```
#include "KerbalSimpit.h"
#include "KerbalSimpitMessageTypes.h"
#include "PayloadStructs.h"

rotationMessage myRotation;                 //Newly added 6/28
translationMessage myTranslation;           //Newly added 6/28

//Arduino Due has 54 "Digital Input/Output (I/O)" Pins

#define IOPIN0 0      // ~RX 0
#define IOPIN1 1      // ~TX 0
#define IOPIN2 2      // STAGE
#define IOPIN3 3      // ABORT
#define IOPIN4 4      // LIGHTS
#define IOPIN5 5      // GEAR
#define IOPIN6 6      // BRAKES
#define IOPIN7 7      // RCS
#define IOPIN8 8      // SAS
#define IOPIN9 9
#define IOPIN10 10
#define IOPIN11 11
#define IOPIN12 12
#define IOPIN13 13    // ~Builtin LED light
#define IOPIN14 14    // ~TX 3
#define IOPIN15 15    // ~RX 3
#define IOPIN16 16    // ~TX 2
#define IOPIN17 17    // ~RX 2
#define IOPIN18 18    // ~TX 1
#define IOPIN19 19    // ~RX 1
#define IOPIN20 20    // ~SDA
#define IOPIN21 21    // ~SCL
#define IOPIN22 22    // STAGE LED Indicator light
#define IOPIN23 23    // LIGHT LED indicator light
#define IOPIN24 24    // GEAR LED indicator light
#define IOPIN25 25    // BRAKES LED indicator light
#define IOPIN26 26    // CUSTOM ACTION GROUP 1 (CAG1)
#define IOPIN27 27    // CUSTOM ACTION GROUP 2 (CAG2)
#define IOPIN28 28    // CUSTOM ACTION GROUP 3 (CAG3)
#define IOPIN29 29    // CUSTOM ACTION GROUP 4 (CAG4)
#define IOPIN30 30    // CUSTOM ACTION GROUP 5 (CAG5)
#define IOPIN31 31    // CUSTOM ACTION GROUP 6 (CAG6)
#define IOPIN32 32    // CUSTOM ACTION GROUP 7 (CAG7)
#define IOPIN33 33    // CUSTOM ACTION GROUP 8 (CAG8)
#define IOPIN34 34    // CUSTOM ACTION GROUP 9 (CAG9)
#define IOPIN35 35    // CUSTOM ACTION GROUP 10 (CAG10)
#define IOPIN36 36    // Stability Assist (Necessary?)
#define IOPIN37 37    // Prograde
#define IOPIN38 38    // Retrograde
#define IOPIN39 39    // Normal
#define IOPIN40 40    // Anti-Normal
#define IOPIN41 41    // Radial In
#define IOPIN42 42    // Radial Out
#define IOPIN43 43    // Target
#define IOPIN44 44    // Anti-Target
#define IPPIN45 45    // Maneuver
#define IOPIN46 46
#define IOPIN47 47
#define IOPIN48 48
#define IOPIN49 49
#define IOPIN50 50
#define IOPIN51 51
```

```
#define IOPIN52 52
#define IOPIN53 53
#define IOPIN54 54

//Arduino Due has 12 "Analog In" Pins

#define ANALPIN0 A0    // the THROTTLE potentiometer
#define ANALPIN1 A1    // the X ROTATION potentiometer
#define ANALPIN2 A2    // the Y ROTATION potentiometer
#define ANALPIN3 A3    // the Z Rotation potentiometer
#define ANALPIN4 A4    // the X TRANSLATION potentiometer
#define ANALPIN5 A5    // the Y TRANSLATION potentiometer
#define ANALPIN6 A6    // the Z TRANSLATION potentiometer
#define ANALPIN7 A7
#define ANALPIN8 A8
#define ANALPIN9 A9
#define ANALPIN10 A10
#define ANALPIN11 A11

const int stagePin = IOPIN2;              // the STAGE button
const int abortPin = IOPIN3;              // the ABORT button
const int lightsPin = IOPIN4;             // the LIGHTS switch
const int gearPin = IOPIN5;               // the GEAR switch
const int brakesPin = IOPIN6;             // the BRAKE switch
const int rcsPin = IOPIN7;                // the RCS switch
const int sasPin = IOPIN8;                // the SAS switch
const int cag1Pin = IOPIN26;              // the CUSTOM ACTION GROUP 1 button
const int cag2Pin = IOPIN27;              // the CUSTOM ACTION GROUP 2 button
const int cag3Pin = IOPIN28;              // the CUSTOM ACTION GROUP 3 button
const int cag4Pin = IOPIN29;              // the CUSTOM ACTION GROUP 4 button
const int cag5Pin = IOPIN30;              // the CUSTOM ACTION GROUP 5 button
const int cag6Pin = IOPIN31;              // the CUSTOM ACTION GROUP 6 button
const int cag7Pin = IOPIN32;              // the CUSTOM ACTION GROUP 7 button
const int cag8Pin = IOPIN33;              // the CUSTOM ACTION GROUP 8 button
const int cag9Pin = IOPIN34;              // the CUSTOM ACTION GROUP 9 button
const int cag10Pin = IOPIN35;             // the CUSTOM ACTION GROUP 10 button

const int stageLedPin = IOPIN22;          // the STAGE LED pin
const int lightsLedPin = IOPIN23;         // the LIGHTS LED pin
const int gearLedPin = IOPIN24;           // the GEAR LED pin
const int brakesLedPin = IOPIN25;         // the BRAKE LED pin

const int throttlePin = ANALPIN0;         // the THROTTLE potentiometer
const int rotationX = ANALPIN1;           // the X ROTATION potentiometer
const int rotationY = ANALPIN2;           // the Y ROTATION potentiometer
const int rotationZ = ANALPIN3;           // the Z Rotation potentiometer
const int translationX = ANALPIN4;        // the X TRANSLATION potentiometer
const int translationY = ANALPIN5;        // the Y TRANSLATION potentiometer
const int translationZ = ANALPIN6;        // the Z TRANSLATION potentiometer


int ledState = HIGH;                      // the current state of the output pin
int stageButtonState;                     // the current reading from the STAGE input pin
int abortButtonState;                     // the current reading from the ABORT input pin
int lightsButtonState;                    // the current reading from the LIGHTS input pin
int gearButtonState;                      // the current reading from the GEAR input pin
int brakesButtonState;                    // the current reading from the BRAKE input pin
int rcsButtonState;                       // the current reading from the RCS input pin
int sasButtonState;                       // the current reading from the SAS input pin
int cag1ButtonState;                      // the current reading from the CUSTOM ACTION GROUP 1 input pin
int cag2ButtonState;                      // the current reading from the CUSTOM ACTION GROUP 2 input pin
```

```
int cag3ButtonState;              // the current reading from the CUSTOM ACTION GROUP 3 input pin
int cag4ButtonState;              // the current reading from the CUSTOM ACTION GROUP 4 input pin
int cag5ButtonState;              // the current reading from the CUSTOM ACTION GROUP 5 input pin
int cag6ButtonState;              // the current reading from the CUSTOM ACTION GROUP 6 input pin
int cag7ButtonState;              // the current reading from the CUSTOM ACTION GROUP 7 input pin
int cag8ButtonState;              // the current reading from the CUSTOM ACTION GROUP 8 input pin
int cag9ButtonState;              // the current reading from the CUSTOM ACTION GROUP 9 input pin
int cag10ButtonState;             // the current reading from the CUSTOM ACTION GROUP 10 input pin

int throttle;                     // the current reading from the THROTTLE potentiometer
int rotationX_Read;               // the current reading from the X-AXIS ROTATION potentiometer
int rotationY_Read;               // the current reading from the Y-AXIS ROTATION potentiometer
int rotationZ_Read;               // the current reading from the Z-AXIS ROTATION potentiometer
int rotationX_Mapped;
int rotationY_Mapped;
int rotationZ_Mapped;
int translationX_Read;            // the current reading from the X-AXIS ROTATION potentiometer
int translationY_Read;            // the current reading from the X-AXIS ROTATION potentiometer
int translationZ_Read;            // the current reading from the X-AXIS ROTATION potentiometer
int translationX_Mapped;
int translationY_Mapped;
int translationZ_Mapped;

int lastledState = HIGH;          // the previous state of the output pin
int lastStageButtonState = LOW;   // the previous reading from the STAGE input pin
int lastAbortButtonState = LOW;   // the previous reading from the ABORT input pin
int lastLightsButtonState = LOW;  // the previous reading from the LIGHTS input pin
int lastGearButtonState;          // the previous reading from the GEAR input pin
int lastBrakesButtonState = LOW;  // the previous reading from the BRAKE input pin
int lastRCSButtonState = LOW;     // the previous reading from the RCS input pin
int lastSASButtonState = LOW;     // the previous reading from the SAS input pin
int lastCag1ButtonState = LOW;    // the previous reading from the CUSTOM ACTION GROUP 1 input pin
int lastCag2ButtonState = LOW;    // the previous reading from the CUSTOM ACTION GROUP 2 input pin
int lastCag3ButtonState = LOW;    // the previous reading from the CUSTOM ACTION GROUP 3 input pin
int lastCag4ButtonState = LOW;    // the previous reading from the CUSTOM ACTION GROUP 4 input pin
int lastCag5ButtonState = LOW;    // the previous reading from the CUSTOM ACTION GROUP 5 input pin
int lastCag6ButtonState = LOW;    // the previous reading from the CUSTOM ACTION GROUP 6 input pin
int lastCag7ButtonState = LOW;    // the previous reading from the CUSTOM ACTION GROUP 7 input pin
int lastCag8ButtonState = LOW;    // the previous reading from the CUSTOM ACTION GROUP 8 input pin
int lastCag9ButtonState = LOW;    // the previous reading from the CUSTOM ACTION GROUP 9 input pin
int lastCag10ButtonState = LOW;   // the previous reading from the CUSTOM ACTION GROUP 10 input pin

/* the following variables are unsigned long's because the time, measured
in miliseconds, will quickly become a bigger number than can be stored
in an int. */

unsigned long lastStageDebounceTime = 0;  // the last time the STAGE pin was toggled
unsigned long lastAbortDebounceTime = 0;  // the last time the ABORT pin was toggled
unsigned long lastLightsDebounceTime = 0; // the last time the LIGHTS pin was toggled
unsigned long lastGearDebounceTime = 0;   // the last time the GEAR pin was toggled
unsigned long lastBrakesDebounceTime = 0; // the last time the BRAKE pin was toggled
unsigned long lastRCSDebounceTime = 0;    // the last time the RCS pin was toggled
unsigned long lastSASDebounceTime = 0;    // the last time the SAS pin was toggled
unsigned long lastCag1DebounceTime = 0;   // the last time the CUSTOM ACTION GROUP 1 pin was toggled
unsigned long lastCag2DebounceTime = 0;   // the last time the CUSTOM ACTION GROUP 2 pin was toggled
unsigned long lastCag3DebounceTime = 0;   // the last time the CUSTOM ACTION GROUP 3 pin was toggled
unsigned long lastCag4DebounceTime = 0;   // the last time the CUSTOM ACTION GROUP 4 pin was toggled
unsigned long lastCag5DebounceTime = 0;   // the last time the CUSTOM ACTION GROUP 5 pin was toggled
unsigned long lastCag6DebounceTime = 0;   // the last time the CUSTOM ACTION GROUP 6 pin was toggled
unsigned long lastCag7DebounceTime = 0;   // the last time the CUSTOM ACTION GROUP 7 pin was toggled
unsigned long lastCag8DebounceTime = 0;   // the last time the CUSTOM ACTION GROUP 8 pin was toggled
```

```
unsigned long lastCag9DebounceTime = 0;   // the last time the CUSTOM ACTION GROUP 9 pin was toggled
unsigned long lastCag10DebounceTime = 0;  // the last time the CUSTOM ACTION GROUP 10 pin was toggled

unsigned long stageDebounceDelay = 50;      // the STAGE debounce time; increase if the output flickers
unsigned long abortDebounceDelay = 50;      // the ABORT debounce time; increase if the output flickers
unsigned long lightsDebounceDelay = 50;     // the LIGHTS debounce time; increase if the output flickers
unsigned long gearDebounceDelay = 50;       // the GEAR debounce time; increase if the output flickers
unsigned long brakesDebounceDelay = 50;   // the BRAKE debounce time; increase if the output flickers
unsigned long rcsDebounceDelay = 50;        // the RCS debounce time; increase if the output flickers
unsigned long sasDebounceDelay = 50;        // the SAS debounce time; increase if the output flickers
unsigned long cag1DebounceDelay = 50;     // the CUSTOM ACTION GROUP 1 debounce time; increase if the output flickers
unsigned long cag2DebounceDelay = 50;     // the CUSTOM ACTION GROUP 2 debounce time; increase if the output flickers
unsigned long cag3DebounceDelay = 50;     // the CUSTOM ACTION GROUP 3 debounce time; increase if the output flickers
unsigned long cag4DebounceDelay = 50;     // the CUSTOM ACTION GROUP 4 debounce time; increase if the output flickers
unsigned long cag5DebounceDelay = 50;     // the CUSTOM ACTION GROUP 5 debounce time; increase if the output flickers
unsigned long cag6DebounceDelay = 50;     // the CUSTOM ACTION GROUP 6 debounce time; increase if the output flickers
unsigned long cag7DebounceDelay = 50;     // the CUSTOM ACTION GROUP 7 debounce time; increase if the output flickers
unsigned long cag8DebounceDelay = 50;     // the CUSTOM ACTION GROUP 8 debounce time; increase if the output flickers
unsigned long cag9DebounceDelay = 50;     // the CUSTOM ACTION GROUP 9 debounce time; increase if the output flickers
unsigned long cag10DebounceDelay = 50;    // the CUSTOM ACTION GROUP 10 debounce time; increase if the output flickers

KerbalSimpit mySimpit(Serial);          // Declare a KerbalSimpit object that will communicate using the "Serial" device.

void setup() {
  Serial.begin(115200);                 // Open the serial connection.

  // Set initial pin states, and turn on the LED
  pinMode(LED_BUILTIN, OUTPUT);

  pinMode(stagePin, INPUT);
  pinMode(abortPin, INPUT);
  pinMode(lightsPin, INPUT);
  pinMode(gearPin, INPUT);
  pinMode(brakesPin, INPUT);
  pinMode(rcsPin, INPUT);
  pinMode(sasPin, INPUT);

  pinMode(cag1Pin, INPUT);
  pinMode(cag2Pin, INPUT);
  pinMode(cag3Pin, INPUT);
  pinMode(cag4Pin, INPUT);
  pinMode(cag5Pin, INPUT);
  pinMode(cag6Pin, INPUT);
  pinMode(cag7Pin, INPUT);
  pinMode(cag8Pin, INPUT);
  pinMode(cag9Pin, INPUT);
  pinMode(cag10Pin, INPUT);

  pinMode(throttlePin, INPUT);
  pinMode(rotationX, INPUT);
  pinMode(rotationY, INPUT);
  pinMode(rotationZ, INPUT);
  pinMode(translationX, INPUT);
  pinMode(translationY, INPUT);
  pinMode(translationZ, INPUT);

  digitalWrite(stageLedPin, HIGH);        //Should we turn this off after handshake?
  digitalWrite(lightsLedPin, HIGH);       //Should we turn this off after handshake?
  digitalWrite(gearLedPin, HIGH);         //Should we turn this off after handshake?
  digitalWrite(brakesLedPin, HIGH);       //Should we turn this off after handshake?
```

```
/*This loop continually attempts to handshake with the plugin.
It will keep retrying until it gets a successful handshake.*/

  while (!mySimpit.init()) {
    delay(100);
  }
  digitalWrite(LED_BUILTIN, LOW);          // Turn off the built-in LED to indicate handshaking is complete.

  mySimpit.registerChannel(THROTTLE_MESSAGE);
  mySimpit.registerChannel(ROTATION_MESSAGE);
  mySimpit.registerChannel(TRANSLATION_MESSAGE);
}


void loop() {

  //****************************************** STAGE ******************************************
  int stageReading = digitalRead(stagePin);            // Read the state of the stage button into a local variable.

  /*check to see if you just pressed the stage button
  (i.e. the input went from LOW to HIGH) and you've waited
  long enough since the last press to ignore any noise:*/

  if (stageReading != lastStageButtonState) {          // If the switch changed, due to noise or pressing:
    lastStageDebounceTime = millis();                  // Reset the debouncing timer.
  }
  if ((millis() - lastStageDebounceTime) > stageDebounceDelay) {

    /* whatever the reading is at, it's been there for longer
       than the debounce delay, so take it as the actual current state:*/

    if (stageReading != stageButtonState) {            // If the button state has changed:
      stageButtonState = stageReading;
      if (stageButtonState == HIGH) {                  // If the new button state is HIGH, that means the button has just been pressed.
        mySimpit.activateAction(STAGE_ACTION);         // Send a message to the plugin activating the Stage action group. The plugin
will then activate the next stage.
      }
    }
  }
   digitalWrite(stageLedPin, stageButtonState);        // Set the LED to match the state of the button.
   lastStageButtonState = stageReading;                // Save the reading.  Next time through the loop, it'll be the lastButtonState.


  //****************************************** ABORT ******************************************
  int abortReading = digitalRead(abortPin);

  if (abortReading != lastAbortButtonState) {
    lastAbortDebounceTime = millis();
  }
  if ((millis() - lastAbortDebounceTime) > abortDebounceDelay) {

    if (abortReading != abortButtonState) {
      abortButtonState = abortReading;
      if (abortButtonState == HIGH) {
        mySimpit.activateAction(ABORT_ACTION);
      }else{
        mySimpit.deactivateAction(ABORT_ACTION);
      }
    }
  }
   lastAbortButtonState = abortReading;
```

```
//************************************** LIGHTS ***************************************
int lightsReading = digitalRead(lightsPin);
if (lightsReading != lastLightsButtonState) {
  lastLightsDebounceTime = millis();
}
if ((millis() - lastLightsDebounceTime) > lightsDebounceDelay) {

  if (lightsReading != lightsButtonState) {
    lightsButtonState = lightsReading;
    if (lightsButtonState == HIGH) {
      mySimpit.activateAction(LIGHT_ACTION);

    }else{
      mySimpit.deactivateAction(LIGHT_ACTION);
    }
  }
}
  digitalWrite(lightsLedPin, lightsButtonState);
  lastLightsButtonState = lightsReading;


//************************************** GEAR ***************************************
int gearReading = digitalRead(gearPin);
if (gearReading != lastGearButtonState) {
  lastGearDebounceTime = millis();
}
if ((millis() - lastGearDebounceTime) > gearDebounceDelay) {

  if (gearReading != gearButtonState) {
    gearButtonState = gearReading;
    if (gearButtonState == HIGH) {
      mySimpit.toggleAction(GEAR_ACTION);
    }
  }
}
  digitalWrite(gearLedPin, gearButtonState);
  lastGearButtonState = gearReading;


//************************************** BRAKES ***************************************
int brakesReading = digitalRead(brakesPin);
if (brakesReading != lastBrakesButtonState) {
  lastBrakesDebounceTime = millis();
}
if ((millis() - lastBrakesDebounceTime) > brakesDebounceDelay) {

  if (brakesReading != brakesButtonState) {
    brakesButtonState = brakesReading;
    if (brakesButtonState == HIGH) {
      mySimpit.toggleAction(BRAKES_ACTION);
    }
  }
}
  digitalWrite(brakesLedPin, brakesButtonState);
  lastBrakesButtonState = brakesReading;


//************************************** RCS ***************************************
 int rcsReading = digitalRead(rcsPin);
```

```
 if (rcsReading != lastRCSButtonState) {
  lastRCSDebounceTime = millis();
 }
 if ((millis() - lastRCSDebounceTime) > rcsDebounceDelay) {

  if (rcsReading != rcsButtonState) {
   rcsButtonState = rcsReading;
   if (rcsButtonState == HIGH) {
    mySimpit.activateAction(RCS_ACTION);

   }else{
    mySimpit.deactivateAction(RCS_ACTION);
   }
  }
 }
  lastRCSButtonState = rcsReading;


//************************************** SAS ***************************************
 int sasReading = digitalRead(sasPin);
 if (sasReading != lastSASButtonState) {
  lastSASDebounceTime = millis();
 }
 if ((millis() - lastSASDebounceTime) > sasDebounceDelay) {

  if (sasReading != sasButtonState) {
   sasButtonState = sasReading;
   if (sasButtonState == HIGH) {
    mySimpit.activateAction(SAS_ACTION);

   }else{
    mySimpit.deactivateAction(SAS_ACTION);
   }
  }
 }
  lastSASButtonState = sasReading;


//************************************** CUSTOM ACTION GROUP 1 ***************************************
 int cag1Reading = digitalRead(cag1Pin);
 if (cag1Reading != lastCag1ButtonState) {
  lastCag1DebounceTime = millis();
 }
 if ((millis() - lastCag1DebounceTime) > cag1DebounceDelay) {

  if (cag1Reading != cag1ButtonState) {
   cag1ButtonState = cag1Reading;
   if (cag1ButtonState == HIGH) {
    mySimpit.activateCAG(1);
   }else{
    mySimpit.deactivateCAG(1);
   }
  }
 }
  lastCag1ButtonState = cag1Reading;


//************************************** CUSTOM ACTION GROUP 2 ***************************************
 int cag2Reading = digitalRead(cag2Pin);
 if (cag2Reading != lastCag2ButtonState) {
  lastCag2DebounceTime = millis();
```

```
}
if ((millis() - lastCag2DebounceTime) > cag2DebounceDelay) {

  if (cag2Reading != cag2ButtonState) {
    cag2ButtonState = cag2Reading;
    if (cag2ButtonState == HIGH) {
      mySimpit.activateCAG(2);
    }else{
      mySimpit.deactivateCAG(2);
    }
  }
}
  lastCag2ButtonState = cag2Reading;

  //****************************************** CUSTOM ACTION GROUP 3 ******************************************
int cag3Reading = digitalRead(cag3Pin);
if (cag3Reading != lastCag3ButtonState) {
  lastCag3DebounceTime = millis();
}
if ((millis() - lastCag3DebounceTime) > cag3DebounceDelay) {

  if (cag3Reading != cag3ButtonState) {
    cag3ButtonState = cag3Reading;
    if (cag3ButtonState == HIGH) {
      mySimpit.activateCAG(3);
    }else{
      mySimpit.deactivateCAG(3);
    }
  }
}
  lastCag3ButtonState = cag3Reading;

  //****************************************** CUSTOM ACTION GROUP 4 ******************************************
int cag4Reading = digitalRead(cag4Pin);
if (cag4Reading != lastCag4ButtonState) {
  lastCag4DebounceTime = millis();
}
if ((millis() - lastCag4DebounceTime) > cag4DebounceDelay) {

  if (cag4Reading != cag4ButtonState) {
    cag4ButtonState = cag4Reading;
    if (cag4ButtonState == HIGH) {
      mySimpit.activateCAG(4);
    }else{
      mySimpit.deactivateCAG(4);
    }
  }
}
  lastCag4ButtonState = cag4Reading;

  //****************************************** CUSTOM ACTION GROUP 5 ******************************************
int cag5Reading = digitalRead(cag5Pin);
if (cag5Reading != lastCag5ButtonState) {
  lastCag5DebounceTime = millis();
}
if ((millis() - lastCag5DebounceTime) > cag5DebounceDelay) {

  if (cag5Reading != cag5ButtonState) {
    cag5ButtonState = cag5Reading;
    if (cag5ButtonState == HIGH) {
      mySimpit.activateCAG(5);
```

```
    }else{
      mySimpit.deactivateCAG(5);
    }
  }
}
  lastCag5ButtonState = cag5Reading;

  //************************************** CUSTOM ACTION GROUP 6 **************************************
int cag6Reading = digitalRead(cag6Pin);
if (cag6Reading != lastCag6ButtonState) {
  lastCag6DebounceTime = millis();
}
if ((millis() - lastCag6DebounceTime) > cag6DebounceDelay) {

  if (cag6Reading != cag6ButtonState) {
    cag6ButtonState = cag6Reading;
    if (cag6ButtonState == HIGH) {
      mySimpit.activateCAG(6);
    }else{
      mySimpit.deactivateCAG(6);
    }
  }
}
  lastCag6ButtonState = cag6Reading;

  //************************************** CUSTOM ACTION GROUP 7 **************************************
int cag7Reading = digitalRead(cag7Pin);
if (cag7Reading != lastCag7ButtonState) {
  lastCag7DebounceTime = millis();
}
if ((millis() - lastCag7DebounceTime) > cag7DebounceDelay) {

  if (cag7Reading != cag7ButtonState) {
    cag7ButtonState = cag7Reading;
    if (cag7ButtonState == HIGH) {
      mySimpit.activateCAG(7);
    }else{
      mySimpit.deactivateCAG(7);
    }
  }
}
  lastCag7ButtonState = cag7Reading;

  //************************************** CUSTOM ACTION GROUP 8 **************************************
int cag8Reading = digitalRead(cag8Pin);
if (cag8Reading != lastCag8ButtonState) {
  lastCag8DebounceTime = millis();
}
if ((millis() - lastCag8DebounceTime) > cag8DebounceDelay) {

  if (cag8Reading != cag8ButtonState) {
    cag8ButtonState = cag8Reading;
    if (cag8ButtonState == HIGH) {
      mySimpit.activateCAG(8);
    }else{
      mySimpit.deactivateCAG(8);
    }
  }
}
  lastCag8ButtonState = cag8Reading;
```

```
  //************************************** CUSTOM ACTION GROUP 9 **************************************
int cag9Reading = digitalRead(cag9Pin);
if (cag9Reading != lastCag9ButtonState) {
  lastCag9DebounceTime = millis();
}
if ((millis() - lastCag9DebounceTime) > cag9DebounceDelay) {

  if (cag9Reading != cag9ButtonState) {
    cag9ButtonState = cag9Reading;
    if (cag9ButtonState == HIGH) {
      mySimpit.activateCAG(9);
    }else{
      mySimpit.deactivateCAG(9);
    }
  }
}
  lastCag9ButtonState = cag9Reading;

  //************************************** CUSTOM ACTION GROUP 10 **************************************
int cag10Reading = digitalRead(cag10Pin);
if (cag10Reading != lastCag10ButtonState) {
  lastCag10DebounceTime = millis();
}
if ((millis() - lastCag10DebounceTime) > cag10DebounceDelay) {

  if (cag10Reading != cag10ButtonState) {
    cag10ButtonState = cag10Reading;
    if (cag10ButtonState == HIGH) {
      mySimpit.activateCAG(10);
    }else{
      mySimpit.deactivateCAG(10);
    }
  }
}
  lastCag10ButtonState = cag10Reading;

  //************************************** THROTTLE **************************************

int throttleRead = analogRead(A0);
throttle = map(throttleRead, 0, 1023, 0, 32767);
mySimpit.send(THROTTLE_MESSAGE, throttle);

  //************************************** ROTATION **************************************

int rotationX_Read = analogRead(rotationX);

  // takes a reading of the X-axis rotation potentiometer

if (rotationX_Read < 510 && rotationX_Read > 500) {
  rotationX_Mapped = 0;
}

// if the reading comes back in the dead band (no
// X-axis movement detected on JS): no X-axis command sent

/* NOTE: We may have to figure out these values on our own for
        our JS. We may be able to use the values we got
        previously */

if (rotationX_Read <= 500) {
  rotationX_Mapped = map(rotationX_Read, 330, 500, -32768, 0);
```

```
}

/* if reading is below dead band: signal is mapped
(assigned a value to be used by the microcontroller)
this scales the pot reading (330 to 500) to a value usable by the
microcontroller (-32768 to 0 . . . because this is on the negative side)*/

if (rotationX_Read >= 510) {
  rotationX_Mapped = map(rotationX_Read, 510, 693, 0, 32767);
}

/* if reading is above dead band: signal is mapped
(assigned a value to be used by the microcontroller)
this scales the pot reading (510 to 693) to a value usable by the
microcontroller (0 to 32767 . . . because this is on the positive side)*/

int rotationY_Read = analogRead(rotationY);

if (rotationY_Read < 518 && rotationY_Read > 508) {
  rotationY_Mapped = 0;
}

if (rotationY_Read <= 508) {
  rotationY_Mapped = map(rotationY_Read, 344, 508, -32768, 0);
}

if (rotationY_Read >= 518) {
  rotationY_Mapped = map(rotationY_Read, 518, 680, 0, 32767);
}

int rotationZ_Read = analogRead(rotationZ);

if (rotationZ_Read < 520 && rotationZ_Read > 510) {
  rotationZ_Mapped = 0;
}

if (rotationZ_Read <= 510) {
  rotationZ_Mapped = map(rotationZ_Read, 296, 510, -32768, 0);
}

if (rotationZ_Read >= 520) {
  rotationZ_Mapped = map(rotationZ_Read, 520, 733, 0, 32767);
}

myRotation.mask = 1|2|4;

myRotation.pitch = rotationY_Mapped;
myRotation.roll = rotationX_Mapped;
myRotation.yaw = rotationZ_Mapped;

mySimpit.send(ROTATION_MESSAGE, myRotation);

   //************************************** TRANSLATION ****************************************

int translationX_Read = analogRead(translationX);

  // takes a reading of the X-axis rotation potentiometer

if (translationX_Read < 510 && translationX_Read > 500) {
  translationX_Mapped = 0;
}
```

```
// if the reading comes back in the dead band (no
// X-axis movement detected on JS): no X-axis command sent

/* NOTE: We may have to figure out these values on our own for
        our JS. We may be able to use the values we got
        previously */

if (translationX_Read <= 500) {
  translationX_Mapped = map(translationX_Read, 330, 500, -32768, 0);
}

/* if reading is below dead band: signal is mapped
(assigned a value to be used by the microcontroller)
this scales the pot reading (330 to 500) to a value usable by the
microcontroller (-32768 to 0 . . . because this is on the negative side)*/

if (translationX_Read >= 510) {
  translationX_Mapped = map(translationX_Read, 510, 693, 0, 32767);
}

/* if reading is above dead band: signal is mapped
(assigned a value to be used by the microcontroller)
this scales the pot reading (510 to 693) to a value usable by the
microcontroller (0 to 32767 . . . because this is on the positive side)*/

int translationY_Read = analogRead(translationY);

if (translationY_Read < 518 && translationY_Read > 508) {
  translationY_Mapped = 0;
}

if (translationY_Read <= 508) {
  translationY_Mapped = map(translationY_Read, 344, 508, -32768, 0);
}

if (translationY_Read >= 518) {
  translationY_Mapped = map(translationY_Read, 518, 680, 0, 32767);
}

int translationZ_Read = analogRead(translationZ);

if (translationZ_Read < 520 && translationZ_Read > 510) {
  translationZ_Mapped = 0;
}

if (translationZ_Read <= 510) {
  translationZ_Mapped = map(translationZ_Read, 296, 510, -32768, 0);
}

if (translationZ_Read >= 520) {
  translationZ_Mapped = map(translationZ_Read, 520, 733, 0, 32767);
}

myTranslation.mask = 1|2|4;

myTranslation.X = translationY_Mapped;
myTranslation.Y = translationX_Mapped;
myTranslation.Z = translationZ_Mapped;

mySimpit.send(TRANSLATION_MESSAGE, myTranslation);
```

```
}
```