```c
/* ************************************************************** */
/* File name:        pid.c                                        */
/* File description: This file has a couple of useful functions to   */
/*                   control the implemented PID controller         */
/* Author name:      julioalvesMS, lagoAF, rBacurau                 */
/* Creation date:    21jun2018                                      */
/* Revision date:    31jul2020                                      */
/* ************************************************************** */

#include "pid.h"

pid_data_type pidConfig;

/* ********************************************** */
/* Method name:        pid_init                   */
/* Method description: Initialize the PID controller*/
/* Input params:       n/a                        */
/* Output params:      n/a                        */
/* ********************************************** */
void pid_init(void)
{
 pidConfig.fKp = 0.0;
 pidConfig.fKd = 0.0;
 pidConfig.fKi = 0.0;
 pidConfig.fError_previous = 0;
 pidConfig.fError_sum = 0.0;
}


/* ************************************************ */
/* Method name:        pid_setKp                    */
/* Method description: Set a new value for the PID    */
/*                     proportional constant          */
/* Input params:       fKp: New value                 */
/* Output params:      n/a                          */
/* ************************************************ */
void pid_setKp(float fKp)
{
 pidConfig.fKp = fKp;
}


/* ************************************************ */
/* Method name:        pid_getKp                    */
/* Method description: Get the value from the PID     */
/*                     proportional constant          */
/* Input params:       n/a                          */
/* Output params:      float: Value                   */
/* ************************************************ */
float pid_getKp(void)
{
 return pidConfig.fKp;
}


/* ************************************************ */
/* Method name:        pid_setKi                    */
/* Method description: Set a new value for the PID    */
/*                     integrative constant           */
/* Input params:       fKi: New value                 */
/* Output params:      n/a                          */
/* ************************************************ */
void pid_setKi(float fKi)
{
 pidConfig.fKi = fKi;
}
```

```
/* ********************************************** */
/* Method name:        pid_getKi                  */
/* Method description: Get the value from the PID */
/*                     integrative constant       */
/* Input params:       n/a                        */
/* Output params:      float: Value               */
/* ********************************************** */
float pid_getKi(void)
{
 return pidConfig.fKi;
}


/* ********************************************** */
/* Method name:        pid_setKd                  */
/* Method description: Set a new value for the PID */
/*                     derivative constant        */
/* Input params:       fKd: New value             */
/* Output params:      n/a                        */
/* ********************************************** */
void pid_setKd(float fKd)
{
 pidConfig.fKd = fKd;
}


/* ********************************************** */
/* Method name:        pid_getKd                  */
/* Method description: Get the value from the PID */
/*                     derivative constant        */
/* Input params:       n/a                        */
/* Output params:      float: Value               */
/* ********************************************** */
float pid_getKd(void)
{
 return pidConfig.fKd;
}


/* ********************************************** */
/* Method name:        pid_updateData             */
/* Method description: Update the control output  */
/*                     using the reference and sensor */
/*                     value                      */
/* Input params:       fSensorValue: Value read from  */
/*                     the sensor                 */
/*                     fReferenceValue: Value used as */
/*                     control reference          */
/*        fDutyCycleHeater: Value of the */
/*        heater duty cycle     */
/* Output params:      float: New Control effort     */
/* ********************************************** */
float pidUpdateData(unsigned char ucTempAtual, float fSetValue, float fDutyCycleHeater)
{
 float fError, fDifference, fOut;

 fError = fSetValue - ucTempAtual;
 /*Devemos incrementar o erro apenas se nÃŁo houver saturado o duty cycle evitando o wind up*/
 if(fDutyCycleHeater < 1.0|| fDutyCycleHeater > 0.0){
  pidConfig.fError_sum += fError;
 }
 fDifference = pidConfig.fError_previous - fError;

 fOut = pidConfig.fKp*fError
   + pidConfig.fKi*pidConfig.fError_sum
   + pidConfig.fKd*fDifference;
```

```
 pidConfig.fError_previous = fError;

 if (fOut>1)
  fOut = 1;

 else if (fOut<0.0)
  fOut = 0.0;

 return fOut;
}
```