```
/*---------------------------------------------------*\
|*        - Natural Language for CORTEX on Virtual Worlds -          *|
|*                                                                   *|
|*        This file includes all of the Natural Language             *|
|*        functions for CORTEX on Virtual Worlds.                        *|
\*-----------------------------------------------4246-*/

#pragma systemFile              // eliminates warning for "unreferenced" functions
#include "JoystickDriver.c"             // for use with joystick commands

// enum for RIGHT and LEFT direction:
typedef enum tDirections
{
  right = 0,
  left  = 1
}tDirections;

// enum for VEX LCD buttons:
typedef enum tVEXLCDButtons
{
  leftBtnVEX          = 1,
  centerBtnVEX        = 2,
  //leftANDcenterBtnVEX   = 3,
  rightBtnVEX         = 4//,
  //leftANDrightBtnVEX    = 5,
  //centerANDrightBtnVEX  = 6,
  //allBtnsVEX          = 7
}tVEXLCDButtons;

// enum for units
typedef enum tMovementUnits
{
  degrees           = 0,
  rotations         = 1,
  milliseconds      = 2,
  seconds           = 3,
  minutes           = 4
} tMovementUnits;


/************************\
|*  MOVEMENT FUNCTIONS  *|
```

```
\*************************************************************************************************************
*****************************/

//-----------------------------------------| startMotor |-----------------------------------------
#ifndef _STARTMotorTypes_H_GUARD
#define _STARTMotorTypes_H_GUARD

void startMotor(tMotor motorPort = port1, int speed = 95)
{
  motor[motorPort] = speed;
}

#endif
//-------------------------------------------------------------------------------------------------

//-----------------------------------------| Forward |-----------------------------------------
#ifndef _FORWARD_H_GUARD
#define _FORWARD_H_GUARD

void forward(int speed = 95)
{
  if(speed >= 0)
  {
    motor[port2] = speed;
    motor[port3] = speed;
  }
  else if(speed < 0)
  {
    motor[port2] = (-1 * speed);
    motor[port3] = (-1 * speed);
  }
}

#endif
//-------------------------------------------------------------------------------------------------

//-----------------------------------------| Backward |-----------------------------------------
#ifndef _BACKWARD_H_GUARD
#define _BACKWARD_H_GUARD

void backward(int speed = -95)
{
  if(speed <= 0)
```

```
  {
    motor[port2] = speed;
    motor[port3] = speed;
  }
  else if(speed > 0)
  {
    motor[port2] = (-1 * speed);
    motor[port3] = (-1 * speed);
  }
}

#endif
//-------------------------------------------------------------------------------------------------

//-----------------------------------------| SwingTurn |-----------------------------------------
#ifndef _SWINGTURN_H_GUARD
#define _SWINGTURN_H_GUARD

void swingTurn(tDirections direction = right, int speed = 95)
{
  if(direction == left)      // left
  {
    if(speed >= 0)
    {
      motor[port2] = speed;
      motor[port3] = 0;
    }
    else if(speed < 0)
    {
      motor[port2] = 0;
      motor[port3] = speed;
    }
  }
  else if(direction == right) // right
  {
    if(speed >= 0)
    {
      motor[port2] = 0;
      motor[port3] = speed;
    }
    else if(speed < 0)
    {
      motor[port2] = speed;
```

```
    motor[port3] = 0;
   }
  }
}

#endif
//--------------------------------------------------------------------------------------------------

//----------------------------------------| PointTurn |-----------------------------------------
#ifndef _POINTTURN_H_GUARD
#define _POINTTURN_H_GUARD

void pointTurn(tDirections direction = right, int speed = 95)
{
  if(direction == left)      // left
  {
    motor[port2] = abs(speed);
    motor[port3] = abs(speed) * -1;
    //if(speed >= 0)
    //{
    //  motor[port2] = speed * -1;
    //  motor[port3] = speed;
    //}
    //else if(speed < 0)
    //{
    //  motor[port2] = speed;
    //  motor[port3] = speed * -1;
    //}
  }
  else if(direction == right) // right
  {
    motor[port2] = abs(speed) * -1;
    motor[port3] = abs(speed);
    //if(speed >= 0)
    //{
    //  motor[port2] = speed;
    //  motor[port3] = speed * -1;
    //}
    //else if(speed < 0)
    //{
    //  motor[port2] = speed * -1;
    //  motor[port3] = speed;
    //}
```

```
  }
}

#endif
//-----------------------------------------------------------------------------------------------

//-----------------------------------------------| Stop |----------------------------------------
#ifndef _STOP_H_GUARD
#define _STOP_H_GUARD

void stop()
{
  motor[port2] = 0;
  motor[port3] = 0;
}

#endif
//-----------------------------------------------------------------------------------------------

//-------------------------------------------| StopMotor |-------------------------------------
#ifndef _STOPMOTOR_H_GUARD
#define _STOPMOTOR_H_GUARD

void stopMotor(tMotor motorPort = port1)
{
  motor[motorPort] = 0;
}

#endif
//-----------------------------------------------------------------------------------------------

//-----------------------------------------| LineTrackTIME |-----------------------------------
#ifndef _LINETRACKTIME_H_GUARD
#define _LINETRACKTIME_H_GUARD

void lineTrackForTime(float trackTime = 5.0, int threshold = 2048, tSensors leftSensorPort = in1,
tSensors centerSensorPort = in2, tSensors rightSensorPort = in3)
{
  float timeStart = ((float)nPgmTime / 1000);

  while(((float)nPgmTime / 1000) - timeStart < trackTime)
  {
    // RIGHT sensor sees dark:
```

```
    if(SensorValue(rightSensorPort) > threshold)
    {
      // counter-steer right:
      motor[port2] = 0;
      motor[port3] = 45;
    }
    // CENTER sensor sees dark:
    if(SensorValue(centerSensorPort) > threshold)
    {
      // go straight
      motor[port2] = 45;
      motor[port3] = 45;
    }
    // LEFT sensor sees dark:
    if(SensorValue(leftSensorPort) > threshold)
    {
      // counter-steer left:
      motor[port2] = 45;
      motor[port3] = 0;
    }
    wait1Msec(1);
  }
}

#endif
//-----------------------------------------------------------------------------------------------

//---------------------------------------| LineTrackROTATIONS |---------------------------------
#ifndef _LINETRACKROTS_H_GUARD
#define _LINETRACKROTS_H_GUARD

void lineTrackForRotations(float rotations = 3.0, int threshold = 2048, tSensors leftSensorPort =
in1, tSensors centerSensorPort = in2, tSensors rightSensorPort = in3)
{
  SensorValue[dgtl1] = 0;
  SensorValue[dgtl3] = 0;

  while((SensorValue[dgtl1] < (rotations * 360))  &&  (SensorValue[dgtl3] < (rotations * 360)))
  {
    // RIGHT sensor sees dark:
    if(SensorValue(rightSensorPort) > threshold)
    {
      // counter-steer right:
```

```
      motor[port2] = 0;
      motor[port3] = 45;
    }
    // CENTER sensor sees dark:
    if(SensorValue(centerSensorPort) > threshold)
    {
      // go straight
      motor[port2] = 45;
      motor[port3] = 45;
    }
    // LEFT sensor sees dark:
    if(SensorValue(leftSensorPort) > threshold)
    {
      // counter-steer left:
      motor[port2] = 45;
      motor[port3] = 0;
    }
    wait1Msec(1);
  }
}

#endif
//-------------------------------------------------------------------------------------------------

//-----------------------------------| MoveStraightROTATIONS |-----------------------------------
#ifndef _MOVESTRAIGHTROTS_H_GUARD
#define _MOVESTRAIGHTROTS_H_GUARD

void moveStraightForRotations(float rotations = 1.0, tSensors rightEncoderPort = dgtl1,
tSensors leftEncoderPort = dgtl3)
{
  SensorValue[rightEncoderPort] = 0;
  SensorValue[leftEncoderPort] = 0;

  while(SensorValue[rightEncoderPort] < (abs(rotations) * 360))
  {
    if(SensorValue[rightEncoderPort] == SensorValue[leftEncoderPort]) // If rightEncoder has
counted the same amount as leftEncoder:
    {
      // Move Forward
      motor[port2] = 85;            // Right Motor is run at power level 85
      motor[port3]  = 85;                     // Left Motor is run at power level 85
    }
```

```
   else if(SensorValue[rightEncoderPort] > SensorValue[leftEncoderPort]) // If rightEncoder has
counted more encoder counts
   {
     // Turn slightly right
     motor[port2] = 65;           // Right Motor is run at power level 65
     motor[port3]  = 85;                  // Left Motor is run at power level 85
   }
   else // Only runs if leftEncoder has counted more encoder counts
   {
     // Turn slightly left
     motor[port2] = 85;           // Right Motor is run at power level 85
     motor[port3]  = 65;                  // Left Motor is run at power level 65
   }
  }
}

#endif
//-------------------------------------------------------------------------------------------------

//---------------------------------------| MoveStraightTIME |-----------------------------------
#ifndef _MOVESTRAIGHTTIME_H_GUARD
#define _MOVESTRAIGHTTIME_H_GUARD

void moveStraightForTime(float seconds = 5.0, tSensors rightEncoderPort = dgtl1, tSensors
leftEncoderPort = dgtl3)
{
  SensorValue[rightEncoderPort] = 0;
  SensorValue[leftEncoderPort] = 0;

  float timeStart = ((float)nPgmTime / 1000);

  while(((float)nPgmTime / 1000) - timeStart < seconds)
  {
    if(SensorValue[rightEncoderPort] == SensorValue[leftEncoderPort]) // If rightEncoder has
counted the same amount as leftEncoder:
   {
     // Move Forward
     motor[port2] = 85;           // Right Motor is run at power level 85
     motor[port3]  = 85;                  // Left Motor is run at power level 85
   }
   else if(SensorValue[rightEncoderPort] > SensorValue[leftEncoderPort]) // If rightEncoder has
counted more encoder counts
   {
```

```
      // Turn slightly right
      motor[port2] = 65;            // Right Motor is run at power level 65
      motor[port3]  = 85;                   // Left Motor is run at power level 85
    }
    else // Only runs if leftEncoder has counted more encoder counts
    {
      // Turn slightly left
      motor[port2] = 85;            // Right Motor is run at power level 85
      motor[port3]  = 65;                   // Left Motor is run at power level 65
    }
  }
}

#endif
//--------------------------------------------------------------------------------------------------

//----------------------------------------| TankControl |-----------------------------------------
#ifndef _TANKCONTROL_H_GUARD
#define _TANKCONTROL_H_GUARD

void tankControl(short &rightJoystick = joystick.joy1_y2, short &leftJoystick = joystick.joy1_y1,
short threshold = 10)
{
  getJoystickSettings(joystick);
  if(rightJoystick <= abs(threshold)  &&  rightJoystick >= (abs(threshold) * -1))
    motor[port2] = 0;
  else
    motor[port2] = rightJoystick;

  if(leftJoystick <= abs(threshold)  &&  leftJoystick >= (abs(threshold) * -1))
    motor[port3] = 0;
  else
    motor[port3] = leftJoystick;
}

#endif
//--------------------------------------------------------------------------------------------------

//----------------------------------------| ArcadeControl |-----------------------------------------
#ifndef _ARCADECONTROL_H_GUARD
#define _ARCADECONTROL_H_GUARD
```

```
void arcadeControl(short &verticalJoystick = joystick.joy1_y2, short &horizontalJoystick =
joystick.joy1_x2, short threshold = 10)
{
  getJoystickSettings(joystick);
  if( (verticalJoystick <= abs(threshold)  &&  verticalJoystick >= (abs(threshold) * -1))  &&
    (horizontalJoystick <= abs(threshold)  &&  horizontalJoystick >= (abs(threshold) * -1)))
  {
    motor[port2] = 0;
    motor[port3] = 0;
  }
  else
  {
    motor[port2] = (verticalJoystick + horizontalJoystick) / 2;
    motor[port3] = (verticalJoystick - horizontalJoystick) / 2;
  }
}
#endif
//---------------------------------------------------------------------------------------------------

/*********************************************************************************************************************
*********************************\
|*  END MOVEMENT FUNCTIONS  *|
\*****************************/
/**********************\
|*  "WAIT" FUNCTIONS  *|
\*********************************************************************************************************************
*******************************/

//-----------------------------------------| Wait |-----------------------------------------
#ifndef _WAIT_H_GUARD
#define _WAIT_H_GURAD

void wait(float waitTime = 1.0)
{
  long mins       = 0;
  long secs       = 0;
  long millisecs  = 0;
  if(waitTime >= 0)
  {
    mins = (waitTime / 60.0);
    secs = ((waitTime * 1000) / 1000) - (mins * 60);
    millisecs = (waitTime * 1000) - (((long)waitTime * 1000));
  }
```

```
  else if(waitTime < 0)
  {
    mins = ((waitTime * -1) / 60.0);
    secs = (((waitTime * -1) * 1000) / 1000) - (mins * 60);
    millisecs = ((waitTime * -1) * 1000) - (((long)(waitTime * -1) * 1000));
  }

  for(long i=0; i<mins; i++)      // minutes loop
  {
    wait1Msec(60000);
  }

  for(long j=0; j<secs; j++)      // seconds loop
  {
    wait1Msec(1000);
  }

  wait1Msec(millisecs);
}

#endif
//---------------------------------------------------------------------------------------------------

//-------------------------------------| WaitInMilliseconds |-------------------------------------
#ifndef _WAITINMILLISECONDS_H_GUARD
#define _WAITINMILLISECONDS_H_GUARD

void waitInMilliseconds(long waitTime = 1000)
{
  wait1Msec((long)waitTime);
}

#endif
//---------------------------------------------------------------------------------------------------
/*****************************************************************************************************
*****************************\
|*  END "WAIT" FUNCTIONS  *|
\*************************/
/***********************\
|*  "UNTIL" FUNCTIONS  *|
\*****************************************************************************************************
*****************************/
```

```
//-----------------------------------| UntilCompassDegrees |-----------------------------------
#ifndef _UNTILCOMPASSDEGREES_H_GUARD
#define _UNTILCOMPASSDEGREES_H_GUARD

void untilCompassDegrees(int degrees = 90, int buffer = 0)
{
  int goal1 = (SensorValue(in7) + degrees) % 360;   // 'goal' is be the current compass reading +
our turning degrees
  /* We have to use modulus 360 to keep our goal within the bounds   */
  /* of the compass (1 to 360 degrees, 0 is 360).                    */

  int goal2 = abs((SensorValue(in7) - degrees) % 360);   //(goal1 + 180) % 360;

  int upperBound1 = goal1 + buffer;
  int lowerBound1 = goal1 - buffer;
  int upperBound2 = goal2 + buffer;
  int lowerBound2 = goal2 - buffer;

  //while(SensorValue(S2) != goal1 && SensorValue(S2) != goal2);    // while the compass
sensor reading is not yet our goal:
  while((SensorValue(in7) > upperBound1 || SensorValue(in7) < lowerBound1)  &&
(SensorValue(in7) > upperBound2 || SensorValue(in7) < lowerBound2)){wait1Msec(1);}

}

#endif
//-----------------------------------------------------------------------------------------------

//-------------------------------------| UntilSonarLessThan |-------------------------------------
#ifndef _UNTILSONARLESSTHAN_H_GUARD
#define _UNTILSONARLESSTHAN_H_GUARD

void untilSonarLessThan(int distance = 30, tSensors sensorPort = dgtl8)
{
  while(SensorValue[sensorPort] >= distance || SensorValue[sensorPort] == -1){wait1Msec(1);}
}

#endif
//-----------------------------------------------------------------------------------------------

//-----------------------------------| UntilSonarGreaterThan |-----------------------------------
#ifndef _UNTILSONARGREATERTHAN_H_GUARD
#define _UNTILSONARGREATERTHAN_H_GUARD
```

```
void untilSonarGreaterThan(int distance = 30, tSensors sensorPort = dgtl8)
{
  while(SensorValue[sensorPort] < distance || SensorValue[sensorPort] == -1){wait1Msec(1);}
}

#endif
//-----------------------------------------------------------------------------------------------

//---------------------------------------| UntilRotations |--------------------------------------
#ifndef _UNTILROTATIONS_H_GUARD
#define _UNTILROTATIONS_H_GUARD

void untilRotations(float rotations = 1.0, tSensors sensorPort = dgtl1)
{
  SensorValue(sensorPort) = 0;

  //while(SensorValue(sensorPort) < (abs(rotations) * 360)  &&  SensorValue(sensorPort) >
(abs(rotations) * (360 * -1)));
  while((   (SensorValue(sensorPort) < (rotations * 360))   ||
    (SensorValue(sensorPort) < (rotations * (360 * -1)))   )   &&   (   (SensorValue(sensorPort) >
(rotations * 360))   ||
  (SensorValue(sensorPort) > (rotations * (360 * -1)))   ))
  {
    wait1Msec(1);
  }

}

#endif
//-----------------------------------------------------------------------------------------------

//-------------------------------------| UntilEncoderCounts |------------------------------------
#ifndef _UNTILENCODERCOUNTS_H_GUARD
#define _UNTILENCODERCOUNTS_H_GUARD

void untilEncoderCounts(int distance = 360, tSensors sensorPort = dgtl1)
{
  SensorValue[sensorPort] = 0;

  //while(SensorValue[sensorPort] < (abs(distance))  &&  SensorValue[sensorPort] >
(abs(distance) * -1));
  while((   (SensorValue(sensorPort) < distance)   ||
```

```
    (SensorValue(sensorPort) < (distance * -1))   )   &&   (   (SensorValue(sensorPort) > distance)
||
  (SensorValue(sensorPort) > (distance * -1))   ))
  {
    wait1Msec(1);
  }

}

#endif
//-----------------------------------------------------------------------------------------------

//----------------------------------------| UntilTouch |----------------------------------------
#ifndef _UNTILTOUCH_H_GUARD
#define _UNTILTOUCH_H_GUARD

void untilTouch(tSensors sensorPort = dgtl6)
{
  while(SensorValue[sensorPort] != 1){wait1Msec(1);}
}

#endif
//-----------------------------------------------------------------------------------------------

//----------------------------------------| UntilRelease |----------------------------------------
#ifndef _UNTILRELEASE_H_GUARD
#define _UNTILRELEASE_H_GUARD

void untilRelease(tSensors sensorPort = dgtl6)
{
  while(SensorValue[sensorPort] != 0){wait1Msec(1);}
}

#endif
//-----------------------------------------------------------------------------------------------

//---------------------------------| UntilPotentiometerLessThan |---------------------------------
#ifndef _UNTILPOTENTIOMETERLESSTHAN_H_GUARD
#define _UNTILPOTENTIOMETERLESSTHAN_H_GUARD

void untilPotentiometerLessThan(int position = 2048, tSensors sensorPort = in6)
{
  while(SensorValue[sensorPort] > abs(position)){wait1Msec(1);}
```

```
}

#endif
//-----------------------------------------------------------------------------------------------

//--------------------------------| UntilPotentiometerGreaterThan |--------------------------------
#ifndef _UNTILPOTENTIOMETERGREATERTHAN_H_GUARD
#define _UNTILPOTENTIOMETERGREATERTHAN_H_GUARD

void untilPotentiometerGreaterThan(int position = 2048, tSensors sensorPort = in6)
{
  while(SensorValue[sensorPort] < abs(position)){wait1Msec(1);}
}

#endif

//-----------------------------------------| SetServo |-----------------------------------------
#ifndef _SETSERVO_H_GUARD
#define _SETSERVO_H_GUARD

void setServo(tMotor servoPort = port6, signed byte position = 0)
{
  motor[servoPort] = position;
}

#endif

//-----------------------------------------| UntilBump |-----------------------------------------
#ifndef _UNTILBUMP_H_GUARD
#define _UNTILBUMP_H_GUARD

void untilBump(tSensors sensorPort = dgtl6, int delayTimeMS = 10)
{
  while(SensorValue[sensorPort] != 1)
                    wait1Msec(1);

  int tempvar = delayTimeMS;
  wait1Msec(tempvar);

  while(SensorValue[sensorPort] == 1)
                    wait1Msec(1);
}
```

```
#endif
//-----------------------------------------------------------------------------------------------

//---------------------------------------| UntilButtonPress |--------------------------------------
#ifndef _UNTILBUTTONPRESS_H_GUARD
#define _UNTILBUTTONPRESS_H_GUARD

void untilButtonPress(short button = centerBtnVEX)
{
  while(true)
  {
    if(nLCDButtons == button)
    {
      break;
    }
    wait1Msec(1);
  }
}

#endif
//-----------------------------------------------------------------------------------------------

//-----------------------------------------| UntilDark |-----------------------------------------
#ifndef _UNTILDARK_H_GUARD
#define _UNTILDARK_H_GUARD

void untilDark(int threshold = 2048, tSensors sensorPort = in1)
{
  while(SensorValue[sensorPort] < threshold){wait1Msec(1);}
}

#endif
//-----------------------------------------------------------------------------------------------

//-----------------------------------------| UntilLight |----------------------------------------
#ifndef _UNTILLIGHT_H_GUARD
#define _UNTILLIGHT_H_GUARD

void untilLight(int threshold = 2048, tSensors sensorPort = in1)
{
  while(SensorValue[sensorPort] > threshold){wait1Msec(1);}
}
```

```
#endif
//--------------------------------------------------------------------------------------------------

/*****************************************************************************************************
*******************************\
|*  END "UNTIL" FUNCTIONS  *|
\*****************************/
void resetTimer(TTimers currentTimer = T1)
{
  clearTimer(currentTimer);
}

float getTimer(TTimers currentTimer = T1, const tMovementUnits unitType = seconds)
{
  if(unitType == milliseconds)
    return (time1[currentTimer]);
  else if(unitType == seconds)
    return ((float)time1[currentTimer]/1000.0);
  else if(unitType == minutes)
    return (((float)time1[currentTimer] / 1000.0) / 60.0);

  return 0.0;
}
```