/\*

\* File: FS\_SubHSM.c

\* Author: Brian Naranjo

\* Date: 11/13/19

\*

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* MODULE #INCLUDE \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "BOARD.h"

#include "AD.h"

#include "ES\_Configure.h"

#include "ES\_Framework.h"

#include "BotHSM.h"

#include "FS\_SubHSM.h"

#include "Motors.h"

#include <stdio.h>

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* MODULE #DEFINES \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define TIME\_TILL\_PARALLEL 2000 //2sec, Amount of time needed to spin bot parallel to side

#define TIME\_TO\_CHECK 1000 //Duration for checking Side, If expired, no TrackWire

#define TANK\_TURN\_SPEED 600 //Speed for Tank Turning Smoothly

#define TURN\_SPEEDL 500 //Speed for constant motion around target, L Motor

#define TURN\_SPEEDR 800 //Speed for constant motion around target, R Motor

#define STOP\_SPEED 0 //Macro to stop motors

#define D\_THRESHOLD 3 //Macro for distance to wall to assume parallel to wall

#define TIME\_TO\_SIDE\_ONE 500

#define TIME\_TO\_SIDE\_TWO 400

#define TIME\_TO\_SIDE\_THREE 1000

#define TIME\_TO\_CHECK 200

#define TIME\_S\_MOTION\_1 500

#define TIME\_S\_MOTION\_2 500

#define TIME\_S\_MOTION\_3 500

#define S1\_LEFT\_SPEED 500

#define S1\_RIGHT\_SPEED 500

#define S2\_LEFT\_SPEED 500

#define S2\_RIGHT\_SPEED 500

#define S3\_LEFT\_SPEED 500

#define S3\_RIGHT\_SPEED 500

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* PRIVATE FUNCTION PROTOTYPES \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* PRIVATE MODULE VARIABLES \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

static uint8\_t MyPriority;

typedef enum {

INIT\_STATE,

INITIAL TURN,

REALIGN\_FRONT,

REALIGN\_BACK,

CHECK\_TRACKWIRE,

MOVE\_TO\_F\_SIDE1,

MOVE\_TO\_F\_SIDE2,

MOVE\_TO\_F\_SIDE3,

MOVE\_TO\_B\_SIDE1,

MOVE\_TO\_B\_SIDE2,

MOVE\_TO\_B\_SIDE3,

PARALLEL\_START,

S\_MOTION\_1,

S\_MOTION\_2,

S\_MOTION\_3,

PARALLEL\_DONE,

} FS\_State\_t;

static FS\_State\_t CurrentState = INIT\_STATE; //Initialize static Variables

static ES\_Event ThisEvent;

static uint8\_t MyPriority;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* PUBLIC FUNCTIONS \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*

\* @Function InitFS\_SubHSM(uint8\_t Priority)

\* @param Priority - internal variable to track which event queue to use

\* @return TRUE or FALSE

\* @brief This will get called by the framework at the beginning of the code

\* execution. It will post an ES\_INIT event to the appropriate event

\* queue, which will be handled inside RunFS\_SubHSM function.

\* Returns TRUE if successful, FALSE otherwise

\* @author J. Edward Carryer, 2011.10.23 19:25

\* @editor Brian Naranjo, 11/13/19 \*/

uint8\_t InitFS\_SubHSM(void){

ES\_Event returnEvent;

CurrentState = INIT\_STATE;

ThisEvent.EventType = ES\_INIT;

returnEvent = RunFS\_SubHSM(INIT\_EVENT);

if (returnEvent.EventType == ES\_NO\_EVENT) {

return TRUE;

} else {

return FALSE;

}

}

/\*\*

\* @Function PostFS\_SubHSM(ES\_Event ThisEvent)

\* @param ThisEvent - the event (type and param) to be posted to queue

\* @return TRUE or FALSE

\* @brief This function is a wrapper to the queue posting function, and its name

\* will be used inside ES\_Configure to point to which queue events should

\* be posted to. Remember to rename to something appropriate.

\* Returns TRUE if successful, FALSE otherwise

\* @author J. Edward Carryer, 2011.10.23 19:25

\* @editor Brian Naranjo, 11/13/19 \*/

uint8\_t PostFS\_SubHSM(ES\_Event ThisEvent){

return ES\_PostToService(MyPriority, ThisEvent);

}

/\*\*

\* @Function RunFS\_SubHSM(ES\_Event ThisEvent)

\* @param ThisEvent - the event (type and param) to be responded.

\* @return Event - return event (type and param), in general should be ES\_NO\_EVENT

\* @brief This function is where you implement the whole of the service,

\* as this is called any time a new event is passed to the event queue.

\* @note Returns ES\_NO\_EVENT if the event have been "consumed."

\* @author J. Edward Carryer, 2011.10.23 19:25

\* @editor Brian Naranjo, 11/13/19 \*/

ES\_Event RunFS\_SubHSM(ES\_Event ThisEvent){

uint8\_t makeTransition = FALSE;

FS\_State\_t NextState; //declare NextState Variable

ES\_Tattle(); // trace call stack

switch (CurrentState){

case INIT\_STATE:

if (ThisEvent.EventType == ES\_INIT){// only respond to ES\_Init

NextState = PARALLEL\_PARK; //Transition to First State

ES\_Timer\_InitTimer(PARKING\_TIMER, TIME\_TILL\_PARALLEL);

makeTransition = TRUE;

ThisEvent.EventType = ES\_NO\_EVENT; //Initialize Event

}

break;

case PARALLEL\_PARK:

if(ThisEvent.EventType == ES\_TIMEOUT){ //If Parallel Park Done, Check Side

NextState = CHECK\_SIDE; //Next State

ES\_Timer\_InitTimer(CHECK\_TIMER, TIME\_TO\_CHECK);

makeTransition = TRUE; //Switch States

ThisEvent.EventType = ES\_NO\_EVENT; //Initialize Event

Motors\_LeftDrive(STOP\_SPEED); //Set Both Motors to STOP

Motors\_RightDrive(STOP\_SPEED);

}

Motors\_TankTurn(TANK\_TURN\_SPEED, TANK\_TURN\_RIGHT);

break;

case CHECK\_SIDE:

if(ThisEvent.EventType == ES\_TIMEOUT){

NextState = SWITCH\_SIDE; //Next State if Check Timer Expires

makeTransition = TRUE; //Switch States

ES\_Timer\_InitTimer(SS\_TIMER, TIME\_SWITCH\_SIDE); //Time for bot to switch side

ThisEvent.EventType = ES\_NO\_EVENT; //Initialize Event

} else if(ThisEvent.EventType == TRACKWIRE\_BOTH\_TRIGGERED){

NextState = SIDE\_FOUND; //Next State

makeTransition = TRUE; //Switch States

ThisEvent.EventType = ES\_NO\_EVENT; //Initialize Event

}

Motors\_LeftDrive(STOP\_SPEED); //Set Both Motors to STOP

Motors\_RightDrive(STOP\_SPEED);

break;

case SWITCH\_SIDE:

if(ThisEvent.EventType == ES\_TIMEOUT){

NextState = CHECK\_SIDE; //Once Side is Switched, Check again

makeTransition = TRUE;

ThisEvent.EventType = ES\_NO\_EVENT;

}

Motors\_LeftDrive(TURN\_SPEEDL); //Set Both Motors to Turn Left around target

Motors\_RightDrive(TURN\_SPEEDR);

break;

case SIDE\_FOUND:

if(ThisEvent.EventType == ES\_Init){

NextState = PARALLEL\_PARK; //Transition back to First State In case Stuck

ES\_Timer\_InitTimer(PARKING\_TIMER, TIME\_TILL\_PARALLEL);

makeTransition = TRUE;

ThisEvent.EventType = ES\_NO\_EVENT; //Initialize Event

}

Motors\_LeftDrive(STOP\_SPEED); //Set Both Motors to STOP

Motors\_RightDrive(STOP\_SPEED);

break;

case INIT\_STATE:

if (ThisEvent.EventType == ES\_INIT){// only respond to ES\_Init

NextState = INITIAL\_TURN; //Transition to First State

ES\_Timer\_InitTimer(PARKING\_TIMER, TIME\_TILL\_PARALLEL);

makeTransition = TRUE;

ThisEvent.EventType = ES\_NO\_EVENT; //Initialize Event

}

break;

case INITIAL\_TURN:

Motors\_TankTurn(APPROACH\_SPEED, TANK\_TURN\_RIGHT);

if((Ping\_Distance() > D\_THRESHOLD) && (Ping\_Distance2() > D\_THRESHOLD)){

NextState = CHECK\_TRACKWIRE;

makeTransition = TRUE;

ThisEvent.EventType = ES\_NO\_EVENT; //Initialize Event

ES\_Timer\_InitTimer(PARKING\_TIMER, TIME\_TO\_CHECK);

}

// if (ThisEvent.EventType == ES\_TIMEOUT){

// if((Ping\_Distance() < D\_THRESHOLD) && (Ping\_Distance2() < D\_THRESHOLD)){ //both sides close

// NextState = CHECK\_TRACKWIRE;

// makeTransition = TRUE;

// ThisEvent.EventType = ES\_NO\_EVENT; //Initialize Event

// } else if((Ping\_Distance() > D\_THRESHOLD) && (Ping\_Distance2() < D\_THRESHOLD)){

// NextState = REALIGN\_FRONT;

// makeTransition = TRUE;

// ThisEvent.EventType = ES\_NO\_EVENT; //Initialize Event

// } else if((Ping\_Distance() < D\_THRESHOLD) && (Ping\_Distance2() > D\_THRESHOLD)){

// NextState = REALIGN\_BACK;

// makeTransition = TRUE;

// ThisEvent.EventType = ES\_NO\_EVENT; //Initialize Event

// }

// }

break;

case REALIGN\_FRONT:

Motors\_TankTurn(APPROACH\_SPEED, TANK\_TURN\_LEFT);

if((Ping\_Distance() > D\_THRESHOLD) && (Ping\_Distance2() > D\_THRESHOLD)){

NextState = CHECK\_TRACKWIRE;

makeTransition = TRUE;

ThisEvent.EventType = ES\_NO\_EVENT; //Initialize Event

ES\_Timer\_InitTimer(PARKING\_TIMER, TIME\_TO\_CHECK);

}

break;

case REALIGN\_BACK:

Motors\_TankTurn(APPROACH\_SPEED, TANK\_TURN\_LEFT);

if((Ping\_Distance() > D\_THRESHOLD) && (Ping\_Distance2() > D\_THRESHOLD)){

NextState = CHECK\_TRACKWIRE;

makeTransition = TRUE;

ThisEvent.EventType = ES\_NO\_EVENT; //Initialize Event

}

break;

case CHECK\_TRACKWIRE:

Motors\_LeftDrive(STOP\_SPEED); //Set Both Motors to STOP

Motors\_RightDrive(STOP\_SPEED);

if(ThisEvent.EventType == TRACKWIRE\_BOTH\_TRIGGERED){

NextState = PARALLEL\_START;

makeTransition = TRUE;

ThisEvent.EventType = ES\_NO\_EVENT; //Initialize Event

} else if(ThisEvent.EventType == TRACKWIRE\_L\_TRIGGERED){

NextState = MOVE\_TO\_B\_SIDE1;

makeTransition = TRUE;

ThisEvent.EventType = ES\_NO\_EVENT; //Initialize Event

ES\_Timer\_InitTimer(PARKING\_TIMER, TIME\_TO\_SIDE\_ONE);

} else if(ThisEvent.EventType == TRACKWIRE\_R\_TRIGGERED){

NextState = MOVE\_TO\_F\_SIDE1;

makeTransition = TRUE;

ThisEvent.EventType = ES\_NO\_EVENT; //Initialize Event

ES\_Timer\_InitTimer(PARKING\_TIMER, TIME\_TO\_SIDE\_ONE);

}

break;

case MOVE\_TO\_F\_SIDE1:

if(ThisEvent.EventType == ES\_TIMEOUT){

NextState = MOVE\_TO\_F\_SIDE2;

makeTransition = TRUE;

ThisEvent.EventType = ES\_NO\_EVENT; //Initialize Event

ES\_Timer\_InitTimer(PARKING\_TIMER, TIME\_TO\_SIDE\_TWO);

}

break;

case MOVE\_TO\_F\_SIDE2:

if(ThisEvent.EventType == ES\_TIMEOUT){

NextState = MOVE\_TO\_F\_SIDE3;

makeTransition = TRUE;

ThisEvent.EventType = ES\_NO\_EVENT; //Initialize Event

ES\_Timer\_InitTimer(PARKING\_TIMER, TIME\_TO\_SIDE\_THREE);

}

break;

case MOVE\_TO\_F\_SIDE3:

if(ThisEvent.EventType == ES\_TIMEOUT){

NextState = PARALLEL\_START;

makeTransition = TRUE;

ThisEvent.EventType = ES\_NO\_EVENT; //Initialize Event

ES\_Timer\_InitTimer(PARKING\_TIMER, TIME\_TO\_CHECK);

}

break;

case MOVE\_TO\_B\_SIDE1:

if(ThisEvent.EventType == ES\_TIMEOUT){

NextState = MOVE\_TO\_B\_SIDE2;

makeTransition = TRUE;

ThisEvent.EventType = ES\_NO\_EVENT; //Initialize Event

ES\_Timer\_InitTimer(PARKING\_TIMER, TIME\_TO\_SIDE\_TWO);

}

break;

case MOVE\_TO\_B\_SIDE2:

if(ThisEvent.EventType == ES\_TIMEOUT){

NextState = MOVE\_TO\_B\_SIDE3;

makeTransition = TRUE;

ThisEvent.EventType = ES\_NO\_EVENT; //Initialize Event

ES\_Timer\_InitTimer(PARKING\_TIMER, TIME\_TO\_SIDE\_THREE);

}

break;

case MOVE\_TO\_B\_SIDE3:

if(ThisEvent.EventType == ES\_TIMEOUT){

NextState = PARALLEL\_START;

makeTransition = TRUE;

ThisEvent.EventType = ES\_NO\_EVENT; //Initialize Event

ES\_Timer\_InitTimer(PARKING\_TIMER, TIME\_TO\_CHECK);

}

break;

case PARALLEL\_START:

Motors\_LeftDrive(STOP\_SPEED); //Set Both Motors to STOP

Motors\_RightDrive(STOP\_SPEED);

if(ThisEvent.EventType == ES\_TIMEOUT){

NextState = S\_MOTION\_1;

makeTransition = TRUE;

ThisEvent.EventType = ES\_NO\_EVENT; //Initialize Event

ES\_Timer\_InitTimer(PARKING\_TIMER, TIME\_S\_MOTION\_1);

}

break;

case S\_MOTION\_1:

Motors\_LeftDrive(S1\_LEFT\_SPEED); //Set Both Motors to STOP

Motors\_RightDrive(S1\_RIGHT\_SPEED);

Motors\_LeftDirection(MOTOR\_BACKWARD);

Motors\_RightDirection(MOTOR\_BACKWARD);

if(ThisEvent.EventType == ES\_TIMEOUT){

NextState = S\_MOTION\_2;

makeTransition = TRUE;

ThisEvent.EventType = ES\_NO\_EVENT; //Initialize Event

ES\_Timer\_InitTimer(PARKING\_TIMER, TIME\_S\_MOTION\_2);

}

break;

case S\_MOTION\_2:

Motors\_LeftDrive(S2\_LEFT\_SPEED); //Set Both Motors to STOP

Motors\_RightDrive(S2\_RIGHT\_SPEED);

Motors\_LeftDirection(MOTOR\_BACKWARD);

Motors\_RightDirection(MOTOR\_BACKWARD);

if(ThisEvent.EventType == ES\_TIMEOUT){

NextState = S\_MOTION\_3;

makeTransition = TRUE;

ThisEvent.EventType = ES\_NO\_EVENT; //Initialize Event

ES\_Timer\_InitTimer(PARKING\_TIMER, TIME\_S\_MOTION\_3);

}

break;

case S\_MOTION\_3:

Motors\_LeftDrive(S3\_LEFT\_SPEED); //Set Both Motors to STOP

Motors\_RightDrive(S3\_RIGHT\_SPEED);

Motors\_LeftDirection(MOTOR\_FORWARD);

Motors\_RightDirection(MOTOR\_FORWARDD);

if(ThisEvent.EventType == GUN\_BUMPERS\_HIT){

NextState = PARALLEL\_DONE;

makeTransition = TRUE;

ThisEvent.EventType = ES\_NO\_EVENT; //Initialize Event

Motors\_LeftDrive(STOP\_SPEED); //Set Both Motors to STOP

Motors\_RightDrive(STOP\_SPEED);

} else if(ThisEvent.EventType == ES\_TIMEOUT){

NextState = S\_MOTION\_1;

makeTransition = TRUE;

ThisEvent.EventType = ES\_NO\_EVENT; //Initialize Event

}

case PARALLEL\_DONE:

Motors\_LeftDrive(STOP\_SPEED); //Set Both Motors to STOP

Motors\_RightDrive(STOP\_SPEED);

break;

case default:

break;

if (makeTransition == TRUE) { // making a state transition, send EXIT and ENTRY

// recursively call the current state with an exit event

RunFS\_SubHSM(EXIT\_EVENT); // <- Exit SM

CurrentState = NextState; //Change States

RunFS\_SubHSM(ENTRY\_EVENT); // <- Enter SM

}

}

ES\_Tail(); // trace call stack end

return ThisEvent;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* PRIVATE FUNCTIONs \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/