# Embedded Software Engineering
## WS 2004/2005

LVA: Prof. Christoph Kirsch

Students:
Richard Bauer
Mohamed El Khattaf
Christine Grammerstätter

# contents

- introduction
- project description
- I/O
- E machine
- particularities
- demonstration

# project

- Implementation of an E-Machine on a RCX - brick
- running an e-code application with two modes according to a given giotto-model
- Roboter - behavior: acts autonomously , avoids hitting obstacles.

# I/O

- input - sensor
- output - actuator
- task - driver

# E-Machine

implemented E-Code instructions

```
#define E_RETURN        0
#define E_CALL          1
#define E_RELEASE       2
#define E_FUTURE        3
#define E_FUTURE_REL    4
#define E_IF            5
#define E_JUMP          6
#define E_IF_REL        7
#define E_JUMP_REL      8
#define E_CANCEL        9
```

# E-Code-structure

```
struct eCommand{
    unsigned int command;
    unsigned int method;
    unsigned int time;
    unsigned int e_address;
};
```

# task-handling

**problem: task-termination**

- solution: semaphore-array

```
static tid_t task_sem[] =
{
    FREE_TASK, FREE_TASK,...FREE_TASK
};
```

- used for exception-handling (E_RELEASE / E_CANCEL)

# time-trigger

```
struct trigger                          // trigger function returns
{                                       // boolean if triggered
    unsigned int  trigger;              // pointer to an adresss in
                                        // the E code
    unsigned int  e_address;            // system start time in ms
    unsigned long trigger_start;        // trigger activation-time
                                        // in ms
    unsigned long trigger_delta;        // time after which trigger
                                        // becomes true
};
```

# direction- & sensor types

- enum dir_t  {turn_left, turn_right, forward, backward, stop, idle};
- enum sensor_state  {hit_front_center, hit_front_left, hit_front_right,hit_back_center, hit_back_left, hit_back_right, NON};

# driving-command

```
struct driving_command {
    int priority;
    enum dir_t direction;
    boolean changed;
    int speed;
    signed long duration;
};
```

# ports

- input ports
- output ports
- task ports

# tasks

- direction-change
- obstacle-handling
- motor-control
- show_standby_mode
- show_running_mode

# Direction-Change

**Input**: nothing
**Output**: DC_OUT

```
long int left_right = ( random() % 4 );
// select new direction by random


if ( left_right == 0 || 1 || 2 || 3 )      {
    DC_OUT.priority     = 2 ;
    DC_OUT.direction    = turn_left/-right/back/forward;
    DC_OUT.speed        = MAX_SPEED ;
    DC_OUT.duration     = 1000 + ( random () % 6 ) ;
    DC_OUT.changed      = TRUE ;
}
```

# obstacle-handling

**Input:** S1, S2, S3
**Output:** OH_OUT

- check_sensors determines if we hit an obstacle
- case hit_front_right...:

```
OH_OUT.priority     = 4;
OH_OUT.direction    = turn_left...;
OH_OUT.speed        = MAX_SPEED ;
OH_OUT.duration     = 500 ;
OH_OUT.changed      = TRUE;
break;
```

# motor-control

**Input:** MC_IN_DC, MC_IN_OH
**Output:** M1, M2, SPEED

- if ( MC_IN_OH.changed == TRUE ) {

  MC_IN_OH.changed = FALSE ;

  if ( MC_IN_OH.priority > active_command.priority || active_command.duration <= 0 ) {

  active_command = MC_IN_OH ;

  }

  }
- if ( active_command.duration > 0 ) {...}

# motor-control

```
if ( active_command.duration > 0  ) {
        switch ( active_command.direction ) {
                case forward:
                M1 = fwd ;

                M2 = fwd ;

                break;...}
    SPEED = active_command.speed ;
    if (active_command.duration > Period_Task_MC)
        active_command.duration =
        active_command.duration - Period_Task_MC ;
    else
        active_command.duration = 0;
} else { M1=off; M2=off; }}
```

# E code (old version)

```
{E_CALL,      driver_sensor,          0,      0 }   // 0 standby-mode
{E_IF_REL,    mode_switch_1,          4,      4 }   // 1 test for mode-switch
{E_RELEASE,   sense_less,             0,      0 }   // 2
{E_FUTURE,    time_trigger,         500,    a_1 }   // 3
{E_RETURN,    0,                      0,      0 }   // 4

{E_CALL,      driver_actuator,        0,      0 }   // 5 running-mode
{E_CALL,      driver_sensor,          0,      0 }   // 6
{E_CALL,      driver_OH2MC,           0,      0 }   // 7
{E_CALL,      driver_DC2MC,           0,      0 }   // 8

{E_RELEASE,   dir_change,             0,      0 }   // 9
{E_RELEASE,   motor_control,          0,      0 }   // 10
{E_RELEASE,   obstacle_handling,      0,      0 }   // 11
{E_FUTURE,    time_trigger,Period_Task_MC,a_1}   //12
{E_RETURN,    0,                      0,      0}   // 13
```

# E code - new version(1)

```
static const struct eCommand eCode[] = {

                                S T A N D B Y - M O D E

M O D E - Initialisation

a_1:@0 { E_CALL,      driver_m_standby_init,      0,          0 }

a_2:@1 { E_CALL,      driver_sensor,              0,          0 }

  2 Mode-switch-test: jmp into active-mode @a_3
        { E_IF_REL,   mode_switch_1,              0,       r_a3 }

  3 display spent time in standby-mode in seconds
        { E_RELEASE, show_standby_mode,           0, a_c_ssm }

  4 looping to a_2

        { E_FUTURE,  time_trigger, Period_standby,       a_2 }

        { E_RETURN,  0,                           0,          0 }
```

# E code (2)

R U N N I N G - M O D E

M O D E - Initialisation -

a_3:@6 { E_CALL, driver_m_running_init, 0, 0 }

7 delay-unit of 1000 ms for hardware-initilisation

{ E_FUTURE, time_trigger, 1000, a_4 }

{ E_RETURN, 0, 0, 0 }

a_4:@9 this is start of running-mode and eblock nr 1 of this mode

9 check for mode-switch back to standby-mode

{ E_IF, mode_switch_2, 0, a_1 }

# E code (3)

D R I V E R - Calls: the drivers used in e-block 1

```
    { E_CALL,     driver_actuator,     0,     0 }
    { E_CALL,     driver_sensor,       0,     0 }
    { E_CALL,     driver_OH2MC,        0,     0 }
    { E_CALL,     driver_DC2MC,        0,     0 }
```

T H E  T A S K S started in e-block 1

processing of output of obstacle-handling and direction-change writing to motor-ports M1, M2

```
    { E_RELEASE, motor_control,        0,a_c_motor}
```

checks if the robot hit any obstacle

```
    { E_RELEASE, obstacle_handling,    0,   a_c_oh }
```

randomly choosing a newdirection

```
    { E_RELEASE, dir_change,           0,   a_c_dc }
```

show on display a blinking ´-´, indicating that we are in runing mode

```
    { E_RELEASE, show_running_mode, 0, a_c_srm }
```

# E code (4)

```
set time-trigger, loop to e-block 2 of running mode
    { E_FUTURE_REL, time_trigger, Period_running,        2 }
    { E_RETURN,     0,                    0,             0 }

DRIVER-CALLS in e-block 2,3,4,5 of running-mode
    {E_CALL,        driver_actuator,      0,             0 }
    {E_CALL,        driver_sensor,        0,             0 }
    {E_CALL,        driver_OH2MC,         0,             0 }

THE TASKS of block 2,3,4,5
    { E_RELEASE,    motor_control,        0,a_c_motor }
    { E_RELEASE,    obstacle_handling,    0,    a_c_oh }

THE FUTURE instructin of eblock 2,3,4
    { E_FUTURE_REL, time_trigger, Period_running,        2 }
    { E_RETURN,     0,                    0,             0 }

FUTURE-instructino of block 5: jumping back to first ebock of running-mode
    { E_FUTURE,     time_trigger, Period_running,      a_4 }
    { E_RETURN,     0,                    0,             0 }
```

# E code (5)

E X E P T I O N - H A N D L I N G (simplified implementation)

1. stand-by-mode:

    kill task
    { E_CANCEL, show_standby_mode, 0, 0 }

    restart mode
    { E_CALL, clean_up_sstandbym, 0, 0 }
    { E_JUMP, 0, 0,a_1 }

2. running-mode:
    for each task in running mode do the following

    { E_CANCEL, <task>, 0, 0 }
    { E_CALL, <call clean-up-funcitons> 0, 0 }
    { E_JUMP, 0, 0,a_4 }

}

# Wrapper

```
wrapper[motor_control] =
   (unsigned int) &Task_motor_control;

wrapper[obstacle_handling] =
   (unsigned int) &Task_obstacle_handling;

wrapper[dir_change] =
   (unsigned int) &Task_direction_change;

...
```

# task-control

```
int schedule(int argv, char **argc) {
    // argv is a index to the wrapper array
    void (*ptr)(void) = (void *) wrapper[ argv ];

    while (task_lock) yield();
    // task_sem contains either task_id if running or
    // -1 if not
    ptr();
    task_sem[ argv ] = FREE_TASK;
    return TRUE;
}
```

# E Machine

```
e_PC = 0;
start_time    = get_system_up_time();
msleep_wakeup = start_time;
error = invoke( e_PC );
if (error<0) err(error);
   while ( !shutdown_requested () )  {   // test if user presses on-off key
   for ( i=0; i<=MAX_TRIGGER; i++ ) {// check all triggers
     if ( trigger_check( i ) == TRUE ) {   // delete trigger from trigger queue
        triggers[i].trigger = FREE_TRIGGER;
        e_PC = triggers[i].e_address;      // set program counter to e_address
        error = invoke( e_PC );            // execute a block of e-code
        if (error<0) err(error);
     }} // for & if
```

# E Machine

```
// compute time for the E machine to wake up next
time2wait =  (msleep_wakeup - get_system_up_time());
        if ( time2wait  > time_eps ) {       // to be more accurate
           msleep( time2wait -time_eps ); // time to sleep is reduced
           // during msleep-operation, the OS will execute all released tasks
        }  // for
    }  // while
    return 0;
```