```
/**
* FILENAME :        iWA_RC_Slave_LC_V3
*
* DESCRIPTION :
*       Software to Control E-Rollator from iHomeLab
*       Implemented are:
*         -Loop Controller PID
*         -Turn off on button press
*         -Time based automatic turn off (WatchDog)
*         -battery low turn off
*         -battery surveillance (ADC-Usage)
*
*       Software is partially copied from existing
*       code created by an unknown author at iHomeLab. This software extends
*       the functionality of the E-Rollator.
*
* FUNCTIONS :
*       void setup(void)
*       void loop(void)
*       void setupCan(void)
*       void readCan(void)
*       void readJoystick(void)
*       void turnOffWatch(void)
*       void batteryStatus(void)
*       void sendSpeedToWheel(int side, int wheelSpeed)
*       void sendCurrentToWheel(int side, int wheelSpeed)
*       void tickISR()
*
* NOTES :
*       System tick can be observed on Pin 8
*       content of this file belongs to iHomeLab
*
* AUTHOR :    Fabian Niederberger        START DATE :    27.11.2016
*
* CHANGES :
* VERSION DATE      WHO     DETAIL
* 1.0     17.11.16  FN      Create file
* 2.0     02.12.16  FN      Joystick by Wire
* 3.0     10.12.16  FN      Delete Bluetooth and clean up
*
*/

#include <TimerOne.h>
#include <mcp_can.h>
#include <mcp_can_dfs.h>
#include <SoftwareSerial.h>
#include <SPI.h>

/*****************************************//**
 *  Makros
 *****************************************/
#define INIT_PIDVAR(X) PIDVar X = {.w_k = 0, .e_k = 0, .u_k = 0, .u_p_k =
  0, .u_i_k = 0, .u_i_k_1 = 0, .u_d_k = 0, .u_d_k_1 = 0, .v_k = 0, .u_arw_k
```

```
      = 0, .u_arw_k_1 = 0, .y_k = 0, .y_k_1 = 0}
53
54  /*****************************************//**
55   *  Parameters & Variables
56   *****************************************/
57  //System declarations
58  #define TICK        40          //System Tick in ms
59  #define BAUDRATE    115200      //Hardware Serial Baudrate (con. to PC)
60
61  //PIN declarations
62  #define BATTERY     A0          //battery balance on Analog 0
63  #define PWRON       3           //ON/OFF Transistor on Digital 3
64  #define PWROFF      4           //Power OFF Switch on Digital 4
65  #define RxD         7           //Software Rx on Digital 7
66  #define TxD         6           //Software Tx on Digital 6
67  #define LED         8           //Status LED Digital 8
68  #define CSCAN       10          //Chip select for CAN Shield Digital 10
69  #define JX          A2          //Joystick X value
70  #define JY          A3          //Joystick Y Value
71
72  //developers controls
73  #define CURRENT             1   //if 1 Motor is controlled by setting
      current
74                                  //if 0 Motor is controlled by setting speed
75  #define DEBUG_ENABLED       0   //debugging via serial port
76  #define DEBUG_PID           0   //to debug PID
77  #define PID_ENABLED         1   //PID will be active
78  #define ROLLATOR_UNPLUGGED  0   //If Rollator is unplugged and tests are
      executed
79  #define MEASURE_TICK        1   //if 1: LED -> TICK, if 0: LED -> Main Loop
80
81  //Declarations
82  MCP_CAN CAN(CSCAN);             //CAN Shield
83
84  //PID Parameters
85  static const double Kp      = 40;       //proportional gain
86  static const double T       = (double) TICK;
87  static const double Ti      = 18;       //i factor
88  static const double Td      = 1;        //d factor
89  static const double Tr      = Ti;       //anti-reset windup (ARW) factor
      (defined by Th. Prud'homme)
90  static const double N       = 1/T;   //Filter for D-Part
91  static const int u_min      = -2000;    //actuator min, saturation of
      current
92  static const int u_max      = 2000;     //actuator max, saturation of
      current
93  static const double ad      = Td / (Td + N*T/1000); //numerator D
94  static const double bd      = Kp * Td * N / (Td + N*T/1000); //denominator D
95  static const double ai      = Kp * T / Ti; //numerator I
96  static const double ar      = T / Tr;  //numerator ARW
97  static const double factor  = 0.20;     //factor to adjust speed to current
      control ~5.2
98
```

```
 99  typedef struct {
100    double w_k;           //target, given by joystick
101    double e_k;           //target/actual error
102    double u_k;           //actuator at time k -> value for motor
103    double u_p_k;         //proportional part
104    double u_i_k;         //integral part
105    double u_i_k_1;       //actuators value at time k-1
106    double u_d_k;         //deviation part
107    double u_d_k_1;       //deviation part at time k-1
108    double v_k;           //actuator before ARW
109    double u_arw_k;       //ARW at time k
110    double u_arw_k_1;     //error anti-reset windup at time k-1
111    double y_k;           //speed
112    double y_k_1;         //speed at time k-1
113  } PIDVar;
114
115  //Motor controls
116  static const int setLeft        = 266;      //control motor Left
117  static const int setRight       = 298;
118  static const int getSpeedLeft   = 256;      //read motor Left
119  static const int getSpeedRight  = 288;
120  static const int getOdoLeft     = 257;      //odo ticks left wheel
121  static const int getOdoRight    = 289;      //odo ticks right wheel
122  static PIDVar left;                         //PID variables for left motor
123  static PIDVar right;
124  static unsigned long pidCount   = 0UL;      //to Debug PID
125
126  //Battery Surveillance Parameters
127  static const float turnOffV     = 3.4;      //device turns off at this
       battery cell voltage
128  static const int batCheck = 120 * (1000/TICK);  //battery checking time in s
129  static const float analogRef    = 5.0;      //ADC reference voltage
130  static const int adcRes         = 1024;     //ADC Resolution
131  static float cellVoltage        = 0;        //battery cell voltage
132  static int batCount             = 0;
133
134  //Automatic Turn Off Parameters
135  static const unsigned long turnOfftime = 600UL * (1000 / TICK); //turn off
       value in seconds
136  static unsigned long watchCounter = 0UL;                        //counter
       turn off watchdog
137
138  /**
139   * an integer 16 bit is splitted into two 8 bit characters
140   * <p>
141   * this union is called before sending data to CAN shield
142   */
143  union {
144    int integer;
145    unsigned char byte[2];
146  } int2byte;
147
148  /*****************************************//**
```

```
149   *  System Setup
150   *********************************************/
151  /**
152   * Initializes Board and Parameters before entering the
153   * main loop
154   * <p>
155   * This function runs only once on startup.
156   */
157  void setup()
158  {
159    unsigned long tOne = 0;
160    //setup power Management
161    pinMode(PWROFF,INPUT);
162    pinMode(PWRON, OUTPUT);
163    //turn on Rollator
164    digitalWrite(PWRON, HIGH);
165
166    //initialize Timer
167    tOne = TICK * 1000UL;
168    Timer1.initialize(tOne);                  //Timer time in set in uSeconds
169    Timer1.attachInterrupt(tickISR);
170
171    //setup serial connections
172    if(DEBUG_ENABLED || DEBUG_PID)            //serial connection to PC
173      Serial.begin(BAUDRATE);                 //Hardware UART to PC
174    pinMode(RxD, INPUT);                      //Software UART to BT
175    pinMode(TxD, OUTPUT);
176
177    //setup status LED
178    pinMode(LED, OUTPUT);
179
180    if(!ROLLATOR_UNPLUGGED)
181    {
182      //setup CAN shield
183      setupCan();
184    }
185
186    //Init Variables for PID
187    INIT_PIDVAR(left);
188    INIT_PIDVAR(right);
189
190      if(DEBUG_PID)
191      {
192          Serial.println("ad = "+String(ad)+"\tbd = "+String(bd)+"\tai =
              "+String(ai)+"\tar = " + String(ar));
193          Serial.println("cnt,w_k,e_k,u_k,u_i_k,u_d_k,y_k,u_arw_k" );
194          Serial.println(String(pidCount)+","+String(left.w_k)+","+String
              (left.e_k)+","+String(left.u_k)+","+String(left.u_i_k)+","+String
              (left.u_d_k)+","+String(left.y_k)+","+String(left.u_arw_k));
195      }
196  }
197
```

```
198 /*****************************************//**
199  *  Main
200  *****************************************/
201 /**
202  * All tasks will be done sequentially. Flags that
203  * have been set in the ISRs will be checked and executed
204  * <p>
205  * This function runs endlessly after setup
206  */
207 void loop()
208 {
209     if(!MEASURE_TICK)
210         digitalWrite(LED, digitalRead(LED)^1);
211 
212     //check button 2, when pressed, turn off
213     if(digitalRead(PWROFF) == 0)
214     {
215         if(DEBUG_ENABLED)
216             Serial.println("Switch 2 pressed, Shut-Down Rollator");
217         digitalWrite(PWRON, LOW);
218     }
219 
220     //check battery after defined time
221     if(batCount >= batCheck)
222     {
223         batCount = 0;
224         batteryStatus();
225     }
226 
227     if (!ROLLATOR_UNPLUGGED)
228     {
229         //read joystick for target new target values
230         readJoystick();
231         //read actual values
232         readCAN();
233 
234         if(CURRENT)
235         {
236             //send current values to motors
237             sendCurrentToWheel(setLeft, ((int) left.u_k));
238             sendCurrentToWheel(setRight, ((int) right.u_k));
239             if(DEBUG_ENABLED)
240                 Serial.println("Current: " + String(left.u_k) + " / " +      ↵
                      String(right.u_k));
241         }
242         else
243         {
244             //send Speed values to motors
245             sendSpeedToWheel(setLeft, ((int) left.u_k));
246             sendSpeedToWheel(setRight, ((int) right.u_k));
247             if(DEBUG_ENABLED)
248                 Serial.println("Speed: " + String(left.u_k) + " / " + String ↵
                      (right.u_k));
```

```
249         }
250   }
251 }
252
253 /*****************************************//**
254  *  Setup Peripheral Components
255  *****************************************/
256
257 /**
258  * Sends initializing commands to can shield to set it
259  * up for communication with motor controller from the
260  * wheels, left and right.
261  * <p>
262  * This function is called once on setup.
263  */
264 void setupCan()
265 {
266 START_SETUP:
267   if(CAN_OK == CAN.begin(CAN_1000KBPS))
268   {
269     if(DEBUG_ENABLED)
270         Serial.println("CAN Init ok");
271   }
272   else
273   {
274     if(DEBUG_ENABLED)
275         Serial.println("Can't init CAN\nTrying again...");
276     delay(100);
277     goto START_SETUP;
278   }
279 }
280
281 /*****************************************//**
282  *  Functions
283  *****************************************/
284 /**
285  * Reads Joystick values for target Speed
286  * <p>
287  * This function runs each loop.
288  */
289 void readJoystick()
290 {
291     int x = 0;
292     int y = 0;
293
294     x = analogRead(JX);
295     y = analogRead(JY);
296     x -= 512;
297     y -= 512;
298
299     //noise hyst.
300     if (abs(x) < 20)
301     x = 0;
```

```
302     else if (x > 500)
303     x = 500;
304     else if (x < -500)
305         x = -500;
306
307     if (abs(y) < 20)
308     y = 0;
309     else if (y > 500)
310     y = 500;
311     else if (y < -500)
312     y = -500;
313
314     //calculate motor values
315     if(y >= 0){
316         right.w_k = factor*(-x + y);
317         left.w_k = factor*(x + y);
318     }
319     else{
320         right.w_k = factor*(x + y);
321         left.w_k = factor*(-x + y);
322     }
323
324     if(DEBUG_ENABLED)
325         Serial.println("Joystick:\t" + String(x) + "/"+ String(y) + "\t" +
            String(left.w_k) + "/" + String(right.w_k));
326 }
327
328 /**
329  * reads speed information from CAN-Bus
330  * <p>
331  * this function runs each loop
332  */
333 void readCAN()
334 {
335     while(CAN_MSGAVAIL == CAN.checkReceive())                // check if
        data incoming
336     {
337         unsigned char len = 0;
338         unsigned char buf[8];
339         unsigned char val[2] = {0, 0};
340         int sigVal[2] = {0, 0};
341         CAN.readMsgBuf(&len, buf);                           // read data,
            len: data length, buf: data buf
342         unsigned int canId = CAN.getCanId();
343         //CAN-Frame is Speed/Accel. Calculate usable value
344         if((canId == 256) || (canId == 288)){                            //
            show can ID
345             for(int i = 0; i<2; i++)                          // read only
                first 2 bytes (speed)
346             {
347                 val[i] = 256 - buf[i];
348             }
349         if(DEBUG_ENABLED)
```

```
350             Serial.println(String(canId) + "," + String(val[0]) + "," +
                  String(val[1]));
351
352         //prepare forward values for controlling
353         if((val[0] > 0) && (val[0] < 128))
354         {
355             sigVal[0] = val[0] - 1;
356             sigVal[1] = val[1];
357         }
358
359         //prepare reverse values for controlling
360         else if((val[0] >= 128))
361         {
362             sigVal[0] = (int) val[0];
363             sigVal[0] = sigVal[0] - 255;
364             sigVal[1] = (int) val[1];
365             sigVal[1] = (sigVal[1] - 255);
366         }
367
368         //additional reverse values
369         else if(val[1] != 0)
370         {
371             sigVal[0] = (int) val[0];
372             sigVal[0] = sigVal[0];
373             sigVal[1] = val[1];
374             sigVal[1] = (int) sigVal[1] - 255;
375         }
376
377         if(canId == 256)
378             left.y_k = sigVal[0] * 256 + sigVal[1];
379         if(canId == 288)
380             right.y_k = sigVal[0] * 256 + sigVal[1];
381
382         if (DEBUG_ENABLED)
383             Serial.println("y_k: " + String(canId) + ",\t" + String(sigVal
                  [1]) + ";");
384         }
385   }
386 }
387
388
389 /**
390  * increments a counter. If limit is reached Rollator will
391  * be turned off to prevent battery discharge
392  * <p>
393  * This Function is called on each system tick
394  */
395 void turnOffWatch()
396 {
397   watchCounter++;
398   if(watchCounter >= turnOfftime)
399   {
400     digitalWrite(PWRON, LOW);
```

```
401     }
402   }
403
404   /**
405    * battery cell voltage will be calculated and on low
406    * voltage Rollator will be turned off
407    * <p>
408    * this functions will be called after defined time
409    */
410   void batteryStatus()
411   {
412     cellVoltage = analogRead(BATTERY);
413     cellVoltage = (cellVoltage * analogRef) / adcRes;     //Voltage at Analog
        Port
414     cellVoltage = (cellVoltage * 242) / 22;               //Resistor Divider
        R1 + R2 = 242k, R2 = 22k
415     if (DEBUG_ENABLED)
416       Serial.println("Battery:\t" + String(cellVoltage) + " V");
417     cellVoltage = cellVoltage/6;                          //calc cell voltage
418     //on low battery voltage, turn off Rollator
419     if(cellVoltage <= turnOffV)
420       digitalWrite(PWRON, LOW);
421   }
422
423   /*****************************************//**
424    *  Motor Control Functions
425    *****************************************/
426   /**
427    * send speed (RPM) infromation to wheel
428    * <p>
429    * this function is called in every loop wether the information
430    * has changed or not. Motor needs periodically instructions othwerwise
431    * turns off
432    *
433    * @param  side  which Motor will be controlled
434    * @param  wheelSpeed RPM information
435    *
436    */
437   void sendSpeedToWheel(int side, int wheelSpeed)
438   {
439     unsigned char sendBuf[5];
440     int2byte.integer = wheelSpeed;
441     sendBuf[0] = 1;
442     sendBuf[1] = int2byte.byte[0];  //arduino int is 2 bytes (16 bit)
443     sendBuf[2] = int2byte.byte[1];
444     sendBuf[3] = 0;
445     sendBuf[4] = 0;
446
447     CAN.sendMsgBuf(side, 0, 5, sendBuf);
448   }
449
450   /**
451    * send current (mA) infromation to wheel
```

```
452   * <p>
453   * this function is called in every loop wether the information
454   * has changed or not. Motor needs periodically instructions othwerwise
455   * turns off
456   *
457   * @param  side  which Motor will be controlled
458   * @param  wheelSpeed RPM information
459   *
460   */
461  void sendCurrentToWheel(int side, int wheelSpeed)
462  {
463    unsigned char sendBuf[5];
464    int wheelCurrent = -2*wheelSpeed;
465
466    int2byte.integer = wheelCurrent/3;
467    sendBuf[0] = 2;
468    sendBuf[1] = 0;
469    sendBuf[2] = 0;
470    sendBuf[3] = int2byte.byte[1];
471    sendBuf[4] = int2byte.byte[0];
472
473    CAN.sendMsgBuf(side, 0, 5, sendBuf);
474  }
475
476  /*********************************************//**
477   *  Interrupt Service Routines
478   ***********************************************/
479   /**
480   * Time critical elements are "registered" here and
481   * flags will be set to signal a execution requirement
482   * for those functions
483   * the Tick toggles LED to measure accuracy from
484   * outside
485   * <p>
486   * Timer interrupt service routine will be called on
487   * each timer overflow
488   */
489   void tickISR()
490  {
491      if(MEASURE_TICK)
492          digitalWrite(LED, digitalRead(LED)^1);
493
494      turnOffWatch();
495
496      //PID Controller turned off
497      if (!PID_ENABLED)
498      {
499          left.u_k = left.w_k;
500          right.u_k = right.w_k;
501      }
502
503      if(!ROLLATOR_UNPLUGGED && PID_ENABLED)
504      {
```

```
505         //calculate new error for PID
506         left.e_k = left.w_k - left.y_k;
507         right.e_k = right.w_k - right.y_k;
508
509         //P
510         left.u_p_k = Kp * left.e_k;
511         right.u_p_k = Kp * right.e_k;
512
513         //I
514         left.u_i_k = left.u_i_k_1 + (ai * left.e_k)/1000;       //Tick
              correction
515         right.u_i_k = right.u_i_k_1 + (ai * right.e_k)/1000;
516
517         //D with filter and only on output ,not on error
518         left.u_d_k = ad * left.u_d_k_1 + bd * (-left.y_k + left.y_k_1);
519         left.u_d_k = ad * left.u_d_k_1 + bd * (-right.y_k + right.y_k_1);
520
521         //actuator before ARW
522         left.v_k    = left.u_p_k + left.u_i_k + left.u_d_k + left.u_arw_k_1;
523         right.v_k   = right.u_p_k + right.u_i_k + right.u_d_k +
              right.u_arw_k_1;
524
525         //Saturation
526         if (left.v_k > u_max)
527             left.u_k = u_max;
528         else if (left.v_k < u_min)
529             left.u_k = u_min;
530         else
531             left.u_k = left.v_k;
532
533         if (right.v_k > u_max)
534             right.u_k = u_max;
535         else if (right.v_k < u_min)
536             right.u_k = u_min;
537         else
538             right.u_k = right.v_k;
539
540         //ARW
541         left.u_arw_k = left.u_arw_k_1 + (ar * (left.u_k - left.v_k))/1000;
                //Tick correction
542         right.u_arw_k = right.u_arw_k_1 + (ar * (right.u_k -
              right.v_k))/1000;
543
544         if(DEBUG_PID)
545         {
546             pidCount++;
547             Serial.println(String(pidCount)+","+String(left.w_k)+","+String
                (left.e_k)+","+String(left.u_k)+","+String(left.u_i_k)
                +","+String(left.u_d_k)+","+String(left.y_k)+","+String
                (left.u_arw_k_1));
548         }
549
550
```

```
551          //Store variables
552          left.u_i_k_1 = left.u_i_k;
553          right.u_i_k_1 = right.u_i_k;
554          left.u_arw_k_1 = left.u_arw_k;
555          right.u_arw_k_1 = right.u_arw_k;
556          left.y_k_1 = left.y_k;
557          right.y_k_1 = right.y_k;
558          left.u_d_k_1 = left.u_d_k;
559          right.u_d_k_1 = right.u_d_k;
560
561          //I-Reset
562          if((abs(left.y_k) < 10) && (abs(left.w_k) <= 1))
563          {
564          left.u_i_k_1 = 0;
565          left.u_arw_k_1 = 0;
566          }
567          if((abs(right.y_k) < 10) && (abs(right.w_k) <= 1))
568          {
569              right.u_i_k_1 = 0;
570              right.u_arw_k_1 = 0;
571          }
572
573
574      }
575
576      if(!ROLLATOR_UNPLUGGED)
577          batCount++;
578  }
```