```
#include "motor.h"
#include "../vars/constants.h"
#include "../controller/controller.h"
#include "../music/buzzer.h"
#include "../led/led.h"
#include "../ultrasonic-sensors/sensors.h"
#include "../music/MP3.h"
#include "servoController.h"

#define DIRECTION_FORWARDS  0
#define DIRECTION_RIGHT     90
#define DIRECTION_BACKWARDS 180
#define DIRECTION_LEFT      270

std::string MP3::mp3File;
int MP3::VOLUME;
TaskHandle_t MP3::mp3TaskHandle;

class Movement
{
    private:
    Motor Motor_FL;
    Motor Motor_FR;
    Motor Motor_BL;
    Motor Motor_BR;

    Controller controller;
    Led led = Led();
    Sensors sensors = Sensors();

    ServoController servoController;

    enum MovementModeEnum
    {
        MovementMode_JoyLeft,
        MovementMode_GasBreak,
        MovementMode_LAST,
    };

    public:
    uint16_t DIRECTION, SPEED;
    uint8_t MovementMode;
    bool controllerButtonSelectPrev = false, controllerButtonCirclePrev
       = false;

    void init()
    {
        servoController.init();

        Motor_FL.CHANNEL = 0;
        Motor_FL.DIR_PIN = constants::pins::motor::FrontLeft_Dir;
        Motor_FL.SPEED_PIN = constants::pins::motor::FrontLeft_Speed;
        Motor_FL.FORWARD_DIRECTION = false;
```

```
53          Motor_FL.init();
54
55          Motor_FR.CHANNEL = 1;
56          Motor_FR.DIR_PIN = constants::pins::motor::FrontRight_Dir;
57          Motor_FR.SPEED_PIN = constants::pins::motor::FrontRight_Speed;
58          Motor_FR.FORWARD_DIRECTION = true;
59          Motor_FR.init();
60
61          Motor_BL.CHANNEL = 2;
62          Motor_BL.DIR_PIN = constants::pins::motor::BackLeft_Dir;
63          Motor_BL.SPEED_PIN = constants::pins::motor::BackLeft_Speed;
64          Motor_BL.FORWARD_DIRECTION = false;
65          Motor_BL.init();
66
67          Motor_BR.CHANNEL = 3;
68          Motor_BR.DIR_PIN = constants::pins::motor::BackRight_Dir;
69          Motor_BR.SPEED_PIN = constants::pins::motor::BackRight_Speed;
70          Motor_BR.FORWARD_DIRECTION = true;
71          Motor_BR.init();
72
73          controller.init();
74
75          MovementMode = MovementMode_GasBreak;
76
77          DIRECTION = 0;
78          SPEED = 0;
79          Apply();
80
81          MP3::mp3File = "/test.mp3";
82          //MAX VOLUME is 4096
83          MP3::VOLUME = 4096;
84      }
85
86      void FullSpeed()
87      {
88          SPEED = MAX_MOTOR_SPEED;
89      }
90
91      void Stop()
92      {
93          SPEED = 0;
94      }
95
96      void Forwards()
97      {
98          DIRECTION = DIRECTION_FORWARDS;
99      }
100     void Right()
101     {
102         DIRECTION = DIRECTION_RIGHT;
103     }
104     void Backwards()
105     {
```

```
106         DIRECTION = DIRECTION_BACKWARDS;
107     }
108     void Left()
109     {
110         DIRECTION = DIRECTION_LEFT;
111     }
112
113     void Apply()
114     {
115         double M_FL_Percent, M_FR_Percent, M_BL_Percent, M_BR_Percent,
              rad;
116
117         if(DIRECTION < 90)
118         {
119             rad = DIRECTION * DEG_TO_RAD;
120             M_FL_Percent = 1.0f;
121             M_FR_Percent = cos(rad * 2);
122             M_BL_Percent = cos(rad * 2);
123             M_BR_Percent = 1.0f;
124         }
125         else if(DIRECTION < 180)
126         {
127             rad = (DIRECTION - 90) * DEG_TO_RAD;
128             M_FL_Percent = cos(rad * 2);
129             M_FR_Percent = -1.0f;
130             M_BL_Percent = -1.0f;
131             M_BR_Percent = cos(rad * 2);
132         }
133         else if (DIRECTION < 270)
134         {
135             rad = (DIRECTION - 90) * DEG_TO_RAD;
136             M_FL_Percent = -1.0f;
137             M_FR_Percent = cos(rad * 2);
138             M_BL_Percent = cos(rad * 2);
139             M_BR_Percent = -1.0f;
140         }
141         else if (DIRECTION < 360)
142         {
143             rad = (DIRECTION - 180) * DEG_TO_RAD;
144             M_FL_Percent = cos(rad * 2);
145             M_FR_Percent = +1,0;
146             M_BL_Percent = +1.0;
147             M_BR_Percent = cos(rad * 2);
148         }
149
150         if(M_FL_Percent < 0)
151         {
152             Motor_FL.SetDirection(false);
153         }
154         else
155         {
156             Motor_FL.SetDirection(true);
157         }
```

```
158
159         if(M_FR_Percent < 0)
160         {
161             Motor_FR.SetDirection(false);
162         }
163         else
164         {
165             Motor_FR.SetDirection(true);
166         }
167
168         if(M_BL_Percent < 0)
169         {
170             Motor_BL.SetDirection(false);
171         }
172         else
173         {
174             Motor_BL.SetDirection(true);
175         }
176
177         if(M_BR_Percent < 0)
178         {
179             Motor_BR.SetDirection(false);
180         }
181         else
182         {
183             Motor_BR.SetDirection(true);
184         }
185
186         Motor_FL.SetSpeed(map(abs(M_FL_Percent) * 100.0, 0, 100, 0,     ↵
              SPEED));
187         Motor_FR.SetSpeed(map(abs(M_FR_Percent) * 100.0, 0, 100, 0,     ↵
              SPEED));
188         Motor_BL.SetSpeed(map(abs(M_BL_Percent) * 100.0, 0, 100, 0,     ↵
              SPEED));
189         Motor_BR.SetSpeed(map(abs(M_BR_Percent) * 100.0, 0, 100, 0,     ↵
              SPEED));
190     }
191
192     void TurnLeft()
193     {
194         Motor_FL.SetDirection(false);
195         Motor_BL.SetDirection(false);
196         Motor_FR.SetDirection(true);
197         Motor_BR.SetDirection(true);
198
199         Motor_FL.SetSpeed(SPEED);
200         Motor_BL.SetSpeed(SPEED);
201         Motor_FR.SetSpeed(SPEED);
202         Motor_BR.SetSpeed(SPEED);
203     }
204
205     void TurnRight()
206     {
```

```
207         Motor_FL.SetDirection(true);
208         Motor_BL.SetDirection(true);
209         Motor_FR.SetDirection(false);
210         Motor_BR.SetDirection(false);
211 
212         Motor_FL.SetSpeed(SPEED);
213         Motor_BL.SetSpeed(SPEED);
214         Motor_FR.SetSpeed(SPEED);
215         Motor_BR.SetSpeed(SPEED);
216     }
217 
218     uint8_t HandleControllerInputs()
219     {
220         sensors.CanGoFront();
221         if(!Ps3.isConnected())
222         {
223             DIRECTION = 0;
224             SPEED = 0;
225             Apply();
226             return 1;
227         }
228 
229         controller.loop();
230 
231         if(controller.konamiCode)
232         {
233             Serial.println("KONAMI CODE ENTERED!");
234             Dance("test");
235         }
236 
237         if(controller.buttonSelect == 1 && controllerButtonSelectPrev
              == false)
238         {
239             MovementMode++;
240             if(MovementMode >= MovementMode_LAST)
241             {
242                 MovementMode = 0;
243             }
244         }
245 
246         if(controller.dPadLeft)
247             servoController.turnServoLowerCW();
248         else if(controller.dPadRight)
249             servoController.turnServoLowerCCW();
250 
251         if(controller.dPadUp)
252             servoController.turnServoUpperCCW();
253         else if(controller.dPadDown)
254             servoController.turnServoUpperCW();
255 
256         if(controller.shoulderButtonL)
257             servoController.retractExtender();
258         else if(controller.shoulderButtonR)
```

```
259             servoController.extendExtender();
260
261         servoController.applyPosition();
262
263
264         controllerButtonSelectPrev = controller.buttonSelect == 1;
265         if(controller.buttonSquare == 1)
266         {
267             //Serial.println(sensors.CanGoFront());
268         }
269         if(controller.buttonCircle == 1 && controllerButtonCirclePrev
              == 0)
270         {
271             led.Toggle();
272         }
273         controllerButtonCirclePrev = controller.buttonCircle == 1;
274         switch(MovementMode)
275         {
276             case MovementMode_JoyLeft:
277                 HandleMovementModeJoyLeft();
278                 break;
279
280             case MovementMode_GasBreak:
281                 HandleMovementModeGasBreak();
282                 break;
283
284             default:
285                 return 2;
286         }
287
288         return 0;
289     }
290
291     void HandleMovementModeGasBreak()
292     {
293         if(controller.joyLX < -10 || controller.joyLX > 10)
294         {
295             SPEED = map(abs(controller.joyLX), 0, 128, 0,
                  MAX_MOTOR_SPEED);
296             if(controller.joyLX < 0)
297             {
298                 TurnLeft();
299             }
300             else
301             {
302                 TurnRight();
303             }
304             return;
305         }
306
307         SPEED = map(abs(controller.throttleGas -
              controller.throttleBreake), 0, 128, 0, MAX_MOTOR_SPEED);
308
```

```
309         if(controller.throttleGas - controller.throttleBreake > 0)
310         {
311             DIRECTION = 180;
312         }
313         else
314         {
315             DIRECTION = 0;
316         }
317 
318         Apply();
319     }
320 
321     void HandleMovementModeJoyLeft()
322     {
323         if(controller.joyRX < -10 || controller.joyRX > 10)
324         {
325             SPEED = map(abs(controller.joyRX), 0, 128, 0,
                MAX_MOTOR_SPEED);
326             if(controller.joyRX < 0)
327             {
328                 TurnLeft();
329             }
330             else
331             {
332                 TurnRight();
333             }
334             return;
335         }
336 
337         if(controller.joyLX != 0)
338         {
339             float tmp = atan(float(controller.joyLY) / float
                (controller.joyLX));
340             DIRECTION = tmp * RAD_TO_DEG + 90;
341 
342             if(controller.joyLX < 0)
343             {
344             DIRECTION += 180;
345             }
346         }
347         else
348         {
349             DIRECTION = 0;
350         }
351         SPEED = map(sqrt(controller.joyLY*controller.joyLY +
              controller.joyLX*controller.joyLX), 0, 128, 0,
              MAX_MOTOR_SPEED);
352         Apply();
353     }
354 
355     void Dance(std::string filename)
356     {
357         Serial.println("Dance start");
```

```
358         Serial.println("0");
359         led.StopBlink();
360         Serial.println("1");
361         fs::File file = SD.open(("/" + filename + ".txt").c_str(),
              FILE_READ);
362         Serial.println("2");
363         if (!file)
364         {
365             Serial.println("3.1");
366             ESP_LOGE("Movement: Dance", "Failed to open file");
367             file.close();
368             return;
369         }
370         char fileLinesTmp[file.size()];
371         Serial.println("2.1");
372         file.read((uint8_t *)fileLinesTmp, sizeof(fileLinesTmp));
373         Serial.println("2.2");
374         file.close();
375         Serial.println("2.3");
376         std::string fileLines = fileLinesTmp;
377         Serial.println("3");
378         MP3::mp3File = ("/" + filename + ".mp3").c_str();
379         Serial.println("4");
380         MP3::Play();
381         Serial.println("5");
382         unsigned long timeStarted = millis(), timeNext = 0;
383         Serial.println("6");
384         uint16_t servoLowerPosPrev =
              servoController.servoLowerPosition, servoUpperPosPrev =
              servoController.servoUpperPosition;
385         bool led1 = false, led2 = false, ledTop = false, lastLine =
              false;
386         std::string fileLine = "";
387         Serial.println("7");
388         DIRECTION = 0;
389         SPEED = 0;
390         eTaskState mp3TaskState;
391         if(MP3::mp3TaskHandle != NULL)
392             mp3TaskState = eTaskGetState(MP3::mp3TaskHandle);
393         else
394             mp3TaskState = eInvalid;
395
396
397         Serial.println("Main dance loop");
398         while(!controller.buttonSelect && (mp3TaskState == eRunning ||
              mp3TaskState == eReady || mp3TaskState == eBlocked))
399         {
400             controller.loop();
401             if(MP3::mp3TaskHandle != NULL)
402                 mp3TaskState = eTaskGetState(MP3::mp3TaskHandle);
403             else
404                 mp3TaskState = eInvalid;
405
```

```
406             if(controller.dPadUp && !controller.dPadUpPrev &&
                  MP3::VOLUME + 512 <= MP3_MAX_VOLUME)
407             {
408                 MP3::VOLUME += 512;
409             }
410             else if(controller.dPadDown && !controller.dPadDownPrev &&
                  MP3::VOLUME - 512 >= MP3_MAX_VOLUME)
411             {
412                 MP3::VOLUME -= 512;
413             }
414
415             if(millis() > timeNext + timeStarted)
416             {
417                 Apply();
418                 servoController.applyPosition();
419                 if(led1)
420                     Led::led_1_on();
421                 else
422                     Led::led_1_off();
423                 if(led2)
424                     Led::led_2_on();
425                 else
426                     Led::led_2_off();
427                 if(ledTop)
428                     Led::led_top_on();
429                 else
430                     Led::led_top_off();
431
432
433                 if(!lastLine)
434                 {
435                     //fileLine:
                      "timeNext,speed,direction,servoLowerPos,servoUpperPos
                      ,led1,led2,ledTop"
436                     //          "8951,256,0,90,50,ON,OFF,OFF"
437
438                     fileLine = fileLines.substr(0, fileLines.find
                      ('\n'));
439                     fileLines = fileLines.substr(fileLines.find('\n') +
                       1);
440                     if(fileLines.find('\n') == fileLines.npos)
441                         lastLine = true;
442
443                     timeNext = atoll(fileLine.substr(0, fileLine.find
                      (',')).c_str());
444
445                     Serial.println(fileLine.c_str());
446                     Serial.println(mp3TaskState);
447
448                     fileLine = fileLine.substr(fileLine.find(',') + 1);
449                     SPEED = atoi(fileLine.substr(0, fileLine.find
                      (',')).c_str());
450
```

```
451                     fileLine = fileLine.substr(fileLine.find(',') + 1);
452                     DIRECTION = atoi(fileLine.substr(0, fileLine.find
                    (',')).c_str());
453
454                     fileLine = fileLine.substr(fileLine.find(',') + 1);
455                     servoController.servoLowerPosition = atoi
                    (fileLine.substr(0, fileLine.find(',')).c_str());
456
457                     fileLine = fileLine.substr(fileLine.find(',') + 1);
458                     servoController.servoUpperPosition = atoi
                    (fileLine.substr(0, fileLine.find(',')).c_str());
459
460                     fileLine = fileLine.substr(fileLine.find(',') + 1);
461                     if(fileLine.substr(0, fileLine.find(',')) == "ON")
462                         led1 = true;
463                     else
464                         led1 = false;
465
466                     fileLine = fileLine.substr(fileLine.find(',') + 1);
467                     if(fileLine.substr(0, fileLine.find(',')) == "ON")
468                         led2 = true;
469                     else
470                         led2 = false;
471
472                     fileLine = fileLine.substr(fileLine.find(',') + 1);
473                     if(fileLine.substr(0, fileLine.find(',')) == "ON")
474                         ledTop = true;
475                     else
476                         ledTop = false;
477                 }
478             }
479         }
480         Serial.println("End of Dance");
481         servoController.servoLowerPosition = servoLowerPosPrev;
482         Serial.println("96");
483         servoController.servoUpperPosition = servoUpperPosPrev;
484         Serial.println("97");
485         servoController.applyPosition();
486         Serial.println("98");
487         MP3::Stop();
488         Serial.println("99");
489         led.StopBlink();
490         Serial.println("100");
491     }
492 };
493
```