```c
/****************************************************************************
 Module
   NavigationFSM.c

 Revision
   1.0.1

 Description
   Flat state machine for bot navigation. Queries

 Notes

 History
 When           Who     What/Why
 -------------- ---     --------
 02/21/13       DYL      began editing for NavigationFSM
****************************************************************************/
/*----------------------------- Include Files -----------------------------*/
/* include header files for this state machine as well as any machines at the
   next lower level in the hierarchy that are sub-machines to this machine*/
#include <stdio.h>
#include <stdlib.h>
#include <mc9s12e128.h>
#include <S12e128bits.h>
#include <Bin_Const.h>
#include <termio.h>
#include <hidef.h>
#include <math.h>
#include "S12eVec.h"

#include "E128_PWM.h"         //has all prescale definitions
#include "E128_SPI.h"
#include "E128_Servo.h"
#include "FAC_FSM.h"
#include "NavigationFSM.h"
#include "AlignPPService.h"
#include "DriveTrainService.h"
#include "ArtilleryFSM.h"
#include "StrategyFSM.h"



/*----------------------------- Module Defines ----------------------------*/


/*---------------------------- Module Functions ---------------------------*/
/* prototypes for private functions for this service.They should be functions
   relevant to the behavior of this service
*/
//returns TRUE if within error of desired coordinate)
//static char ReachedCoord(unsigned char, unsigned char);
static char ReachedCoord(signed int, unsigned char, unsigned char);
static char ReachedTheta(signed int, signed int);
static char TightReachedTheta(signed int, signed int);
static char CalcOptimumRotation(signed int CurTheta, signed int DesiredTheta);
//static void RampDownMoving(unsigned char CurCoord, unsigned char DesiredCoord);
//static void RampDownTurning(unsigned char CurTheta, unsigned char DesiredTheta);
```

```
/*---------------------------- Module Variables ---------------------------*/
// everybody needs a state variable, you may need others as well.
// type of state variable should match htat of enum in header file
static unsigned char Desiredx;
static unsigned char Desiredy;

//variable storing distance between current and desired (x,y,theta)
static signed int dist_x;
static signed int dist_y;
static signed int dist_theta;
static signed char CurDir;
static unsigned char AlignTheta;

//(x,y) for desired location
static signed int AlignXTheta; //align with positive x-axis (270 deg)
static signed int AlignYTheta = (255*18)/36; //align with negative y-axis (180 deg)
static signed int MaxTheta = 255;
static signed int MinTheta = 0;

//360 degrees = [0 - 255], 1 degree < 1 "tick"
static signed int TolTheta = 35; //22 degrees   (1.41 degrees/tick)
static signed int TightTolTheta = 3; //6 degrees
//8ft = [0-255], so 1 ft = 32 "ticks", 1 inch ~ 2.5 ticks
static signed int TolDist = 8; //approx. 1 inch (0.4 in/tick)

// with the introduction of Gen2, we need a module level Priority variable
static uint8_t MyPriority;
static NavigationState_t CurrentState;


/*------------------------------ Module Code ------------------------------*/
/****************************************************************************
 Function
     InitNavigationFSM

 Parameters
     uint8_t : the priorty of this service

 Returns
     boolean, False if error in initialization, TRUE otherwise

 Description
     Saves away the priority, sets up the initial transition and does any
     other required initialization for this state machine
 Notes

 Author
     Jina Wang 3/4/2013
****************************************************************************/
boolean InitNavigationFSM ( uint8_t Priority )
{
   ES_Event ThisEvent;

   MyPriority = Priority;

   // put us into the PreparingToMove
   CurrentState = PreparingToMove;

   // post the initial transition event
   ThisEvent.EventType = ES_INIT;
```

```
    if (ES_PostToService( MyPriority, ThisEvent) == TRUE)
        {
            return TRUE;
        }
    else
        {
            return False;
        }
}

/******************************************************************************
 Function
     PostNavigationFSM

 Parameters
     EF_Event ThisEvent , the event to post to the queue

 Returns
     boolean False if the Enqueue operation failed, TRUE otherwise

 Description
     Posts an event to this state machine's queue
 Notes

 Author
     Jina Wang 3/4/2013
******************************************************************************/
boolean PostNavigationFSM( ES_Event ThisEvent )
{
    return ES_PostToService( MyPriority, ThisEvent);
}

/******************************************************************************
 Function
    RunNavigationFSM

 Parameters
   ES_Event : the event to process

 Returns
   ES_Event, ES_NO_EVENT if no error ES_ERROR otherwise

 Description
   add your description here
 Notes
   uses nested switch/case to implement the machine.
 Author
        Jina Wang 3/4/2013
******************************************************************************/
ES_Event RunNavigationFSM( ES_Event ThisEvent )
{
    // Put all EventTypes used by NavigationFSM to post to other modules.
    ES_Event ReturnEvent, MtrEvent, MtrEvent2, StrategyEvent;
    ReturnEvent.EventType = ES_NO_EVENT; // assume no errors


    /******************************************************************************

                              BEGIN STATE MACHINE CODE
```

```
****************************************************************************/

switch ( CurrentState )
   {

   case ( PreparingToMove ) :
      if (  ThisEvent.EventType == NEW_DESTINATION )
         {
            Desiredx = GetDesiredX(); //xloc of RED resupply service    //215
            Desiredy = GetDesiredY(); //yloc of RED resupply service   //128

            if (QueryColor() == BLUE )
               {
                  AlignXTheta = ( (255*9)/(36) ); //old code 7/36
               }
            else
               {
                  AlignXTheta = ( (255*27)/(36) );
               }

            printf("got event to navigate to %i, %i\r\n", Desiredx, Desiredy);
            //printf("current position is %i, %i, %i\r\n", QueryX(SelfNum), QueryY(SelfNum), QueryTheta
(SelfNum));

            //figure out how far in x,y,theta, bot needs to move
            dist_x = Desiredx - QueryX(SelfNum);
            dist_y = Desiredy - QueryY(SelfNum);
            dist_theta = AlignXTheta - QueryTheta(SelfNum);

            printf("Distance to navigate is %i, %i, %i \r\n", dist_x, dist_y, dist_theta);

            //Determine best direction to rotate
            MtrEvent.EventType = ROTATE;
            MtrEvent.EventParam = CalcOptimumRotation(QueryTheta(SelfNum), AlignXTheta);
            PostDriveTrainService(MtrEvent);

           CurDir = CalcOptimumRotation(QueryTheta(SelfNum), AlignXTheta);

            CurrentState = AligningX;
         }
      break; //end preparing to move

   case ( AligningX ) :
      if (ThisEvent.EventType == FAC_UPDATED)
         {

            // Evaluate if the difference in desired theta and actual theta is less than our tolerance
            if ( ReachedTheta(QueryTheta(SelfNum), AlignXTheta) )
               {
                  //puts("in aligningX and reached desired theta and now movinginX\r\n");
                  //We've reached a value less than the tolerance band, post to stop our motors
                  MtrEvent.EventType = STOP_MOTOR;
                  MtrEvent.EventParam = 0;
                  PostDriveTrainService(MtrEvent);

                  //determine movement in +/- x-direction

                  if (dist_x > 0)
                     {
                        AlignTheta = AlignXTheta;
```

```
                                    if (QueryColor() == RED)
                                        {
                                            MtrEvent.EventType = DRIVE;
                                            MtrEvent.EventParam = FORWARD;
                                            PostDriveTrainService(MtrEvent);
                                        }

                                    else
                                        {
                                            MtrEvent.EventType = DRIVE;
                                            MtrEvent.EventParam = REVERSE;
                                            PostDriveTrainService(MtrEvent);
                                        }

                                    CurrentState = MovingX;
                                }
                            else if (dist_x < 0)
                                {
                                    AlignTheta = AlignXTheta;

                                    if (QueryColor() == RED)
                                        {
                                            MtrEvent.EventType = DRIVE;
                                            MtrEvent.EventParam = REVERSE;
                                            PostDriveTrainService(MtrEvent);
                                        }

                                    else
                                        {
                                            MtrEvent.EventType = DRIVE;
                                            MtrEvent.EventParam = FORWARD;
                                            PostDriveTrainService(MtrEvent);
                                        }

                                    CurrentState = MovingX;
                                }
                            //if at x-coordinate after alignment, align with (-) y-axis  <0 degrees>
                            else
                                {
                                    // We are at our desired x, re-evaluate displacement in theta to align with y-axis
                                    //Determine best direction to rotate
                                    MtrEvent.EventType = ROTATE;
                                    MtrEvent.EventParam = CalcOptimumRotation(QueryTheta(SelfNum),
AlignYTheta);
                                    PostDriveTrainService(MtrEvent);

                                   CurDir = CalcOptimumRotation(QueryTheta(SelfNum), AlignYTheta);

                                    CurrentState = AligningY;
                                }
                        }

                }

            else if ( ThisEvent.EventType == ES_NEW_KEY && ThisEvent.EventParam == 49) //num1
                {
                    puts("simulated successful aligningx\r\n");
                    CurrentState = MovingX;
                }
```

```
        break; //end aligning X

    case ( MovingX ) :
        if (ThisEvent.EventType == FAC_UPDATED)
            {
                if ( ReachedCoord(dist_x, QueryX(SelfNum), Desiredx) )
                    {
                        //puts("in movingX and got to desiredx. Now going to align in y\r\n");
                        // We've reached the desired x, stop motors
                        MtrEvent.EventType = STOP_MOTOR;
                        MtrEvent.EventParam = 0;
                        PostDriveTrainService(MtrEvent);

                        //Determine best direction to rotate
                        MtrEvent.EventType = ROTATE;
                        MtrEvent.EventParam = CalcOptimumRotation(QueryTheta(SelfNum), AlignYTheta);
                        PostDriveTrainService(MtrEvent);

                        CurDir = CalcOptimumRotation(QueryTheta(SelfNum), AlignYTheta);

                        CurrentState = AligningY;
                    }
            }

        else if ( ThisEvent.EventType == ES_NEW_KEY && ThisEvent.EventParam == 50) //num2
            {
                puts("simulated successful movingx\r\n");
                CurrentState = AligningY;
            }
        break; //end moving X

    case ( AligningY ) :
        if (ThisEvent.EventType == FAC_UPDATED)
            {
                if ( ReachedTheta(QueryTheta(SelfNum), AlignYTheta) )
                    {
                        //puts("in aligningY and got to desired theta. Now going to MovingY \r\n");
                        //We've aligned with theta, send out the stop motor command
                        MtrEvent.EventType = STOP_MOTOR;
                        MtrEvent.EventParam = 0;
                        PostDriveTrainService(MtrEvent);

                        //calculate distance and direction to move in y-axis
                        dist_y = Desiredy - QueryY(SelfNum);

                        //determine movement in +/- y-direction
                        //y increases 'down' the board (180 deg), so need to move in reverse
                        if (dist_y > 0)
                            {
                                AlignTheta = AlignYTheta;

                                MtrEvent.EventType = DRIVE;
                                MtrEvent.EventParam = FORWARD;
                                PostDriveTrainService(MtrEvent);

                                CurrentState = MovingY;
                            }
                        else if (dist_y < 0)//drive forward
```

```
                                {
                                    AlignTheta = AlignYTheta;

                                    MtrEvent.EventType = DRIVE;
                                    MtrEvent.EventParam = REVERSE;
                                    PostDriveTrainService(MtrEvent);

                                    CurrentState = MovingY;
                                }
                            else // At desired y after alignment
                                {
                                    MtrEvent.EventType = STOP_MOTOR;
                                    MtrEvent.EventParam = 0;
                                    PostDriveTrainService(MtrEvent);

                                    //Determine best direction to rotate
                                    MtrEvent.EventType = ROTATE;
                                    MtrEvent.EventParam = CalcOptimumRotation(QueryTheta(SelfNum),
AlignXTheta); //MOD TO TARGET THETA
                                    PostDriveTrainService(MtrEvent);

                                    CurDir = CalcOptimumRotation(QueryTheta(SelfNum), AlignXTheta);

                                    CurrentState = TurningToTarget;
                                }
                        }
                }

            else if ( ThisEvent.EventType == ES_NEW_KEY && ThisEvent.EventParam == 51) //num3
                {
                    puts("simulated successful aligningy\r\n");
                    CurrentState = MovingY;
                }
            break; //end AligningY

        case ( MovingY ) :
            if (ThisEvent.EventType == FAC_UPDATED)
                {
                    if ( ReachedCoord(dist_y, QueryY(SelfNum), Desiredy) )
                        {
                            //puts("in movingY and got to desiredy \r\n");
                            MtrEvent.EventType = STOP_MOTOR;
                            MtrEvent.EventParam = 0;
                            PostDriveTrainService(MtrEvent);

                            //Determine best direction to rotate
                            MtrEvent.EventType = ROTATE;
                            MtrEvent.EventParam = CalcOptimumRotation(QueryTheta(SelfNum), AlignXTheta); //
MOD TO TARGET THETA
                            PostDriveTrainService(MtrEvent);

                            CurDir = CalcOptimumRotation(QueryTheta(SelfNum), AlignXTheta);

                            //CurrentState = PreparingToMove;  //TurningToTarget
                            CurrentState = TurningToTarget;
                        }

                }
            else if ( ThisEvent.EventType == ES_NEW_KEY && ThisEvent.EventParam == 52) //num4
                {
```

```
          puts("simulated successful movingy\r\n");
          CurrentState = TurningToTarget;
        }
      break; //end MovingY


      //MAY ONLY BE USEFUL WHEN BACKING UP TO RESUPPLY STATION
    case ( TurningToTarget) :
      if (ThisEvent.EventType == FAC_UPDATED)
        {

          if ( ReachedTheta(QueryTheta(SelfNum), AlignXTheta) ) //MOD TO TARGET THETA
            {

              MtrEvent.EventType = STOP_MOTOR;
              MtrEvent.EventParam = 0;
              PostDriveTrainService(MtrEvent);

              //printf("CurThetaInTurningToTarget = %i\r\n",QueryTheta(SelfNum));

              //PostDestinationReached Event to Strategy
              //StrategyEvent.EventType = DESTINATION_REACHED;
              //StrategyEvent.EventParam = 0;
              //PostStrategyFSM(StrategyEvent);
              puts("got to destination, not tight\r\n");
              CurrentState = TightTurningToTarget;
            }


        }
      break; //TurningToTarget

    case ( TightTurningToTarget ) :
      if (ThisEvent.EventType == FAC_UPDATED)
        {
          //Determine best direction to rotate
          MtrEvent.EventType = ROTATE_HALF;
          MtrEvent.EventParam = CalcOptimumRotation(QueryTheta(SelfNum), AlignXTheta); //MOD
TO TARGET THETA
          PostDriveTrainService(MtrEvent);

          //printf("CurThetaInTightTurningToTarget = %i\r\n",QueryTheta(SelfNum));


          if ( TightReachedTheta(QueryTheta(SelfNum), AlignXTheta) ) //MOD TO TARGET THETA
            {

              MtrEvent.EventType = STOP_MOTOR;
              MtrEvent.EventParam = 0;
              PostDriveTrainService(MtrEvent);

              //PostDestinationReached Event to Strategy
              StrategyEvent.EventType = DESTINATION_REACHED;
              StrategyEvent.EventParam = 0;
              PostStrategyFSM(StrategyEvent);
              printf("got to destination tight with location %i, %i, %i\r\n", QueryX(SelfNum), QueryY
(SelfNum), QueryTheta(SelfNum));
              CurrentState = PreparingToMove;
            }
```

```
                }
            break; //TightTurningToTarget


        } // End switch( CurrentState )

    return ReturnEvent;

}

/****************************************************************************

                        END STATE MACHINE CODE

****************************************************************************/

/*---------------------------- Private Functions----------------------------*/
static char ReachedCoord(signed int dist_, unsigned char CurCoord, unsigned char DesiredCoord)
{
    if ((dist_ < 0) && (CurCoord < (DesiredCoord + TolDist)))
        {
            return TRUE;
        }
    else if ((dist_ > 0) && (CurCoord > (DesiredCoord - TolDist)))
        {
            return TRUE;
        }
    else
        {
            return FALSE;
        }

}

static char ReachedTheta(signed int CurTheta, signed int DesiredTheta)
{
    if ( abs(DesiredTheta - CurTheta) <= TolTheta )
        {
            return TRUE;
        }
    else
        {
            return FALSE;
        }
}

static char TightReachedTheta(signed int CurTheta, signed int DesiredTheta)
{
    if ( abs(DesiredTheta - CurTheta) <= TightTolTheta )
        {
            return TRUE;
        }
    else
        {
            return FALSE;
        }
}

static char CalcOptimumRotation(signed int CurTheta, signed int DesiredTheta)
```

```
{
    signed int LimTheta = 0;

    if ( DesiredTheta <= (MaxTheta/2) )
        {
            LimTheta = DesiredTheta + MaxTheta/2; //DesiredTheta + 180 degrees
            if ( (CurTheta >= DesiredTheta) && (CurTheta < LimTheta) )
                {
                    return CW;
                }
            else
                {
                    return CCW;
                }
        }
    else
        {
            LimTheta = DesiredTheta - MaxTheta/2;

            if ( (CurTheta >= LimTheta) && (CurTheta < DesiredTheta) )
                {
                    return CCW;
                }
            else
                {
                    return CW;
                }
        }

}

unsigned char QueryDesiredTheta(void)
{
    return AlignTheta;
}


// Need this function for P control
void SetThetaManually(unsigned char ManualTheta)
{
    AlignTheta = ManualTheta;
}
```