```
#include "Control.h"
#include "src/EmgSensor.hpp"
// Pin per comandare il motore tramite motor shied (modulo A)
#define CurrentPin A0    // non utilizzato per questo codice
#define BrakePin 9       // HIGH -> blocca l'alimetaziobe al motore
#define PwmPin 3         // 0 -> 255
#define DirectionPin 12  //HIGH -> CW - LOW -> CCW
// Pin utilizzati per leggere e l'encoder
#define EncPinA 2
#define EncPinB 13
#define HomingPin A6
// Pin utilizzati per il pulsante di cambio stato
#define ButtonPin 6
#define ButtonDelay 50  // Delay in ms inserito per evitare di campionare le fluttuazioni che seguono la pressione del pulsante
// Pin utilizzati per i finecorsa
#define FineCorsaIn 10
#define FineCorsaOut 5
#define EmergencyPin 50
//Pin utilizzati per accendere i led
#define OpenedLed 31
#define ReachedLed 33
#define ClosedLed 29
#define ButtonStop 52
#define ButtonStopChange 7

//Utilizzo di maschere per i pin dell'encoder
const unsigned int mask_EncPinA = digitalPinToBitMask(EncPinA);
const unsigned int mask_EncPinB = digitalPinToBitMask(EncPinB);

//Variabili per determinare la posizione di cmano chiusa/aperta
float ManoChiusa =4;
float ManoAperta= 0.01  ;
// Oss non imposto = 0 per problemi con divisione per 0 e per scostarsi dal finecorsa

//Variabile utilizzata per comandare il ponte H tramite Pwm e delimitarne i limiti
int PwmMax = 255;
#define PwmMin 3
#define PwmHoming 10
int PwmValue = 0;
#define ErrorLimit 0.5
#define PwmLimit 5
//bool CheckLimit = true;
#define CurPresa 0.02

//Variabile utilizzate durante le prove per plottare su monitor seriale nuovi valori solo quando il motore è in movimento
float GiriPrev = 0;

//Variabile utilizzata per decidere se aprire o chiudere la Mano
int HandStateNew = 1;  //2 = Mano pronta per essere aperta o mano in apertura
                       //1 = Mano pronta per essere chiusa o mano in chiusura
int HandStateOld = 1;
bool HandState = false;
//Variabili utilizzate per definire la traiettoria nei transitori di chiusura e apertura
bool TrajClosing = false;
bool TrajOpening = false;

//Variabili utilizzate per temporizzare il codice
unsigned long t4, t3, t2, t1;
unsigned long dt = 0;

// Variabile utilizzata per verificare quando è stato raggiunta la destinazione o si è trovato un ostacolo
bool Reached = true;

// //Variabile utilizzata per capire quando viene raggiunta la distinazione per la prima volta (si resetta ogni volta che viene raggiunta la destinazione)
// bool ReachedFT = true;
bool TrajFT = true;

// Variabili utilizzate per la lettura della velocità
```

```
69  int CntPrev = 0;               // Variabile utilizzata per conteggiare i pulse tra un
    intervallo e quello successivo
70  float SpeedRad, SpeedRpm = 0;  // Variabili utilizzate per contenere la misurazione
    di velocità
71
72  // Variabili utilizzate per la lettura di posizione
73  float Error = 0;
74  int PulseCnt = 0;
75  float Giri = 0;
76  float PositionStop=0;
77
78  //Erorre accettabile per determinare raggiunta la posizione finale
79  #define Eps 0.5
80  #define EpsAperta 50
81
82  // Intervalli in ms utilizzati per dare consistenza al codice
83  #define PrintTime 100      // Intervallo in ms tra una stampa seriale e la successiva
84  //#define ReachedTime 2000   // Intervallo in ms dopo il quale se l'Error è inferiore
    al valore determinato (Eps) il motore viene spento
85
86  #define TrajectoryTime 50  // Intervallo in ms tra un aumento/diminuzione di
    Trajectory
87  #define ClosingTime 2000  // Tempo richiesto per la chiusura della mano in ms
88  #define ClosingDelta (ManoChiusa - ManoAperta) / (ClosingTime / TrajectoryTime)
89  #define OpeningDelta ClosingDelta  // Delta di incremento/dcremento della traiettoria
90
91  EmgSensor Emg(Serial3, 115200);
92
93  void setup() {
94    Serial.begin(115200);      // Inizializzazione della comunicazione seriale
95    Emg.begin();
96    analogReadResolution(12);  // Determinazione della risoluzione dell'ADC a 12 bit
97
98    // Inizializzazione dei pin per la lettura dell'encoder
99    REG_PIOB_PDR = mask_EncPinA;    // Attivate peripheral function (disabilitate le
      funzioni di PIO)
100   REG_PIOB_ABSR |= mask_EncPinA;  // Scelta peripheral option B
101   REG_PIOB_PDR = mask_EncPinB;    // Attivate peripheral function (disabilitate le
      funzioni di PIO)
102   REG_PIOB_ABSR |= mask_EncPinB;  // Scelta peripheral option B
103
104   // Attivazione del clock per TC0
105   activateCNT_TC0();
106
107   // Dichichiarazione delle porte come Input/Output
108   pinMode(CurrentPin, INPUT);
109   pinMode(BrakePin, OUTPUT);
110   pinMode(PwmPin, OUTPUT);
111   pinMode(DirectionPin, OUTPUT);
112
113   pinMode(ButtonPin, INPUT_PULLUP);
114   pinMode(ButtonStop, INPUT);
115   pinMode(HomingPin, INPUT);
116   pinMode(FineCorsaIn, INPUT_PULLUP);
117   pinMode(FineCorsaOut, OUTPUT);
118 pinMode(ButtonStopChange, INPUT);
119   pinMode(EmergencyPin, INPUT);
120   //digitalWrite(EmergencyPin, LOW);
121   delay(500);
122   //attachInterrupt(digitalPinToInterrupt(EmergencyPin), Stop, LOW);
123   attachInterrupt(digitalPinToInterrupt(EmergencyPin), Stop, RISING);
124  pinMode (OpenedLed, OUTPUT);
125  pinMode (ReachedLed, OUTPUT);
126  pinMode (ClosedLed, OUTPUT);
127
128   // Imposizione del valore basso per evitare indesiderati reset dell'encoder
129   digitalWrite(FineCorsaOut, LOW);
130
131   //Homing del setup
132   Setpoint = 4;
133 }
134
135
```

```
136  void loop() {
137
138    dt = (millis() - t1);
139    if (dt > (Ts * 1000)) {
140      VarReading();
141      t1 = millis();
142    }
143    HandStateNew=Emg.getHandState();
144    if ((HandStateNew != HandStateOld) || (digitalRead(ButtonPin) == LOW )) {
145      //Serial.println("preChange");
146      Change();  // Imposizione dell'obiettivo da raggiungere in base allo stato
       precedente
147      //Serial.println("postChange");
148      HandStateOld=HandStateNew;
149    }
150    if (digitalRead(ButtonStop) == HIGH) {
151
152      StopPosition();
153    }
154    //if (digitalRead(ButtonStopChange) == HIGH) {
155    //StopPosition1();
156   //}
157
158    if ((millis() - t2) > TrajectoryTime) {
159      Setpoint = Trajectory(Setpoint);  // Creazione di una traiettoria a trapezio per
       evitare brusche risposte a seguito di ingressi a gradino
160      t2 = millis();
161    }
162
163    if (RegEnable) {
164      if ((millis() - t3) > (Ts * 1000)) {
165        LedStatus(HandState, Reached);      // Segnalazione dello stato del sistema
         tramite led
166        CheckReached(Giri, Error, AvgCur);  // Verifica di aver raggiunto la
         destinazione desiderata
167        Output = Regulators(Setpoint, Giri, SpeedRad, AvgCur);
168        PwmValue = MotorAct(Output);        // Calcolo del valore PWM da applicare al
         motore
169        analogWrite(PwmPin, PwmValue);      // Azionamento del motore
170        t3 = millis();
171      }
172    }
173
174    //Stampa su monitor seriale (solo se sono passati almeno PrintTime ms e se il
     motore si è mosso)
175    if (((millis() - t4) > PrintTime) && (Giri != GiriPrev)) {
176      MonitorSeriale(Setpoint, Giri, Error, Output, PwmValue, Current, AvgCur, SpeedRpm
       );
177      t4 = millis();
178      GiriPrev = Giri;
179    }
180
181      /*if (digitalRead(ButtonStopChange) == HIGH) {
182      Change();
183    } */
184
185  }
186
187  // Funzione che permette la corretta lettura delle variabili di Posizione, velocità e
     corrente
188  void VarReading() {
189    // Lettura posizione
190    PulseCnt = REG_TC0_CV0;                // Lettura dei ppr dell'encoder
191    Giri = (float)PulseCnt / EncoderQuad;  // Conversione da ppr a giri del motore (a
       valle del riduttore)
192    // Lettura velocità
193    SpeedRad = SpeedCalc(dt, PulseCnt, CntPrev);
194    SpeedRpm = SpeedRad * RadToRpm;
195    CntPrev = PulseCnt;
196    // Lettura della corrente
197    Current = (((float)analogRead(CurrentPin) / 4095) * 2);
198    AvgCur = AveregeCurrent(Current);  // Valore medio della corrente calcolato per
       evitare letture spurie
```

```
199
200     Error = Setpoint - Giri;  // Errore utilizzato per test di funzionamento
201   }
202
203   // Funzione che permette di imporre il setopint di posizione desiderato
204   void Change() {
205     // Se la mano è chiusa avvio l'apertura
206     if (HandState == true) {
207       //Serial.println("Apro");
208       TrajOpening = true;
209     } else {  // Se la mano è aperta avvio la chiusura
210       //Serial.println("Chiudo");
211       TrajClosing = true;
212     }
213     RegEnable = true;     // Azionamento del controllo
214     delay(ButtonDelay);   // Attesa per evitare di leggere le fluttuazioni del pulsante
215     digitalWrite(BrakePin, LOW);
216     Reached = false;
217     CheckLimit = false;
218     LimitFT = true;
219
220     // E' necessario sapere quanto tempo è passato dalla prima richiesta
221     /*if (TrajFT) {
222       t3 = millis();
223       TrajFT = false;
224     }*/
225   }
226
227   // Funzione che permette di creare una traiettoria a trapezio
228   double Trajectory(double Setpoint) {
229     if (TrajOpening) {
230       if (Setpoint > (ManoAperta + OpeningDelta)) {
231         // Ogni TrajectoryTime ms se il set point != mano aperta, viene decrementato il
          setpoint attuale
232         Setpoint = ManoAperta;
233       } else {
234         // set point = mano aperta
235         Setpoint = ManoAperta;
236         TrajOpening = false;
237         TrajFT = true;
238         CheckLimit = true;
239       }
240     } else if (TrajClosing) {
241       if (Setpoint < (ManoChiusa - ClosingDelta)) {
242         // Ogni TrajectoryTime ms se il set point != mano chiusa, viene incrementato il
          setpoint attuale
243         Setpoint = ManoChiusa;
244       } else {
245         // set point = mano chiusa
246         Setpoint = ManoChiusa;
247         TrajClosing = false;
248         TrajFT = true;
249         CheckLimit = true;
250       }
251     }
252     return Setpoint;
253   }
254
255   // Funzione che permette di accendere i led per monitorare lo stato del sistema
256   void LedStatus(bool HandState, bool Reached) {
257
258     if(HandState == true) {
259       // Mano aperta in chiusura
260       digitalWrite(OpenedLed, HIGH);
261       digitalWrite(ClosedLed, LOW);
262     } else {
263       // Mano chiusa o in apertura
264       digitalWrite(ClosedLed, HIGH);
265       digitalWrite(OpenedLed, LOW);
266       }
267
268       if (Reached) {
269          digitalWrite(ReachedLed, HIGH);
```

```
270       } else
271       {
272        digitalWrite(ReachedLed, LOW);
273       }
274   }
275
276   // Funzione che verifica il corretto raggiungimento del setpoint richiesto
277   void CheckReached(float Giri, float Error, float Current) {
278     // Se la mano è in apertura e l'errore è inferiore a epsilon
279     if ( (HandState == false && ((-EpsAperta <= (Giri / ManoAperta)) && ((Giri /
      ManoAperta) <= EpsAperta)) ) || Current >= CurPresa ) {
280         //Serial.println("Aperto");
281         HandState = true;  // Mano pronta per essere chiusa
282         Reached = true;
283         //RegEnable = false;  // Interrompo il calcolo delle variabili del Pid
284     }
285     // Se la mano è in chiusura e l'errore è inferiore a epsilon
286     if ( (HandState == true && ((-Eps <= (Giri / ManoChiusa) - 1) && ((Giri / ManoChiusa
      ) - 1 <= Eps)) ) || Current >= CurPresa) {
287         //Serial.println("Chiuso");
288         HandState = false;  // Mano pronta per essere aperta
289         Reached = true;
290        // RegEnable = false;  // Interrompo il calcolo delle variabili del Pid
291     }
292   }
293
294   // Funzione che determina il valore PWM da applicare al motore ricevuta l'uscita del
      regolaotre di corrente
295   int MotorAct(double out) {
296
297     // Determinazione della direzione di rotazione
298     if (out > 0) {
299       digitalWrite(DirectionPin, LOW);
300       //Serial.println("Gira1");
301     } else {
302       digitalWrite(DirectionPin, HIGH);
303       //Serial.println("Gira2");
304     }
305
306     // Calcolo il valore PWM da fornire al motore
307     int Pwm = map(fabs(out), 0, MotorVin, 0, PwmMax);
308
309     // Imposizione della dinamica a bassa velocità se ci trova nei pressi della
      destinazione
310     if ( (fabs(Error) < ErrorLimit) && CheckLimit ) {
311       TauPos = 0.2;
312       TauVel = 0.5;
313       CurTau = 0.01;
314       PwmMax = PwmLimit;
315     } else {
316       TauPos = 0.2;
317       TauVel = 0.002;
318       CurTau = 0.0001;
319       PwmMax = 255;
320     }
321
322    if (Pwm < PwmMin) {
323       Pwm = PwmMin;
324     }
325
326     return Pwm;
327   }
328
329   // Funzione che arresta immediatamente il motore qual'ora si raggiunga il finecorsa
      oltre alla posizione di mano completamente chiusa
330    void Stop() {
331     Serial.println("666");
332      analogWrite(PwmPin, 0);
333      digitalWrite(BrakePin, HIGH);
334      Serial.println("Emercency stop");
335      while (1) {
336        delay(1000);
337      }
```

```
}
void StopPosition() {

 analogWrite(PwmPin, 0);
 digitalWrite(BrakePin, HIGH);
 Serial.println("Stop Position");
 PositionStop=Giri;
 ManoChiusa=PositionStop;
}
void StopPosition1() {

 analogWrite(PwmPin, 0);
 digitalWrite(BrakePin, HIGH);
 Serial.println("Stop Position");
 PositionStop=Giri;
 ManoAperta=PositionStop;



}
```