

```

1  /*****
2  Module
3  StageSM.c
4
5  Revision
6  2.0.1
7
8  Description
9  Handles staging.
10 This is based on a template file for implementing state machines.
11
12 *****/
13 /*----- Include Files -----*/
14 // Basic includes for a program using the Events and Services Framework
15 #include "ES_Configure.h"
16 #include "ES_Framework.h"
17
18 /* include header files for this state machine as well as any machines at the
19    next lower level in the hierarchy that are sub-machines to this machine
20 */
21 #include "StageSM.h"
22 #include "Location.h"
23 #include "MasterVehicle.h"
24 #include "LOCMaster.h"
25 /*----- Module Defines -----*/
26
27 // define constants for the states for this machine
28 // and any other local defines
29 /* Team Definition */
30 #define RED 0
31 #define GREEN 1
32
33 #define ENTRY_STATE STAGE_WAITING
34 #define ONE_SEC 976
35 #define HALF_SEC (ONE_SEC / 2)
36 #define FREQ_TIME HALF_SEC
37 #define THREE_SEC 3000
38 #define TWO_SEC 2000
39 #define STAGE_WAITING_TIME TWO_SEC
40 #define COMPETITION_FULL_DUTY 75
41 #define CORRECTION_FULL_DUTY 55
42 /*----- Module Functions -----*/
43 /* prototypes for private functions for this machine, things like during
44    functions, entry & exit functions.They should be functions relevant to the
45    behavior of this state machine
46 */
47 static ES_Event DuringWaiting( ES_Event Event);
48 static ES_Event DuringMoveY( ES_Event Event);
49 static ES_Event DuringMoveX( ES_Event Event);
50 static ES_Event DuringFreqMeasurement( ES_Event Event);
51 static ES_Event DuringWaitingForResponse( ES_Event Event);
52 /*----- Module Variables -----*/
53 // everybody needs a state variable, you may need others as well
54 static StagingState_t CurrentState;
55 static uint32_t CurrentXDestination=0;
56 static uint32_t CurrentYDestination=0;
57 /*----- Module Code -----*/
58 /*****
59 Function
60 RunStageSM
61
62 Parameters
63 ES_Event: the event to process
64
65 Returns
66 ES_Event: an event to return
67
68 *****/
69 ES_Event RunStageSM( ES_Event CurrentEvent )
70 {
71     bool MakeTransition = false; /* are we making a state transition? */
72     StagingState_t NextState = CurrentState;

```

```

73     ES_Event EntryEventKind = { ES_ENTRY, 0 }; // default to normal entry to new state
74     ES_Event ReturnEvent = CurrentEvent; // assume we are not consuming event
75
76     switch ( CurrentState )
77     {
78     case STAGE_WAITING : // If current state is waiting state
79         // Execute During function for state one. ES_ENTRY & ES_EXIT are
80         // processed here allow the lower level state machines to re-map
81         // or consume the event
82         CurrentEvent = DuringWaiting(CurrentEvent);
83         //process any events
84         if ( CurrentEvent.EventType != ES_NO_EVENT ) //If an event is active
85         {
86             switch (CurrentEvent.EventType)
87             {
88                 case STAGE_ACTIVE : //If event is STAGE_ACTIVE
89                     uint8_t dest_index = 0;
90                     // check team selection
91                     // if team is GREEN
92                     if(get_Team() == GREEN){
93                         // get stage destination index
94                         dest_index = queryActiveStagingGreen();
95                         // if index is between 0 and 5, which mean staging is active
96                         // we can get the coordinate x,y for this location
97                         if ((dest_index!=0)&&(dest_index!=5)){
98                             CurrentXDestination = get_Stage_Green_X(dest_index);
99                             CurrentYDestination = get_Stage_Green_Y(dest_index);
100                         }
101                     }
102                     // else if team is RED
103                 else{
104                     // get stage destination index
105                     dest_index = queryActiveStagingRed();
106                     // if index is between 0 and 5, which mean staging is active
107                     // we can get the coordinate x,y for this location
108                     if ((dest_index!=0)&&(dest_index!=5)){
109                         CurrentXDestination = get_Stage_Red_X(dest_index);
110                         CurrentYDestination = get_Stage_Red_Y(dest_index);
111                     }
112                 }
113
114                 // Execute action function for state one : event one
115                 NextState = STAGE_MOVE_Y; //Decide what the next state will be
116                 // for internal transitions, skip changing MakeTransition
117                 MakeTransition = true; //mark that we are taking a transition
118                 // if transitioning to a state with history change kind of entry
119                 // EntryEventKind.EventType = ES_ENTRY_HISTORY;
120                 // optionally, consume or re-map this event for the upper
121                 // level state machine
122                 ReturnEvent.EventType = ES_NO_EVENT;
123                 break;
124
125                 default:
126                     break;
127             }
128         }
129         break;
130
131     case STAGE_MOVE_Y : // If current state is STAGE_MOVE_Y
132         // Execute During function for state one. ES_ENTRY & ES_EXIT are
133         // processed here allow the lower level state machines to re-map
134         // or consume the event
135         CurrentEvent = DuringMoveY(CurrentEvent);
136         //process any events
137         if ( CurrentEvent.EventType != ES_NO_EVENT ) //If an event is active
138         {
139             switch (CurrentEvent.EventType)
140             {
141
142                 case SHOOT_ACTIVE_4: // if event is SHOOT_ACTIVE_4 which is shooting in last 18 sec
143                     NextState = STAGE_WAITING; // set next state to waiting
144                     MakeTransition = true; //mark that we are taking a transition

```

```

145         ReturnEvent.EventType = SHOOT_ACTIVE_4; //return this event to upper SM
146         break;
147
148     case Y_REACHED : //If event is reached target y location
149         // Execute action function for state one : event one
150         NextState = STAGE_MOVE_X; // set next state to moving in x
151         // for internal transitions, skip changing MakeTransition
152         MakeTransition = true; //mark that we are taking a transition
153         // if transitioning to a state with history change kind of entry
154         // EntryEventKind.EventType = ES_ENTRY_HISTORY;
155         // optionally, consume or re-map this event for the upper
156         // level state machine
157         ReturnEvent.EventType = ES_NO_EVENT;
158         break;
159     case CONSTRUCTION_END: //If event is construction end, we want to exit this SM
160         NextState = STAGE_WAITING; //Decide what the next state will be
161         MakeTransition = true; //mark that we are taking a transition
162         ReturnEvent.EventType = CONSTRUCTION_END; // return this event to upper SM
163         break;
164
165     default:
166         break;
167     }
168 }
169 break;
170
171 case STAGE_MOVE_X : // If current state is STAGE_MOVE_X
172     // Execute During function for state one. ES_ENTRY & ES_EXIT are
173     // processed here allow the lower level s tate machines to re-map
174     // or consume the event
175     CurrentEvent = DuringMoveX(CurrentEvent);
176     //process any events
177     if ( CurrentEvent.EventType != ES_NO_EVENT ) //If an event is active
178     {
179         switch (CurrentEvent.EventType)
180         {
181             case SHOOT_ACTIVE_4: // if event is SHOOT_ACTIVE_4 which is shooting in last 18 sec
182                 NextState = STAGE_WAITING; //set next state to waiting
183                 MakeTransition = true; //mark that we are taking a transition
184                 ReturnEvent.EventType = SHOOT_ACTIVE_4; // return this event to upper SM
185                 break;
186
187                 case X_REACHED: // If event is reaching target x location
188                     // we reverifiy that the current y location is still our target y location
189                     if(verify_y_location()){
190                         // if y location is correct, reset PWM to high speed
191                         set_PWM_Full_Duty(COMPETITION_FULL_DUTY);
192                         // set next state to STAGE_VERIFICATION
193                         NextState = STAGE_VERIFICATION;
194                     }
195                     // if our y location is incorrect
196                 else{
197                     // set PWM speed to lower correction speed (to avoid overshoot)
198                     set_PWM_Full_Duty(CORRECTION_FULL_DUTY);
199                     printf("Y location invalid, call move y again\r\n");
200                     // set next state to STAGE_MOVE_Y
201                     NextState = STAGE_MOVE_Y;
202                 }
203                 MakeTransition = true;
204                 ReturnEvent.EventType = ES_NO_EVENT;
205                 break;
206
207                 case CONSTRUCTION_END: //If event is CONSTRUCTION_END
208                     NextState = STAGE_WAITING; //Decide what the next state will be
209                     MakeTransition = true; //mark that we are taking a transition
210                     ReturnEvent.EventType = CONSTRUCTION_END; // return this event to upper SM
211                     break;
212
213             default:
214                 break;
215             }
216         }

```

```

217         break;
218
219     case STAGE_VERIFICATION :           // If current state is STAGE_VERIFICATION
220         // Execute During function for state one. ES_ENTRY & ES_EXIT are
221         // processed here allow the lower level state machines to re-map
222         // or consume the event
223         CurrentEvent = DuringFreqMeasurement(CurrentEvent);
224         //process any events
225         if ( CurrentEvent.EventType != ES_NO_EVENT ) //If an event is active
226         {
227             switch (CurrentEvent.EventType)
228             {
229             case ES_TIMEOUT:
230                 // if we get ES_TIMEOUT from the stage_timer,
231                 // we should exit stage SM and enter stage SM again
232                 if(CurrentEvent.EventParam == STAGE_TIMER){
233                     NextState = STAGE_WAITING;
234                     MakeTransition = true; //mark that we are taking a transition
235                     printf("STAGE SM Timeout, go back to IDLE and restart stage SM\r\n");
236                     ReturnEvent.EventType = ES_TIMEOUT; //return this event to upper SM
237                     // also post RESTART_VERIFY_FREQ to LOCMaster SM
238                     ES_Event to_post;
239                     printf("Post to LOCMaster to restart verifying stage\r\n");
240                     to_post.EventType = RESTART_VERIFY_FREQ;
241                     PostLOCMasterSM(to_post);
242                     break;
243                 }
244
245                 case FINISHED_STAGING: //If event is FINISHED_STAGING
246                     // Execute action function for state one : event one
247                     NextState = STAGE_WAITING; // set next state to STAGE_WAITING
248                     MakeTransition = true; //mark that we are taking a transition
249                     printf("STAGE SM received FINISHED STAGING - returning to STAGE WAITING\r\n");
250                     ReturnEvent.EventType = FINISHED_STAGING; // return this event to upper SM
251                     break;
252
253             case CONSTRUCTION_END: //If event is CONSTRUCTION_END
254                 NextState = STAGE_WAITING; // set next state to waiting
255                 MakeTransition = true; //mark that we are taking a transition
256                 ReturnEvent.EventType = CONSTRUCTION_END; // return this event to upper SM
257                 break;
258
259             default:
260                 break;
261             }
262         }
263         break;
264
265     default:
266         break;
267     }
268     // If we are making a state transition
269     if (MakeTransition == true)
270     {
271         // Execute exit function for current state
272         CurrentEvent.EventType = ES_EXIT;
273         RunStageSM(CurrentEvent);
274
275         CurrentState = NextState; //Modify state variable
276
277         // Execute entry function for new state
278         // this defaults to ES_ENTRY
279         RunStageSM(EntryEventKind);
280     }
281     return(ReturnEvent);
282 }
283 /*****
284 Function
285     StartStageSM
286
287 Parameters
288     None

```

```

289
290     Returns
291         None
292
293     Description
294         Does any required initialization for this state machine
295
296     *****/
297 void StartStageSM ( ES_Event CurrentEvent )
298 {
299     // to implement entry to a history state or directly to a substate
300     // you can modify the initialization of the CurrentState variable
301     // otherwise just start in the entry state every time the state machine
302     // is started
303     if ( ES_ENTRY_HISTORY != CurrentEvent.EventType )
304     {
305         CurrentState = ENTRY_STATE;
306     }
307     // call the entry function (if any) for the ENTRY_STATE
308     RunStageSM(CurrentEvent);
309 }
310
311 /*****
312     Function
313         QueryStageSM
314
315     Parameters
316         None
317
318     Returns
319         StageState_t The current state of the Stage state machine
320
321     *****/
322 StagingState_t QueryStageSM ( void )
323 {
324     return(CurrentState);
325 }
326
327 /*****
328     private functions
329     *****/
330
331 static ES_Event DuringWaiting( ES_Event Event)
332 {
333     ES_Event ReturnEvent = Event; // assume no re-mapping or consumption
334     // process ES_ENTRY, ES_ENTRY_HISTORY & ES_EXIT events
335     if ( (Event.EventType == ES_ENTRY) || (Event.EventType == ES_ENTRY_HISTORY) ){
336         // no the lower level
337     }
338     else if ( Event.EventType == ES_EXIT ){
339     }
340     else{
341     }
342     // return either Event, if you don't want to allow the lower level machine
343     // to remap the current event, or ReturnEvent if you do want to allow it.
344     return(ReturnEvent);
345 }
346
347 static ES_Event DuringMoveY( ES_Event Event)
348 {
349     ES_Event ReturnEvent = Event; // assume no re-mapping or consumption
350     // process ES_ENTRY, ES_ENTRY_HISTORY & ES_EXIT events
351     if ( (Event.EventType == ES_ENTRY) || (Event.EventType == ES_ENTRY_HISTORY) ){
352         // implement any entry actions required for this state machine
353         move_Y(CurrentYDestination);
354     }
355     else if ( Event.EventType == ES_EXIT ){
356         // make sure the motor is stopped when exiting this moving state
357         stopMotor();
358     }
359     else{
360     }

```

```

361     // return either Event, if you don't want to allow the lower level machine
362     // to remap the current event, or ReturnEvent if you do want to allow it.
363     return(ReturnEvent);
364 }
365
366 static ES_Event DuringMoveX( ES_Event Event)
367 {
368     ES_Event ReturnEvent = Event; // assume no re-mapping or consumption
369
370     // process ES_ENTRY, ES_ENTRY_HISTORY & ES_EXIT events
371     if ( (Event.EventType == ES_ENTRY) || (Event.EventType == ES_ENTRY_HISTORY) ){
372         // implement any entry actions required for this state machine
373         move_X(CurrentXDestination); // call move function to target x destination
374     }
375     else if ( Event.EventType == ES_EXIT ){
376         // on exit, give the lower levels a chance to clean up first
377         // make sure the motor is stopped when exiting this moving state
378         stopMotor();
379     }
380     else{
381     }
382     // return either Event, if you don't want to allow the lower level machine
383     // to remap the current event, or ReturnEvent if you do want to allow it.
384     return(ReturnEvent);
385 }
386
387 static ES_Event DuringFreqMeasurement( ES_Event Event)
388 {
389     ES_Event ReturnEvent = Event; // assume no re-mapping or consumption
390
391     // process ES_ENTRY, ES_ENTRY_HISTORY & ES_EXIT events
392     if ( (Event.EventType == ES_ENTRY) || (Event.EventType == ES_ENTRY_HISTORY) ){
393         // implement any entry actions required for this state machine
394         // when entering this state, post ARRIVE_AT_STAGING event to
395         // LOCMaster to start frequency verification process
396         ES_Event to_post;
397         printf("Post arrive at staging to LOC Master\r\n");
398         to_post.EventType = ARRIVED_AT_STAGING;
399         PostLOCMasterSM(to_post);
400         // start STAGE_TIMER to limit the maximum time it takes
401         // to verify frequencies and send two reports
402         ES_Timer_InitTimer(STAGE_TIMER, STAGE_WAITING_TIME);
403     }
404     else if ( Event.EventType == ES_EXIT ){
405     }
406     else{
407     }
408     // return either Event, if you don't want to allow the lower level machine
409     // to remap the current event, or ReturnEvent if you do want to allow it.
410     return(ReturnEvent);
411 }
412

```