```
//  Model railroad traverser control code
//  by Michael Hardwick
//  December 9, 2016

// Define global constants
// Define pins numbers
const int stepPin = 3;    // Connected to the A4988 step pin
const int dirPin = 2;     // Connected to the A4988 dir pin
const int sensorPin = 13; // Connected to the optical sensor signal pin
const int pushPin[6] = {A0, A1, A2, A3, A4, A5}; // Pushbutton pins
const int enablePin = 7;  // Connected to the A4988 enable pin

// Define the microstepping pins (don't need these if the A4988 is
// hardwired in a specific microstepping mode)
const int MS1Pin = 6;    // Connected to the A4988 MS1 pin
const int MS2Pin = 5;    // Connected to the A4988 MS2 pin
const int MS3Pin = 4;    // Connected to the A4988 MS3 pin

// Set other constants
const int towardHome = 0;   // One traverser direction
const int awayFromHome = 1; // The other traverser direction
// Set constants for the positions of each staging track.
// This array holds the number of motor steps from the home position
// for 1/16 microstepping and need to be determined by trial and error.
const int trackPos16[5] = {12016, 9820, 7460, 2616, 112};
const int speedFactor = 5;  // A value between 1 and 10 that determines the
                            // traverser speed. Used to calculate speedDelay16
// Define global variables
int MSMode;           // The microstepping mode (1, 2, 4, 8, or 16)
int speedDelay16;     // The delay (usec) value used for 1/16 microstepping
int speedDelay;       // The delay (usec) for the selected microstepping mode
                      // based on the value of speedDelay16
int trackPos[5] = {0, 0, 0, 0, 0}; // Track positions calculated for
                                   // each microstepping mode
int traverserPos = 0; // Traverser position in steps from home

void setup() {
  Serial.begin(9600); // Enable serial communication for debugging
  Serial.println("Begin setup...");
  // Set up the six push button pins (0 - 5) using pullup resistors
  for (int i=0; i<6; i++) {
    pinMode(pushPin[i], INPUT_PULLUP);
  }
  pinMode(sensorPin, INPUT);
  pinMode(enablePin, OUTPUT);
  pinMode(stepPin, OUTPUT);
  pinMode(dirPin, OUTPUT);
  pinMode(MS1Pin, OUTPUT);
  pinMode(MS2Pin, OUTPUT);
  pinMode(MS3Pin, OUTPUT);
  // Map the speedFactor to a delay in usec for 1/16 stepping mode
  speedDelay16 = map(speedFactor, 1, 10, 2000, 500);
  setSteppingMode(8); // Select 1/8 stepping mode
  digitalWrite(enablePin, LOW); // enable the FET outputs
  moveToHome(); // Find home when the processor resets
}

void loop() {
  // Poll for button presses. If a button is pressed, go to that track
  // or go home if the home button is pressed

  int buttonNumber = pollButtons(); // returns the button number (0-5)
```

```
  if (buttonNumber == 5) { // HOME button was pressed
      Serial.println("Moving to HOME.");
      moveToHome();
  }
  else if (buttonNumber >= 0) {   // Track position button was pressed
    Serial.print("Moving to position ");
    Serial.println(buttonNumber);
    moveToPosition(buttonNumber); // Move to the selected track.
  }
}

int pollButtons() {
  // Returns the index of the button pushed, -1 if no button pushed.
  // The push buttons must be normally open and connected to ground. They
  // must use the internal Arduino pullup resistors (or external pullup resistors,
  // ~10kohm tied to +5V).
  // With high impedance inputs, I had to worry about spurious signals on the
  // pushbutton wires. A simple way to deal with them is to ensure the button
  // is pressed for some period of time longer than the transient, say 100 ms.

  //     TRK1  TRK2  TRK3  TRK4  TRK5  HOME
  // ---------------------------------------
  // |     0     1     2     3     4     5     | <--(Button index)
  // ---------------------------------------

  // Look at each button. If it reads LOW (grounded), it is pressed, so
  // return the button index.
  for (int i=0; i<6; i++) {  // cycle through the buttons
    if (!digitalRead(pushPin[i])) {  // look closer if the button appears pushed
      int counter = 0; // set a counter to zero
      while (!digitalRead(pushPin[i])) { // check the button every 10 ms for 100 ms
        if (counter == 10){  // if the button was checked 10 times, it is pressed
          return i;  // return the button index
        }
        counter++; // increment the counter
        delay(10); // delay 10 milliseconds
      }
    }
  } // End of button loop
  return -1; // No button was pressed, so return -1.
}

void moveToHome() {
  // This routine moves the traverser to the home position in two phases.
  // First, it moves the traverser to the sensor at normal speed. Then it
  // backs it away slightly at normal speed and slowly reapproaches until the
  // sensor is tripped. The slow approach should result in a more repeatable
  // sensor trip point.

  if(!digitalRead(sensorPin)) { // If the sensor is not tripped...
    // move toward HOME a step at a time at normal speed, checking the sensor
    // each time.
    while(!digitalRead(sensorPin)){ // While the sensor is not tripped...
      moveXSteps(towardHome, 1, speedDelay); // Move one step toward home
    }
  }
  // Sensor has now reached home at normal speed. Now back away a bit at
  // normal speed, then reapproach at slow speed.
  moveXSteps(awayFromHome, 50 * MSMode, speedDelay); // '50 * MSMode' is a
  // little less than 1/2 inch.
  while(!digitalRead(sensorPin)){ // While the sensor is not tripped...
    moveXSteps(towardHome, 1, speedDelay * 5); // five times slower
```

```
  }
  Serial.println("Traverser is at HOME position.");
  traverserPos = 0; // At home, so reset the traverser position to zero
}

void moveXSteps(int dir, int numSteps, int delay_usec){
  // This routine sends 'numSteps' pulses to the A4988 board and keeps
  // track of the traverser position.

  digitalWrite(dirPin, dir); // Set the direction for the move
  for(int i=0; i<numSteps; i++) {
    digitalWrite(stepPin, HIGH);   // Beginning of pulse to A4988
    delayMicroseconds(10);         // Can be as low as one microsecond
    digitalWrite(stepPin, LOW);    // End of pulse
    if (dir) {traverserPos++;}     // Update the traverser position
    else {traverserPos--;}
    delayMicroseconds(delay_usec); // this is a delay in microseconds.
  }
}

void moveToPosition(int posNum) {
  // This routine calculates the number of steps from the current
  // traverser position to the target position and calls 'moveXSteps()'
  // to move the traverser

  int dir = 0;
  int numSteps = 0;
  // determine direction to move
  if (trackPos[posNum] > traverserPos) {
    dir = awayFromHome;
    }
  else {
    dir = towardHome;
    }
  // determine number of steps to move
  numSteps = abs(traverserPos - trackPos[posNum]);
  // Now move
  moveXSteps(dir, numSteps, speedDelay);
  Serial.print("Traverser is at position ");
  Serial.println(posNum);
}

void setSteppingMode(int mode) {
  // This function lets you easily experiment with different microstepping
  // modes. Just change the parameter (1,2,4,8, or 16) to the call in setup().
  // The track positions and delay times are automatically scaled for you.

  //       1/16   1/8    1/4    1/2     1
  //     ---------------------------------
  // MS1 | HIGH   HIGH   LOW    HIGH   LOW
  // MS2 | HIGH   HIGH   HIGH   HIGH   LOW
  // MS3 | HIGH   LOW    LOW    LOW    LOW

  MSMode = mode; // set the global variable
  switch (mode)
{
  case 16:   // high  high  high
  digitalWrite(MS1Pin, HIGH);
  digitalWrite(MS2Pin, HIGH);
  digitalWrite(MS3Pin, HIGH);
  for(int i=0; i<6; i++) {
    trackPos[i] = trackPos16[i]; // update the track positions
```

```
  }
  speedDelay = speedDelay16;     // update the traverser speed
  break;
  case 8:    // high  high  low
  digitalWrite(MS1Pin, HIGH);
  digitalWrite(MS2Pin, HIGH);
  digitalWrite(MS3Pin, LOW);
  for(int i=0; i<6; i++) {
    trackPos[i] = trackPos16[i]/2; // update the track positions
  }
  speedDelay = speedDelay16 * 2;   // update the traverser speed
  break;
  case 4:    // low   high  low
  digitalWrite(MS1Pin, LOW);
  digitalWrite(MS2Pin, HIGH);
  digitalWrite(MS3Pin, LOW);
  for(int i=0; i<6; i++) {
    trackPos[i] = trackPos16[i]/4; // update the track positions
  }
  speedDelay = speedDelay16 * 4;   // update the traverser speed
  break;
  case 2:    // high  low   low
  digitalWrite(MS1Pin, HIGH);
  digitalWrite(MS2Pin, LOW);
  digitalWrite(MS3Pin, LOW);
  for(int i=0; i<6; i++) {
    trackPos[i] = trackPos16[i]/8; // update the track positions
  }
  speedDelay = speedDelay16 * 8;   // update the traverser speed
  break;
  case 1:    // low   low   low
  digitalWrite(MS1Pin, LOW);
  digitalWrite(MS2Pin, LOW);
  digitalWrite(MS3Pin, LOW);
  for(int i=0; i<6; i++) {
    trackPos[i] = trackPos16[i]/16; // update the track positions
  }
  speedDelay = speedDelay16 * 16;   // update the traverser speed
  break;
 }
}
```