed, and using Eq. (11) the following equation is considered

$$\Delta\phi_k = sin(\theta_k)\frac{v_k}{l} \tag{15}$$

where, $\Delta\phi_k = \phi_k - \phi_{k-1}$ is the change of the orientation angle ϕ_k. Moreover, $\theta_k \in [\theta_{max}, \theta_{max}] = [-\theta_{max}, 0] \cup [0, \theta_{max}]$ and it is assumed that no direct transition from $[-\theta_{max}, 0]$ to $[0, \theta_{max}]$ or vice versa is permitted (i.e. as long as the same control action α_i, $i = 1, 2$ is applied, the steering angle remains either in $[-\theta_{max}, 0]$ or in $[0, \theta_{max}]$). Finally, without loss of generality it is assumed that the vehicle moves in the forward direction. Thus as long as the same control action is repeated two cases can be distinguished:

$$\text{IF } \theta_k \in [-\theta_{max}, 0] \Rightarrow sin(\theta_k) < 0,\ \Delta e_k < 0 \Rightarrow \phi_k, e_k,\ \text{decrease monotonically} \tag{16}$$

$$\text{IF } \theta_k \in [0, \theta_{max}] \Rightarrow sin(\theta_k) > 0,\ \Delta e_k > 0 \Rightarrow \phi_k, e_k,\ \text{increase monotonically} \tag{17}$$

Lemma 2: Transitions between p_1 and p_2 or p_7 and p_8 are not possible owing to setting the steering angle $\theta = 0^o$ each time a faulty control action is applied.

Proof: Without loss of generality, it is assumed that the system is found in state p_1 with $e_k > 0$ and $\Delta e_k = \Delta\phi_k = sin(\theta_k)\frac{v_k}{l} > 0$, and $v_k > 0$. Then $sin(\theta_k) > 0$, (i.e. $\theta_k \in [0, \theta_{max}]$). According to rule (R_3) θ_k will be set to $\theta_k = 0^o$ and a decrease of θ_k will follow (i.e. now $\theta_k \in [-\theta_{max}, 0]$), which implies $\Delta e_k = \Delta\phi_k = sin(\theta_k)\frac{v_k}{l} < 0$. Hence now one gets $e_k > 0$ and $\Delta e_k < 0$ and the previous control action is *decrease*, i.e. the system goes into state p_4. Therefore, the transition from p_1 to p_2 is not possible. The non-feasibility of the inverse transition from p_2 to p_1, as well as between p_7 and p_8 can be shown in a similar way.

Lemma 3: The states p_3, p_4 and p_5, p_6 of the automaton that models the parking controller operate as sinks. The transitions t_3, t_4 and t_5, t_6 remain inactive.

Proof: Without loss of generality, it is assumed that the system is found in state p_4 (either as an initial state or moving there from state p_1, according to Lemma 2). The possible transitions associated with this state are: (i) t_4 which leads to position p_2, (ii) t_{10} which leads back to position p_4. Both transitions t_4 and t_{10} indicate a 'decrease' control action. This means that, starting from state p_4 the decrease control action is maintained. Therefore, according to Lemma 1 the error function $e_k = f(\alpha_i)$ is a monotonous one (i.e. the sign of Δe_k is not permitted to change).

Since state p_2 corresponds to $\dot{e}$ with opposite sign of p_4 the state from p_4 to p_2 is banned. Consequently, state p_4 operates as sink and transition t_4 is an inactive one (and is denoted in the Petri Net diagram with a dashed line). Similarly, it can be proved that the states p_3, p_5 and p_6 are sinks and that the transitions t_3, t_5 and t_6 remain inactive. Now, the following theorem can be stated:

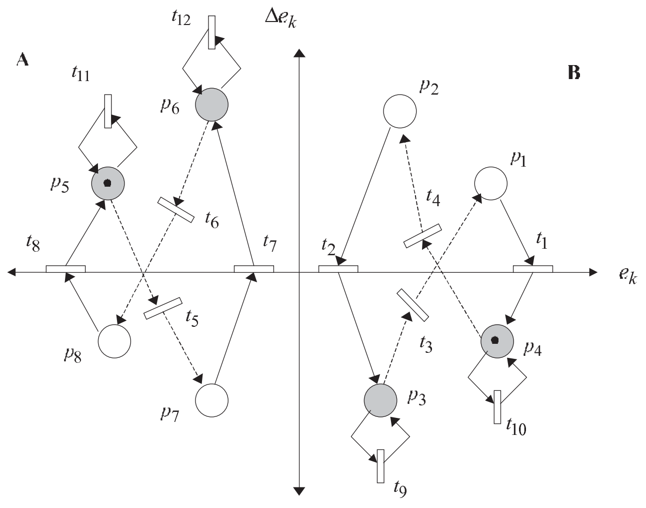

Figure 8: Finite state machine describing the reactive controller

Theorem The proposed automaton assures the convergence of the robotic vehicle to the desirable final position $\phi_d = 0^o$.

Proof: Using the results of Lemma 3, it becomes clear that the vehicle will always end to one of the states p_3, p_4, p_5 or p_6, which belong to the semi-plane $e_k \Delta e_k < 0$. As long as the system remains on semi-plane $e\dot{e}$ it is assured that the Lyapunov function $V = \frac{1}{2}e_k^2$ is negative definite. Thus $lim_{k\to\infty} e_k = 0$ and the system converges to the desirable angle $\phi_d = 0^o$.

6.4 A model of computation for the parallel parking controller

The clocked synchronous model of computation is directly applicable to the previously described Petri Net model (process) of the parking controller. Time is divided into time slots. The events received as inputs by the finite state machine at time slot k are: the sign of the velocity v_k, the sign of the orientation error $e_k = \phi_k - \phi_d$, the sign of the first derivative of the orientation error $\dot{e}_k$, and the previous control action $\alpha_i^k,\ i = 1,2$ which can be an increase or a decrease of the control signal. The process performs internal state transition and the output is generated (the process reacts to the received input). The output consists of the new sign of the velocity v_{k+1}, the new value of the orientation error e_{k+1}, the new value of the first derivative of the orientation error $\dot{e}_{k+1}$, and the new value of the of the control signal $\alpha_i^{k+1},\ i = 1,2$. The computation takes place into one time slot (evaluation cycle), so between the input and the output of the automaton there is a time delay denoted by Δ. Each time slot corresponds to only one input set and only to one output set. The parking process is monotonous: the more input it receives from the orientation sensors the greater the change of the vehicle's orientation becomes. Absent input and output events can also appear during a time slot. The parking process is also continuous: after certain evaluation cycles (time slots) the orientation of the vehicle will have practically converged to the desirable value $\phi_d = 0$, therefore the infinite output signal can be approximated by a finite one. The parking process is also sequential. If the velocity or the orientation sensor stops to give input to the process then the computation of the procedure will be blocked. If the parking controller (automaton) is connected to a different process, for instance an automaton that controls the vehicles velocity then a process network is generated. Several tasks in the operation of the intelligent vehicle can be modeled as processes and thus the overall functioning will be described by a process of higher complexity.

The use of the perfectly synchronous model of computation for the parking controller is not convenient since receiving as input and giving as output the same type of events (velocity v, the orientation error e, first derivative of the orientation error $\dot{e}$, and new value of the of the control signal $\alpha_i,\ i = 1,2$) would result in a recursive equation that would constrain the values of all the involved events.

Using the timed model of computation to describe the parking process means that the granularity of time is much finer and events occur in every nanosecond or picosecond rather than every clock cycle. There is no obvious reason for adopting the timed model of computation since measurements from the vehicle's velocity and orientation sensors can be obtained at a sampling period that is not of the nanosecond order. There is also no need to used a

MoC which considers multiple events in the same evaluation cycle since the evaluation cycle has to produce one signle event which is the vehicle's control signal. The same comments holds for the untimed MoC.

7 Conclusions

The paper has studied the suitability of different models of computation for the design of a reactive controller that enables mobile robots to perform the parallel parking task. The paper's results can be generalized in the design of reactive controllers for several tasks performed by intelligent robots and intelligent vehicles. Reactive systems receive inputs, react to them by computing outputs and wait for the next inputs to arrive. Reactive systems correspond to finite state machines which in turn can be represented with the use of Petri nets. Three different models of computation have been examined: the untimed, the synchronous and the timed MoC.

First, the *untimed model* of computation was analyzed. The parking controller (process) was modeled as state machine in which events do not carry any time information but preserve their order of emission. Second the *synchronous model* of computation was considered. This can be based on partition of time either into time slots or into clock cycles. The perfectly synchronous model assumes that no time advances during the evaluation of a process. Consequently, the results of a computation on input values is already available in the same cycle. The clocked-synchronous model assumes that every simulation of a process takes one cycle. Hence the reaction of a process to an input becomes effective in the next cycle. Third, the *timed model* of computation was examined which assigns a time stamp to each value communicated between processes. This allows to model time-related issues in great detail, but it complicates the model and the task of analysis and simulation.

It was shown that the synchronous model of computation is more appropriate for describing the controller for the parallel parking task. Results on the efficiency of the proposed control algorithm have been obtained with the use of a 4-wheel mobile robot (Coroware CB-WA). This research work will be continued with the detailed presentation of the experimental implementation of the reactive parking controller on the 4-wheel robot.

8 References

[1] Passino K. and Burgess K. , Stability analysis of discrete event systems. *John Wiley*, 1998.
[2] Luzeaux D. and Zavidovique B., Rule-based incremental controllers: an open-door to learning behaviors. In: Proc. Japan-USA Symposium on Flexible Automation, San Francisco, USA, 1992.
[3] Rigatos G.G., Tzafestas S.G. and Evangelidis G.J., Reactive Parking Control of a non-holonomic vehicle via a fuzzy learning automaton. *IEE Proc. on Control Theory and Applications*, 148 (2001), 169–180.
[4] Rigatos G.G., Fuzzy Stochastic Automata for Intelligent Vehicle Control. *IEEE Transactions on Industrial Electronics*, 50 (2003), 76–79.
[5] Jantsch A., Modelling embedded systems and systems on chip: concurrency and time models of computation, *Morgan Kaufmann Publishers*, 2004.
[6] Kozen D.C., Automata and Computability. *Springer*, 1997.
[7] Morderson J.N. and Malik D.S., Fuzzy Automata and Languages. *Chapman & Hall*, 2002.
[8] Meystel A.M. and Albus J.S., Intelligent Systems: Architecture, Design and Control, *John Wiley*, 2002.
[9] Lin F. and Ying H., Modelling and control of fuzzy discrete event systems. *IEEE Transactions on Systems, Man and Cybernetics- Part B: Cybernetics*, 32 (2002), 408–415.
[10] Qiu D., Characterizations of fuzzy finite automata. Fuzzy Sets and Systems, *Elsevier*, 141 (2004), 391–414.
[11] Qiu D., Supervisory control of fuzzy discrete event systems, *IEEE Transactions on Systems, Man and Cybernetics- Part B: Cybernetics*, 35 (2005), 72–88.
[12] Tzafestas S.G. and Rigatos G.G., Stability analysis of an adaptive fuzzy control system using Petri Nets and learning automata. Mathematics and Computers in Simulation, *Elsevier*, 51 (2000), 315–341.
[13] Benveniste A. and Berry G., The synchronous approach to reactive and real-time systems, *Proceedings of the IEEE*, 79 (1991), 1270–1282.
[14] Benveniste A., Caspi P., Edwards S.A., Halbwachs N., le Guernic P. and de Simone R., The Synchronous Languages 12 years later. *Proceedings of the IEEE*, 91 (2003), 64–83.