Junior Design

Final Project

Arduino Software Documentation

Felipe Orrico Scognamiglio
Spydercam 18

Contents

Motor Control Firmware

Function

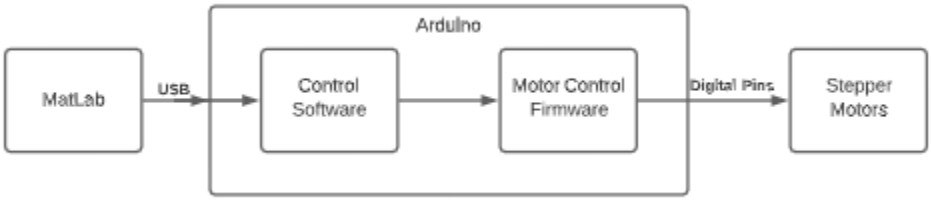

Figure 1: Motor Control Firmware Relationship to rest of Spydercam

The Motor Control Firmware (MCF) functions as an embedded method within the Control Software. As such, it shares the global values of the Control Software and updates them accordingly. The main purpose of the MCF is to appropriately calculate the speed, acceleration/deceleration, direction, and final location of the payload and relay those values to the AccelStepper library that has been customized to perform such tasks.

After the AccelStepper library has received the necessary values, it will initiate the payload movement. The MCF will then block until the payload reaches the desired target or an M6 or M2 command is executed. This happens so that there is no conflict of commands being sent to the block at the same time as it utilizes the same global values as the Control Software.

Currently, there is no implementation of a command queue, and as such, the program will not read any commands that are sent while the payload is in movement, unless it is an M2 or M6 command. For the purposes of letting MatLab know when to send the next command, the MCF will send a “G” whenever it has finished processing the command and is ready to receive again. There is also no implementation for micro-stepping. As this is an early version of the MCF, micro-stepping will only be implemented after the block has been validated with sufficient hardware testing.

Since the hardware implementation of the motor on side B is inverted, the value that is sent to the AccelStepper library is inverted so that it moves in the correct direction.

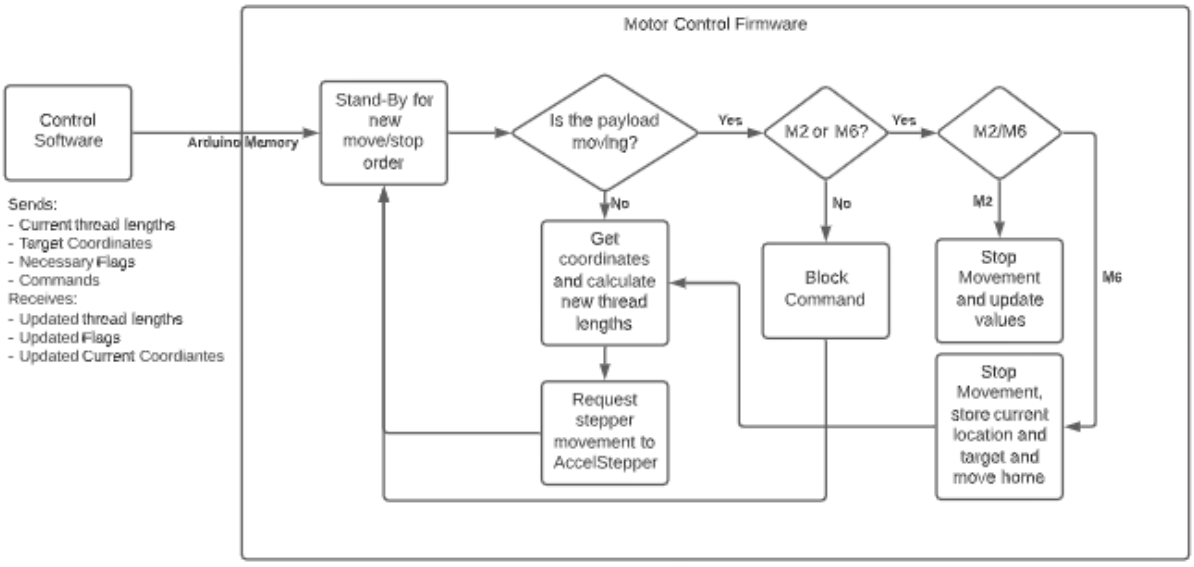

Figure 2: Motor Control Firmware Flow Chart of Operation

Outside of its main function, the MCF will also check if the current state of the payload. If the payload is in movement, it will block any commands that could affect the trajectory, and so, it will only accept M2 (Stop) or M6 (Tool Change) commands.

Variables

The Motor Control Firmware relies on accurate variables to operate correctly. These variables include but are not limited to:

- Current thread lengths.
- Current payload height.
- Target x/y/height locations.
- Motor percent speed as per requirement G-code G1.

With the use of the variables above and constants of the hardware, it is possible to make the necessary calculations that take the payload from point A to point B, or even change the payload height instead and keeping x and y constant.
The main method of this block is the “go” method. It receives the target coordinates for motion. All other values are calculated within the method, or are collected from global variables. First it will request the new thread lengths to be calculated by the Control Software. Then, it will calculate the number of steps necessary to achieve the new thread length using the current thread length and the calculated one. After finding the number of steps for all of the 3 threads, it will clear the current location of each stepper (from the AccelStepper library) and based on the chosen percentage speed, it will calculate the necessary acceleration and max speed for each of the motors. It will then pass the target values to the AccelStepper library so it handles the movement. As stated in the previous section, the functions that calculate the new thread lengths from the Control Software have proven to be unreliable, and need to be developed again.

For this reason, we cannot show the block implemented in the hardware yet as it will not show reliable results.

Control Software

Function

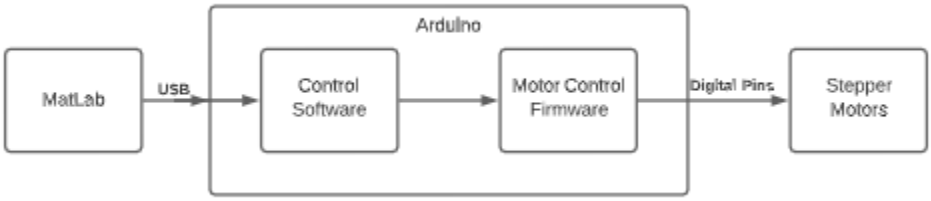

Figure 3: Control Software Relationship to rest of Spydercam

The Control Software's main purpose is to translate G-Code from MatLab into instructions to the Motor Control Firmware block so that it performs the desired operation successfully. The Control Software keeps track of many different variables and constants that are used to calculate and confirm actions based on multiple operation codes it supports.

The Control Software is responsible for calculating and relaying the necessary amount of steps that each of the three motor nodes needs to perform. It is also responsible for gathering operational data from the sensors and relaying that information back to MatLab for Analysis.

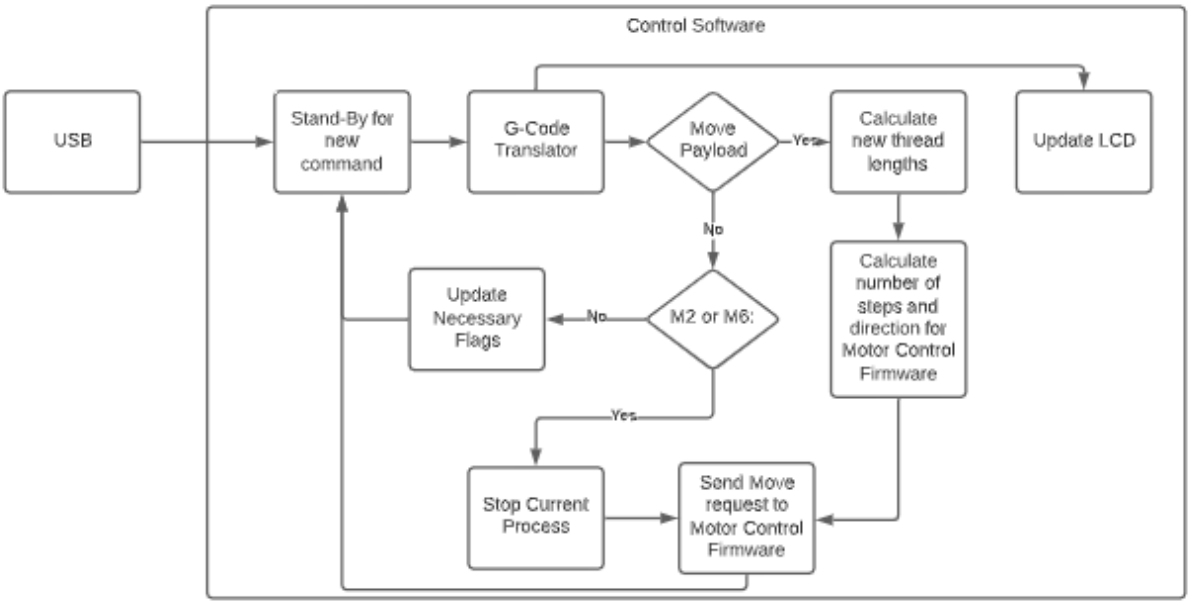

Figure 4: Control Software Flow Chart of Operation

Variables

The Control Software relies on accurate variables to operate correctly. These variables include but are not limited to:

- Current thread lengths.
- Current x/y location.
- Current payload height.
- Target x/y/height locations.
- Motor percent speed as per requirement G-code G1.

With the use of the variables above and constants of the hardware, it is possible to make the necessary calculations that take the payload from point A to point B, or even change the payload height instead and keeping x and y constant.

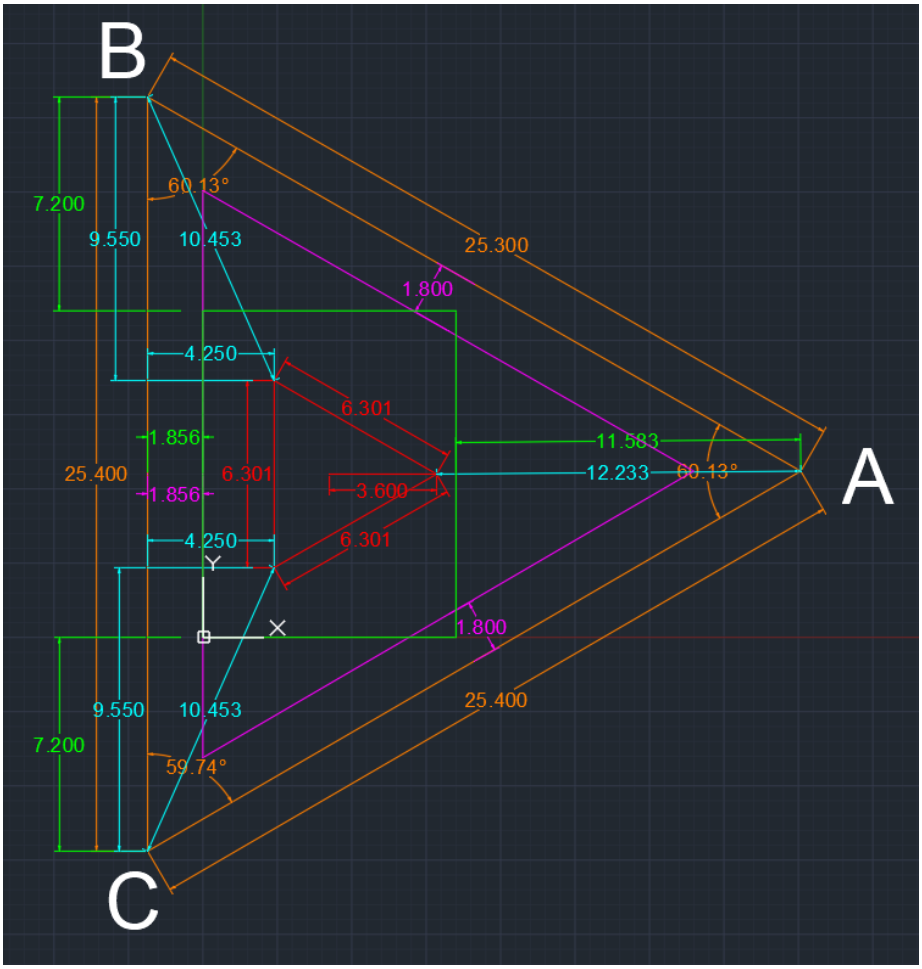

Figure 5: AutoCAD representation of the physical layout of the system.

By using the known values, it is possible to calculate the necessary thread lengths of all the threads at any given X, Y and Z coordinates. As can be seen in figure 5 and 6, the effective distance from one of the

pylons to the payload (at the same height) can be calculated using the Pythagorean theorem, this means that in order to find the actual length of the threads from the pylons to the payload, another triangle is required, where the hypotenuse of that triangle is the length of the thread itself, the opposite side is the distance between the height of the payload and the height of the pylon and the last adjacent side would be the previously calculated distance from the payload to the pylon.

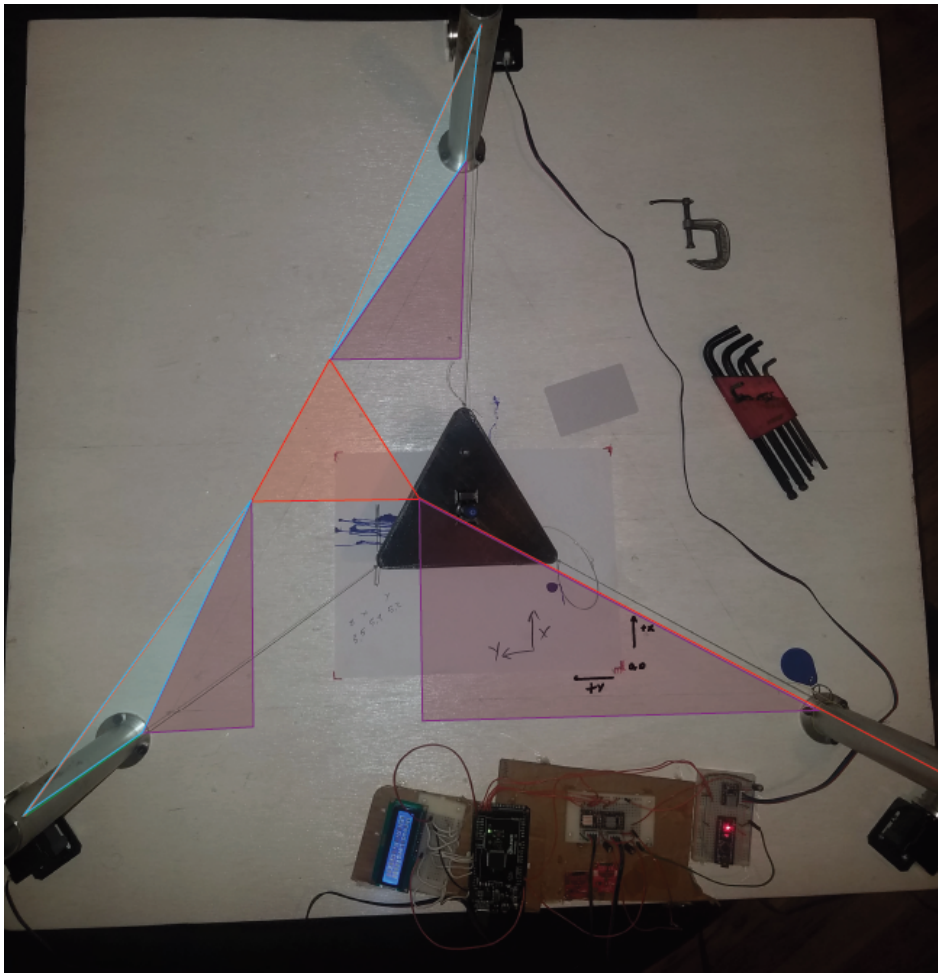

Figure 6: Triangles used for calculating the thread lengths.

Flags

The Control Software keeps track of 2 separate flags.
One of them is the UNIT_MODE that defines the mode of input the coordinates using G-Code. When this flag is TRUE, the Control Software will accept input coordinates in Inches, while if it is FALSE, it will accept inputs in Millimeters. The second flag the Control Software keeps track of is the ABSOLUTE_COORDINATES flag. As the name suggests, this flag keeps track of how the coordinates that are input in G0 or G1 are handled. If the flag value is TRUE, then the coordinates are absolute. The opposite happens when the flag is set to false by a G91 command.

Communication

In order to communicate with the Control Software, some standards must be met.

1. Only the G-Code presented in the table below is accepted..
2. All coordinates must be of type double or integer.
3. The effective speed defined in G1 must be a double from 0.1 to 4.5 inches per second.
4. An operation code must be followed by the proper arguments of that operation. The system will also accept partial arguments. If that is the case, the system will assume that the values of the arguments that were not input stay the same.
5. Operations that do not require payload displacement can be input one after the other.
6. Operations that require the payload to move will disregard any operations that are not M6 or M2 during operation.
7. It is possible to send multiple commands at the same time. However, if a command that requires payload movement is sent first, all commands that are not M2 or M6 will not be listened to by the Control Software.

The following list identifies the currently supported operation codes and their requirements:

G/M-CODE	VAR 1	VAR 2	Var 3	VAR 4	Definition
G0	X	Y	Z	-	Takes Payload to X, Y at max speed
G1	X	Y	Z	F	Takes Payload to X, Y at F speed
G20	-	-	-	-	Mode: Inputs in Inches
G21	-	-	-	-	Mode: Inputs in Millimeters
G90	-	-	-	-	Mode: Absolute Coordinates
G91	-	-	-	-	Mode: Incremental Coordinates
M2	-	-	-	-	End Program
M6	-	-	-	-	Tool Change
C1	-	-	-	-	Send Payload location to Serial
C2	-	-	-	-	Send thread lengths to Serial

The Control Software currently utilizes only 2 interface types: USB and digital Arduino I/O Ports 2, 3, 4, 5, 11, and 12 (for the LCD).

After each command is input, the Control software will relay the current location of the payload and sensor data to MatLab. That information is relayed in the following manner:

X<x-location>Y<y-location>Z<z-location>A<thread-len-a>B<thread-len-b>C<thread-len-c>L<light-level>R<rfid-data>G

The letter ‘G’ that is sent after the information indicates that the software is ready to receive another command.
When a C1 or C2 command is executed, the format is different in the way that the information is sent to MatLab. It will include a ‘#’ to indicate the end of the transmission cycle of the information requested.

Physical System Layout

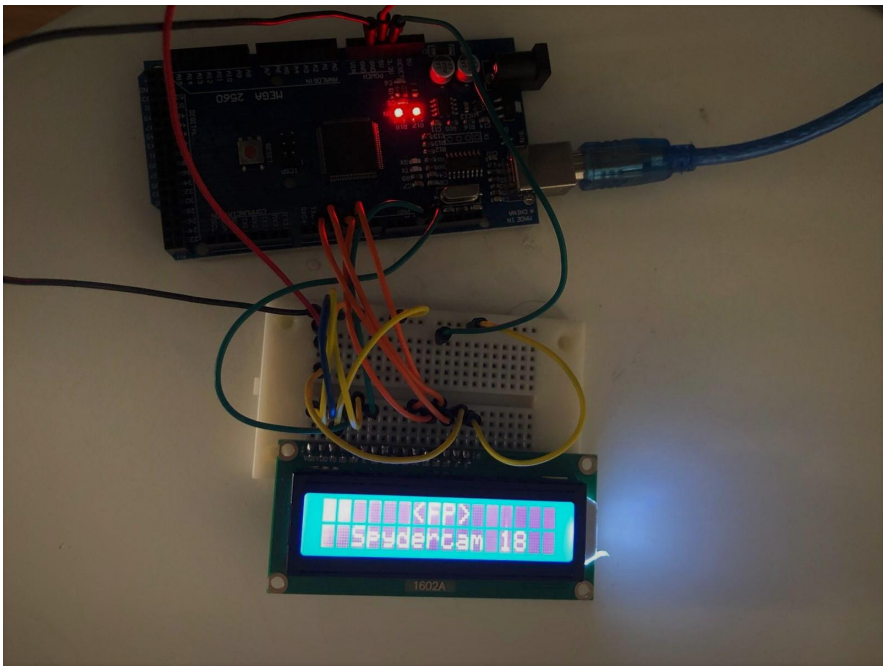

Figure 7: Current Physical Setup of LCD.

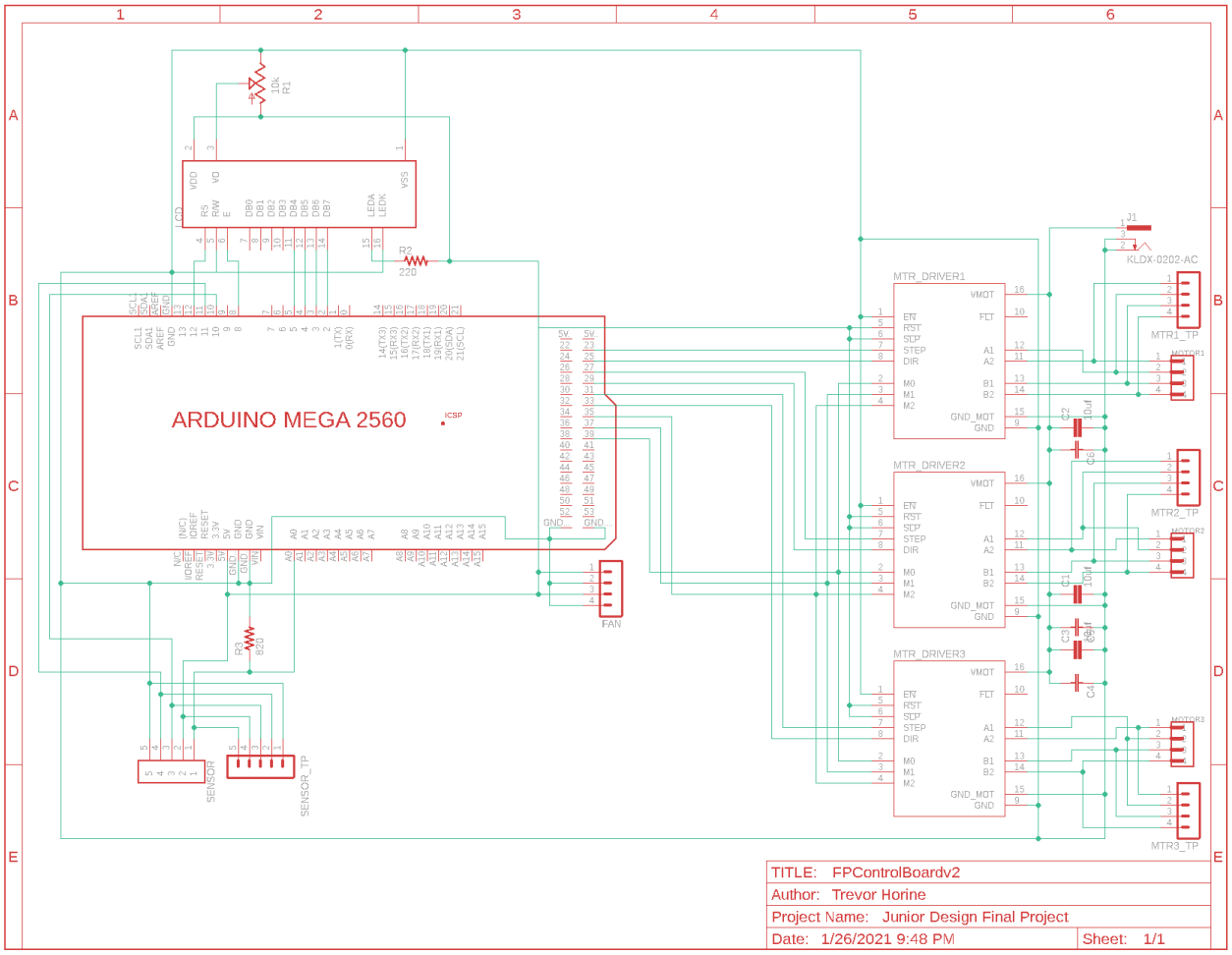

Figure 8: Current Arduino-System connection layout.

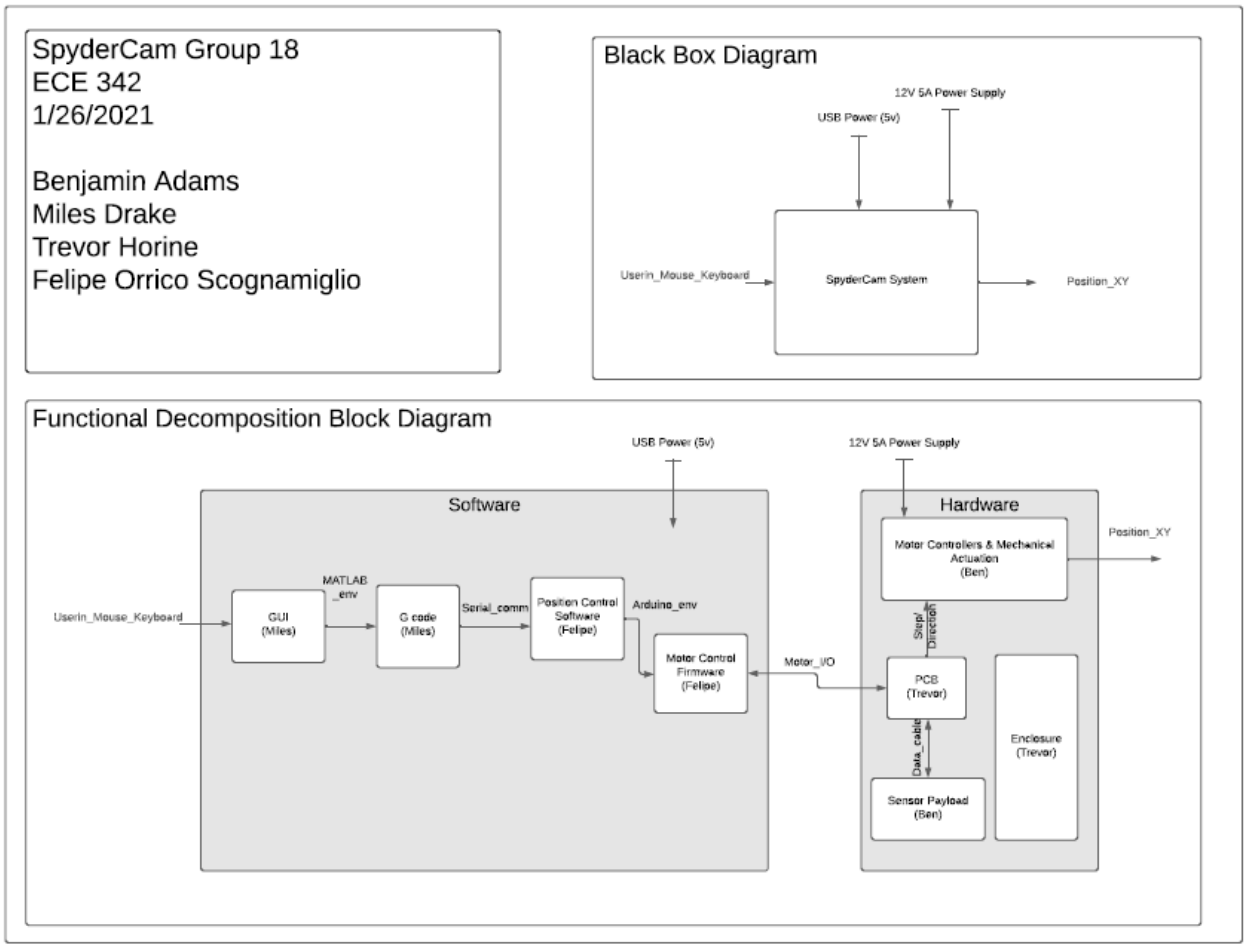

Figure 9: Current Full-System Block Diagram

Interface Definitions	
Userin_Mouse_Keyboard	PS/2 Keyboard Interface Vcc = 5.0V
MATLAB _env	Internally Defined MATLAB Variables Floating-point precision
Serial_comm	Protocol: UART 115200 baud G-Code Commands
Arduino_env	Internally Defined Arduino Variables See ATMEGA 2560 DS
Motor_I/O	10 Digital Pins Vmin = 0V Vmax = 5V
Step/Direction	Vmin = -0.5 V Vmax = 7.0 V
Data_cable	6 Pin High Speed UART (115200 Baud) Vmin = 0V Vmax = 5V
USB_Power	V = 5 V I = 0.5 A P = 2.5 W
Motor_Power	V = 12.0V I = 1.5 A (nominal) I <= 1.1 A (Actual)
Position_XY	x/y payload position spanning 23.00" equilateral triangle Lateral Deviation ≤0.25" per 10" travel

Figure 10: Interface Definitions for Figure 7

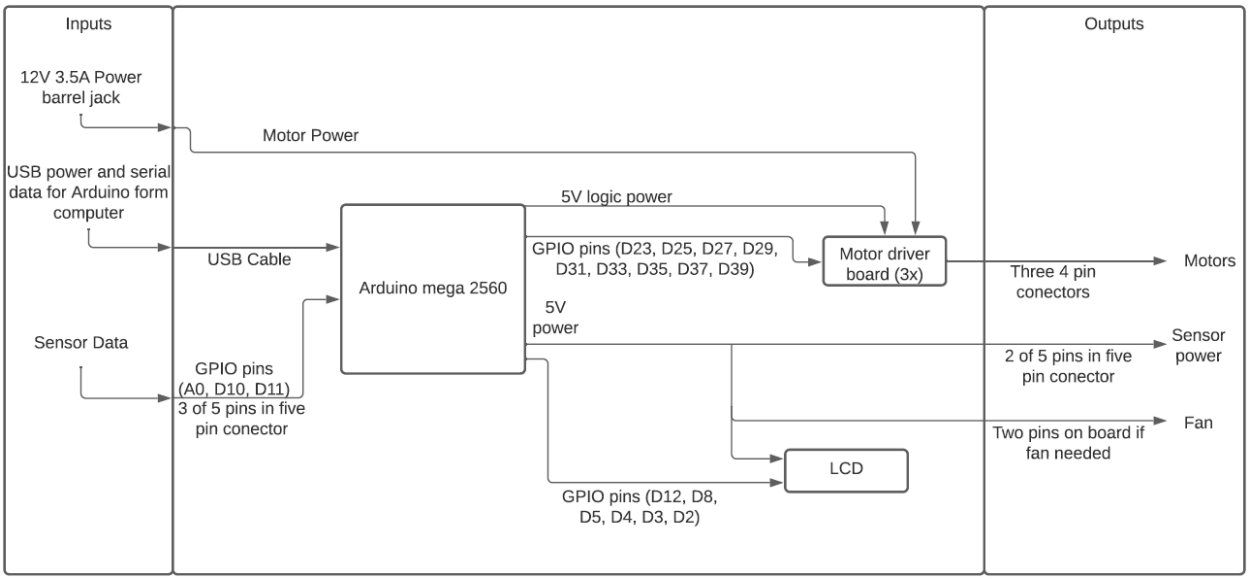

Figure 11: In-depth Full-System Block Diagram

Combined Code

```
/*
  Arduino Control Software Block & Motor Control Firmware
  Author: Felipe Orrico Scognamiglio
  For: Oregon State University - Junior Design - Final Project Spydercam18
  Date: 03/01/2021
  Version: BETA 1.25
*/

/*
  Known Problems:
    - After M2 command, Incremental Coordinate mode will not work properly
    and C1 will not be able to report correct coordinates
    - Inputting M2 and M6 right after the other has undefined behaviour
    - Current X, Y, Z positions are not available after receiving M2/M6, only
    after another G0 or G1 is executed. This happens because it is hard to extrapolate
    the current position only based on current thread lengths. Instead, the program will
    update after another G0 or G1 command.
*/

//////////////////////////LCD SETUP//////////////////////////
#include <LiquidCrystal.h>
const int rs = 12, en = 8, d4 = 5, d5 = 4, d6 = 3, d7 = 2;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);
```

```
//////////////////////LCD SETUP//////////////////////

//////////////////////Sensor SETUP//////////////////////
#include <SoftwareSerial.h>
#include "PN532_SWHSU.h"
#include "PN532.h"

SoftwareSerial SWSerial( 10, 11 ); // RX, TX
PN532_SWHSU pn532swhsu( SWSerial );
PN532 nfc( pn532swhsu );

const int sensor = A0;

float sensorVal = 0;
float voltage = 0;
float lightLevel = 0;

//////////////////////Sensor SETUP//////////////////////


//////////////////////CONSTANTS//////////////////////

//ALL PHYSICAL VALUES ARE REPORTED IN INCHES
const double A_PAPER_DISTANCE = 11.583;
const double B_PAPER_DISTANCE = 7.2;
const double C_PAPER_DISTANCE = 7.2;

const double HEIGHT_PYLON_A = 11.8;
const double HEIGHT_PYLON_B = 11.8;
const double HEIGHT_PYLON_C = 11.8;

const double PAYLOAD_CENTER_THREAD_DISTANCE_A_x = 3.6;
const double PAYLOAD_CENTER_THREAD_DISTANCE_C_x = 1.85;
const double PAYLOAD_CENTER_THREAD_DISTANCE_C_y = 3.15;
const double PAYLOAD_CENTER_THREAD_DISTANCE_B_x = 1.85;
const double PAYLOAD_CENTER_THREAD_DISTANCE_B_y = 3.15;

const double PAPER_HEIGHT = 11.00;
const double PAPER_WIDTH = 8.50;

const double PAPER_DISTANCE_LOW_EDGE = 1.856;

const double HOME_X = 0;
const double HOME_Y = 0;
const double HOME_Z = 6;
```

```
const double INCHES_PER_STEP = 0.024662;
const double MAX_STEPS_PER_SEC = 182;

//////////////////////////CONSTANTS//////////////////////////

//////////////////////////VARIABLES//////////////////////////

double CURRENT_PAYLOAD_HEIGHT = 0;
double CURRENT_THREAD_LEN_A = 0;
double CURRENT_THREAD_LEN_B = 0;
double CURRENT_THREAD_LEN_C = 0;
double CURRENT_X = 0;
double CURRENT_Y = 0;

//values that are updated when there is movement
int OPCODE = -1;
double X_ADDRESS_TARGET = -1;
double Y_ADDRESS_TARGET = -1;
double Z_ADDRESS_TARGET = -1;
int F_PERCENT_SPEED = 100;

///M6///
double M6_SAVE_LEN_A = 0;
double M6_SAVE_LEN_B = 0;
double M6_SAVE_LEN_C = 0;
double M6_SAVE_X = -1;
double M6_SAVE_Y = -1;
double M6_SAVE_Z = -1;
double M6_SAVE_Speed = 0;
bool M6_EN = false;

//////////////////////////VARIABLES//////////////////////////

////////////////////////////FLAGS////////////////////////////

bool UNIT_MODE = true; //true = inches, false = mm;
bool ABSOLUTE_COORDINATES = true; //true = absolute, false = incremental
bool ON_MOVE = false;
bool M2_EN = false;

////////////////////////////FLAGS////////////////////////////

//////////////////////////STEPPER//////////////////////////
```

```
#include "AccelStepper.h"
#include "MultiStepper.h"

//change pins here to reflect correct pins
#define dirPinA 25
#define stepPinA 23
#define dirPinB 29
#define stepPinB 27
#define dirPinC 33
#define stepPinC 31

#define motorInterfaceType 1

AccelStepper stepperA;// = AccelStepper(motorInterfaceType, stepPinA, dirPinA);
AccelStepper stepperB;// = AccelStepper(motorInterfaceType, stepPinB, dirPinB);
AccelStepper stepperC;// = AccelStepper(motorInterfaceType, stepPinC, dirPinC);

MultiStepper steppers = MultiStepper();

//////////////////////////STEPPER/////////////////////////

//////////////////////G-CODE LIB SETUP//////////////////////
#include "gcode.h"
#include "Arduino_Firmware_ALPHA.h"
#define NumberOfCommands 10
//you can add more more commands here if needed and create a function under G-Code to execute.
struct commandscallback commands[NumberOfCommands] =
{{"G0",G0_cmd},{"G1",G1_cmd},{"G20",G20_cmd},{"G21",G21_cmd},{"G90",G90_cmd},{"G91",
G91_cmd},{"M2",M2_cmd},{"M6",M6_cmd},{"C1",C1_cmd},{"C2",C2_cmd}};
gcode Commands(NumberOfCommands,commands);
//////////////////////G-CODE LIB SETUP//////////////////////

//////////////////////////G-Code//////////////////////////

void G0_cmd(){

    if (ON_MOVE || M6_EN) {
      //Serial.println("ON_MOVE");
      return;
    }


    double x_coord = CURRENT_X;
    double y_coord = CURRENT_Y;
    double z_coord = CURRENT_PAYLOAD_HEIGHT;
```

```
    if(Commands.availableValue('X')){
      x_coord = Commands.GetValue('X');
    }
    if(Commands.availableValue('Y')){
      y_coord = Commands.GetValue('Y');
    }
    if(Commands.availableValue('Z')){
      z_coord = Commands.GetValue('Z');
    }

    int f_speed = 100; //maximum speed

    if (!UNIT_MODE){ //translate mm to in
      x_coord = mm_to_in(x_coord);
      y_coord = mm_to_in(y_coord);
      z_coord = mm_to_in(z_coord);
    }

    if (!ABSOLUTE_COORDINATES){//incremental mode
      x_coord += CURRENT_X;
      y_coord += CURRENT_Y;
      z_coord += CURRENT_PAYLOAD_HEIGHT;
    }

    //checking for boundaries
    if (x_coord > 8.5)
      x_coord = 8.5;
    if (y_coord > 11)
      x_coord = 11;
    if (z_coord > 11)
      z_coord = 11;

    //update global values
    Y_ADDRESS_TARGET = y_coord;
    X_ADDRESS_TARGET = x_coord;
    Z_ADDRESS_TARGET = z_coord;
    F_PERCENT_SPEED = f_speed;

    update_lcd(0);

    go(x_coord,y_coord,z_coord, f_speed);
}

void G1_cmd(){
```

```
if (ON_MOVE || M6_EN) return;

double x_coord = CURRENT_X;
double y_coord = CURRENT_Y;
double z_coord = CURRENT_PAYLOAD_HEIGHT;
double f_speed_in = 4.5;
int f_speed = 100; //maximum speed

if(Commands.availableValue('X')){
  x_coord = Commands.GetValue('X');
}
if(Commands.availableValue('Y')){
  y_coord = Commands.GetValue('Y');
}
if(Commands.availableValue('Z')){
  z_coord = Commands.GetValue('Z');
}
if(Commands.availableValue('F')){
  f_speed_in = Commands.GetValue('F');
}

if (!UNIT_MODE){ //translate mm to in
  x_coord = mm_to_in(x_coord);
  y_coord = mm_to_in(y_coord);
  z_coord = mm_to_in(z_coord);
  f_speed_in = mm_to_in(f_speed_in);
}

if (!ABSOLUTE_COORDINATES){//incremental mode
  x_coord += CURRENT_X;
  y_coord += CURRENT_Y;
  z_coord += CURRENT_PAYLOAD_HEIGHT;
}

//checking for boundaries
if (x_coord > 8.5)
  x_coord = 8.5;
if (y_coord > 11)
  x_coord = 11;
if (z_coord > 11)
  z_coord = 11;

f_speed_in = (f_speed_in/4.5)*100;
```

```
    if (f_speed_in > 100)
      f_speed = 100;
    else if (f_speed_in <= 1)
      f_speed = 1;
    else
      f_speed = int(f_speed_in);

    //update global values
    Y_ADDRESS_TARGET = y_coord;
    X_ADDRESS_TARGET = x_coord;
    Z_ADDRESS_TARGET = z_coord;
    F_PERCENT_SPEED = f_speed;

    update_lcd(1);

    go(x_coord, y_coord, z_coord, f_speed);
}

void G20_cmd(){

  if (ON_MOVE || M6_EN) return;

  UNIT_MODE = true;
  update_lcd(20);
}

void G21_cmd(){

  if (ON_MOVE || M6_EN) return;

  UNIT_MODE = false;
  update_lcd(21);
}

void G90_cmd(){

  if (ON_MOVE || M6_EN) return;

  ABSOLUTE_COORDINATES = true;
  update_lcd(90);
}

void G91_cmd(){

  if (ON_MOVE || M6_EN) return;
```

```
  ABSOLUTE_COORDINATES = false;
  update_lcd(91);
}

void M2_cmd(){
  update_lcd(2);
  if (ON_MOVE){
     stepperA.stop();
     stepperB.stop();
     stepperC.stop();

     ON_MOVE = false;
     ABSOLUTE_COORDINATES = true;
     M2_EN = true;
     M6_EN = false;
  }
}

void M6_cmd(){
  update_lcd(6);
  if (!M6_EN){ //m6 first time
    //stop

    //if (ON_MOVE){
    stepperA.stop();
    stepperB.stop();
    stepperC.stop();

    ON_MOVE = false;
    ABSOLUTE_COORDINATES = true;
    M6_EN = true;

    double actual_a = stepperA.currentPosition() * INCHES_PER_STEP;
    double actual_b = -(stepperB.currentPosition() * INCHES_PER_STEP);
    double actual_c = stepperC.currentPosition() * INCHES_PER_STEP;

    CURRENT_THREAD_LEN_A += actual_a;
    CURRENT_THREAD_LEN_B += actual_b;
    CURRENT_THREAD_LEN_C += actual_c;

    M6_SAVE_LEN_A = CURRENT_THREAD_LEN_A;
    M6_SAVE_LEN_B = CURRENT_THREAD_LEN_B;
    M6_SAVE_LEN_C = CURRENT_THREAD_LEN_C;
    M6_SAVE_X = X_ADDRESS_TARGET;
```

```
    M6_SAVE_Y = Y_ADDRESS_TARGET;
    M6_SAVE_Z = Z_ADDRESS_TARGET;

    if (M6_SAVE_X == -1 || M6_SAVE_Y == -1 || M6_SAVE_Z == -1){
      M6_SAVE_X = CURRENT_X;
      M6_SAVE_Y = CURRENT_Y;
      M6_SAVE_Z = CURRENT_PAYLOAD_HEIGHT;
    }
    if (M6_SAVE_X == 0 && M6_SAVE_Y == 0 && M6_SAVE_Z == 0){
      M6_SAVE_X = CURRENT_X;
      M6_SAVE_Y = CURRENT_Y;
      M6_SAVE_Z = CURRENT_PAYLOAD_HEIGHT;
    }
    M6_SAVE_Speed = F_PERCENT_SPEED;

    go_block(HOME_X,HOME_Y,HOME_Z, 100);
  }
  else {

    go_len_block(M6_SAVE_LEN_A,M6_SAVE_LEN_B,M6_SAVE_LEN_C, 100);

    M6_EN = false;

    //go(M6_SAVE_X,M6_SAVE_Y,M6_SAVE_Z,M6_SAVE_Speed);

    /*M6_SAVE_LEN_A = 0;
    M6_SAVE_LEN_B = 0;
    M6_SAVE_LEN_C = 0;
    M6_SAVE_X = -1;
    M6_SAVE_Y = -1;
    M6_SAVE_Z = -1;
    M6_SAVE_Speed = 0;*/
  }
}

void C1_cmd(){ //Write value of X,Y,Z to the Serial Connection

  //if (ON_MOVE || M6_EN) return;

  Serial.write("X");
  Serial.print(CURRENT_X);
  Serial.write("Y");
  Serial.print(CURRENT_Y);
  Serial.write("Z");
  Serial.print(CURRENT_PAYLOAD_HEIGHT);
```

```
  Serial.write("#");
  update_lcd(57);
  delay(500);
}

void C2_cmd() { //Write thread lengths to Serial
  //if (ON_MOVE || M6_EN) return;

  Serial.write("A");
  Serial.print(CURRENT_THREAD_LEN_A);
  Serial.write("B");
  Serial.print(CURRENT_THREAD_LEN_B);
  Serial.write("C");
  Serial.print(CURRENT_THREAD_LEN_C);
  Serial.write("#");
  update_lcd(58);
  delay(500);
}

//////////////////////////G-Code//////////////////////////


//////////////////////THREAD CALCULATIONS//////////////////////
double calculate_thread_height_var(double z){
    return HEIGHT_PYLON_A - z;
}

double calculate_thread_length_A_h(double x, double y, double z) {
  double height_created_triangle =  A_PAPER_DISTANCE + 8.5 - x -
PAYLOAD_CENTER_THREAD_DISTANCE_A_x;
  double side_created_triangle;
  if ((y - 5.5) < 0)
    side_created_triangle = 5.5 - y;
  else
    side_created_triangle = y - 5.5;

  double trace_length = sqrt(pow(height_created_triangle, 2) + pow(side_created_triangle, 2));

  double thread_length = sqrt(pow(trace_length,2) + pow(calculate_thread_height_var(z),2));

  return thread_length;
}

double calculate_thread_length_B_h(double x, double y, double z) {
  double height_created_triangle =  PAPER_HEIGHT + B_PAPER_DISTANCE - y -
```

```
PAYLOAD_CENTER_THREAD_DISTANCE_B_y;
  double side_created_triangle =  PAPER_DISTANCE_LOW_EDGE + x -
PAYLOAD_CENTER_THREAD_DISTANCE_B_x;

  double trace_length = sqrt(pow(height_created_triangle,2) + pow(side_created_triangle,2));

  //double trace_length = PAPER_DISTANCE_1 - y -
PAYLOAD_CENTER_THREAD_DISTANCE_B_y;

  double thread_length = sqrt(pow(trace_length,2) + pow(calculate_thread_height_var(z),2));

  return thread_length;
}

double calculate_thread_length_C_h(double x, double y, double z) {
  double height_created_triangle = C_PAPER_DISTANCE + y -
PAYLOAD_CENTER_THREAD_DISTANCE_B_y;
  double side_created_triangle = PAPER_DISTANCE_LOW_EDGE + x -
PAYLOAD_CENTER_THREAD_DISTANCE_C_x;

  double trace_length = sqrt(pow(height_created_triangle,2) + pow(side_created_triangle,2));

  double thread_length = sqrt(pow(trace_length,2) + pow(calculate_thread_height_var(z),2));

  return thread_length;
}

//////////////////////THREAD CALCULATIONS//////////////////////

//////////////////////UNIT CONVERSION//////////////////////
double mm_to_in(double mm){
  return (mm/(25.4));
}

double in_to_mm(double in){
  return (in *(25.4));
}
//////////////////////UNIT CONVERSION//////////////////////

void update_lcd(int opcode){
  lcd.clear();
  if (opcode == 0 || opcode == 1){ //G0 or G1 (displays as integer for size)
    lcd.setCursor(0, 0); //1st line 1st char
    lcd.print("OPCODE: G");
    lcd.print(opcode);
```

```
  lcd.print(" F:");
  lcd.print(F_PERCENT_SPEED);
  lcd.setCursor(0, 1); //2nd Line 1st char
  double x = X_ADDRESS_TARGET;
  double y = Y_ADDRESS_TARGET;
  if (!UNIT_MODE){ //translate mm to in
     x = in_to_mm(X_ADDRESS_TARGET);
     y = in_to_mm(Y_ADDRESS_TARGET);
  }
  lcd.print("X: ");
  lcd.print(x);
  lcd.setCursor(8, 1); //2nd Line 9th char
  lcd.print("Y: ");
  lcd.print(y);
 }
 else if (opcode == 20){
  lcd.setCursor(0, 0); //1st line 1st char
  lcd.print("OPCODE: G");
  lcd.print(opcode);
  lcd.setCursor(0, 1); //2nd Line 1st char
  lcd.print("Input Mode > IN");
 }
 else if (opcode == 21){
  lcd.setCursor(0, 0); //1st line 1st char
  lcd.print("OPCODE: G");
  lcd.print(opcode);
  lcd.setCursor(0, 1); //2nd Line 1st char
  lcd.print("Input Mode > MM");
 }
 else if (opcode == 90){
  lcd.setCursor(0, 0); //1st line 1st char
  lcd.print("OPCODE: G");
  lcd.print(opcode);
  lcd.setCursor(0, 1); //2nd Line 1st char
  lcd.print("Coord Mode > ABS");
 }
 else if (opcode == 91){
  lcd.setCursor(0, 0); //1st line 1st char
  lcd.print("OPCODE: G");
  lcd.print(opcode);
  lcd.setCursor(0, 1); //2nd Line 1st char
  lcd.print("Coord Mode > INC");
 }
 else if (opcode == 2){
  lcd.setCursor(0, 0); //1st line 1st char
```

```
    lcd.print("OPCODE: M");
    lcd.print(opcode);
    lcd.setCursor(0, 1); //2nd Line 1st char
    lcd.print("END PROGRAM");
  }
  else if (opcode == 6){
    lcd.setCursor(0, 0); //1st line 1st char
    lcd.print("OPCODE: G");
    lcd.print(opcode);
    lcd.setCursor(0, 1); //2nd Line 1st char
    lcd.print("TOOL CHANGE");
  }
  else if (opcode == 57){
    lcd.setCursor(0, 0); //1st line 1st char
    lcd.print("Sending Location");
    lcd.setCursor(0, 1); //2nd Line 1st char
    lcd.print("<<<<<<<<>>>>>>>>");
  }
  else if (opcode == 58){
    lcd.setCursor(0, 0); //1st line 1st char
    lcd.print("Sending Threads");
    lcd.setCursor(0, 1); //2nd Line 1st char
    lcd.print("<<<<<<<<>>>>>>>>");
  }
  else {
    lcd.setCursor(0, 0); //1st line 1st char
    lcd.print("ERROR: ");
    lcd.print(opcode);
    lcd.setCursor(0, 1); //2nd Line 1st char
    lcd.print("OPCODE NOT FOUND");
  }
}

//Calculates the number of steps to move from current_ to new_ length of thread.
//If the output is negative, then the motor must reduce the size of the thread instead of increasing it

////////////////////////CALCULATIONS FOR MOVEMENT////////////////////////
int calculate_number_steps(double current_, double new_){
  int num_steps = int((new_ - current_)/INCHES_PER_STEP);
  return num_steps;
}

int go_block(double x, double y, double z, int f_speed) {
  X_ADDRESS_TARGET = x;
  Y_ADDRESS_TARGET = y;
```

```
Z_ADDRESS_TARGET = z;
F_PERCENT_SPEED = f_speed;

//calculate new necessary thread lengths
double new_len_a = calculate_thread_length_A_h(x,y,z);
double new_len_b = calculate_thread_length_B_h(x,y,z);
double new_len_c = calculate_thread_length_C_h(x,y,z);

//calculate number of steps and direction of movement to move from current pos to new pos.
int steps_A = calculate_number_steps(CURRENT_THREAD_LEN_A, new_len_a);
int steps_B = calculate_number_steps(CURRENT_THREAD_LEN_B, new_len_b);
int steps_C = calculate_number_steps(CURRENT_THREAD_LEN_C, new_len_c);


//Setting important info in stepper classes
stepperA.setCurrentPosition(0);
stepperB.setCurrentPosition(0);
stepperC.setCurrentPosition(0);
int max_speed = int((MAX_STEPS_PER_SEC/100)*f_speed);
stepperA.setMaxSpeed(max_speed);
stepperB.setMaxSpeed(max_speed);
stepperC.setMaxSpeed(max_speed);
//move the payload to location
long steps[] = {steps_A, -steps_B, steps_C};

lcd.clear();
lcd.print("Moving to >>>>");
lcd.setCursor(0, 1);
lcd.print(x);
lcd.print(" ");
lcd.print(y);
lcd.print(" ");
lcd.print(z);

steppers.moveTo(steps);

//move_to(steps_A, steps_B, steps_C);

//blocks until steppers finish moving
while(steppers.run()){
}

//find actual displacement
double actual_a = stepperA.currentPosition() * INCHES_PER_STEP;
double actual_b = -(stepperB.currentPosition() * INCHES_PER_STEP);
```

```
  double actual_c = stepperC.currentPosition() * INCHES_PER_STEP;

  //after done, update current values
  CURRENT_X = x;
  CURRENT_Y = y;
  CURRENT_PAYLOAD_HEIGHT = z;
  CURRENT_THREAD_LEN_A += actual_a;
  CURRENT_THREAD_LEN_B += actual_b;
  CURRENT_THREAD_LEN_C += actual_c;

  return 0;
}

int go_len_block(double a, double b, double c, int f_speed) {

  //calculate number of steps and direction of movement to move from current pos to new pos.
  int steps_A = calculate_number_steps(CURRENT_THREAD_LEN_A, a);
  int steps_B = calculate_number_steps(CURRENT_THREAD_LEN_B, b);
  int steps_C = calculate_number_steps(CURRENT_THREAD_LEN_C, c);


  //Setting important info in stepper classes
  stepperA.setCurrentPosition(0);
  stepperB.setCurrentPosition(0);
  stepperC.setCurrentPosition(0);
  int max_speed = int((MAX_STEPS_PER_SEC/100)*f_speed);
  stepperA.setMaxSpeed(max_speed);
  stepperB.setMaxSpeed(max_speed);
  stepperC.setMaxSpeed(max_speed);
  //move the payload to location
  long steps[] = {steps_A, -steps_B, steps_C};

  steppers.moveTo(steps);

  //blocks until steppers finish moving
  while(steppers.run()){
  }

  //find actual displacement
  double actual_a = stepperA.currentPosition() * INCHES_PER_STEP;
  double actual_b = -(stepperB.currentPosition() * INCHES_PER_STEP);
  double actual_c = stepperC.currentPosition() * INCHES_PER_STEP;

  //after done, update current values
  CURRENT_THREAD_LEN_A += actual_a;
```

```
  CURRENT_THREAD_LEN_B += actual_b;
  CURRENT_THREAD_LEN_C += actual_c;

  X_ADDRESS_TARGET = 0;
  Y_ADDRESS_TARGET = 0;
  Z_ADDRESS_TARGET = 0;

  return 0;
}


int go(double x, double y, double z, int f_speed) {
  X_ADDRESS_TARGET = x;
  Y_ADDRESS_TARGET = y;
  Z_ADDRESS_TARGET = z;
  F_PERCENT_SPEED = f_speed;

  //calculate new necessary thread lengths
  double new_len_a = calculate_thread_length_A_h(x,y,z);
  double new_len_b = calculate_thread_length_B_h(x,y,z);
  double new_len_c = calculate_thread_length_C_h(x,y,z);

  /*Serial.println("");
  Serial.print("New Length A: ");
  Serial.println(new_len_a);
  Serial.print("New Length B: ");
  Serial.println(new_len_b);
  Serial.print("New Length C: ");
  Serial.println(new_len_c);

  Serial.println("");
  Serial.print("CurrentLength A: ");
  Serial.println(CURRENT_THREAD_LEN_A);
  Serial.print("CurrentLength B: ");
  Serial.println(CURRENT_THREAD_LEN_B);
  Serial.print("CurrentLength C: ");
  Serial.println(CURRENT_THREAD_LEN_C);*/

  //calculate number of steps and direction of movement to move from current pos to new pos.
  int steps_A = calculate_number_steps(CURRENT_THREAD_LEN_A, new_len_a);
  int steps_B = calculate_number_steps(CURRENT_THREAD_LEN_B, new_len_b);
  int steps_C = calculate_number_steps(CURRENT_THREAD_LEN_C, new_len_c);


  //Setting important info in stepper classes
```

```
stepperA.setCurrentPosition(0);
stepperB.setCurrentPosition(0);
stepperC.setCurrentPosition(0);
int max_speed = int((MAX_STEPS_PER_SEC/100)*f_speed);
stepperA.setMaxSpeed(max_speed);
stepperB.setMaxSpeed(max_speed);
stepperC.setMaxSpeed(max_speed);
//stepperA.setAcceleration(max_speed/5);
//stepperB.setAcceleration(max_speed/5);
//stepperC.setAcceleration(max_speed/5);
//move the payload to location
long steps[] = {steps_A, -steps_B, steps_C};

lcd.clear();
lcd.print("Moving to: F");
lcd.setCursor(0, 1);
lcd.print(x);
lcd.print(" ");
lcd.print(y);
lcd.print(" ");
lcd.print(z);

steppers.moveTo(steps);

//move_to(steps_A, steps_B, steps_C);

ON_MOVE = true;

//blocks until steppers finish moving
while(steppers.run() && ON_MOVE){
  Commands.available();
}

//find actual displacement
double actual_a = stepperA.currentPosition() * INCHES_PER_STEP;
double actual_b = -(stepperB.currentPosition() * INCHES_PER_STEP);
double actual_c = stepperC.currentPosition() * INCHES_PER_STEP;

/*Serial.println("");
Serial.print("DELTA A:");
Serial.println(actual_a);
Serial.print("DELTA B:");
Serial.println(actual_b);
Serial.print("DELTA C:");
Serial.println(actual_c);
```

```
  Serial.println("\nFinal Stepper Pos:");
  Serial.print("Stepper A: ");
  Serial.println(stepperA.currentPosition());
  Serial.print("Stepper B: ");
  Serial.println(-stepperB.currentPosition());
  Serial.print("Stepper C: ");
  Serial.println(stepperC.currentPosition());*/

  ON_MOVE = false;

  //after done, update current values
  if (!M2_EN && !M6_EN){
    CURRENT_X = x;
    CURRENT_Y = y;
    CURRENT_PAYLOAD_HEIGHT = z;
  }
  if (!M6_EN){
    CURRENT_THREAD_LEN_A += actual_a;
    CURRENT_THREAD_LEN_B += actual_b;
    CURRENT_THREAD_LEN_C += actual_c;
  }
  M2_EN = false;

  X_ADDRESS_TARGET = 0;
  Y_ADDRESS_TARGET = 0;
  Z_ADDRESS_TARGET = 0;

  /*Serial.println("");
  Serial.print("Final Length A: ");
  Serial.println(CURRENT_THREAD_LEN_A);
  Serial.print("Final Length B: ");
  Serial.println(CURRENT_THREAD_LEN_B);
  Serial.print("Final Length C: ");
  Serial.println(CURRENT_THREAD_LEN_C);*/

  return 0;
}

//////////////////////////CALCULATIONS FOR MOVEMENT//////////////////////////

void init_values(){ //expects in not mm
  lcd.clear();
  lcd.print("Location:");
  lcd.setCursor(0, 1);
```

```
  lcd.print("Z, X, Y");
  while(Serial.available() <= 0){
  }
  double h = Serial.parseFloat();
  double x = Serial.parseFloat();
  double y = Serial.parseFloat();

  CURRENT_PAYLOAD_HEIGHT = h;

  double La = calculate_thread_length_A_h(x,y,h);
  double Lb = calculate_thread_length_B_h(x,y,h);
  double Lc = calculate_thread_length_C_h(x,y,h);

  CURRENT_THREAD_LEN_A = La;
  CURRENT_THREAD_LEN_B = Lb;
  CURRENT_THREAD_LEN_C = Lc;
  CURRENT_X = x;
  CURRENT_Y = y;

  lcd.clear();
  lcd.print("Z: ");
  lcd.print(CURRENT_PAYLOAD_HEIGHT);
  lcd.setCursor(0, 1); //2nd Line 1st char
  lcd.print("X: ");
  lcd.print(CURRENT_X);
  lcd.setCursor(8, 1); //2nd Line 9th char
  lcd.print("Y: ");
  lcd.print(CURRENT_Y);

}

void ready_command_lcd(){
  lcd.clear();
  lcd.setCursor(0, 0); //2nd Line 1st char
  lcd.print("    <READY>    ");
  lcd.setCursor(0, 1); //2nd Line 1st char
  lcd.print("   Listening   ");
}

void send_sensor(){


  sensorVal = analogRead(sensor);
  lightLevel = (sensorVal/1023)*10;
  Serial.write("L");
```

```
  Serial.print(lightLevel);

  boolean success = false;
  uint8_t uid[] = { 0, 0, 0, 0, 0, 0, 0 };  // Buffer to store the returned UID
  uint8_t uidLength;                        // Length of the UID (4 or 7 bytes depending on ISO14443A card
type)
  success = nfc.readPassiveTargetID(PN532_MIFARE_ISO14443A, &uid[0], &uidLength);
  long test = 0;
  if (success){
    for (uint8_t i=0; i < uidLength; i++)
    {
      //Serial.print(" 0x");Serial.print(uid[i], HEX);
      test = test + uid[i];
      if(i<3){
      test = test * 256;
      }
    }
    Serial.write("R");
    Serial.write(test);
  } else {
    Serial.write("R0");
  }
}

void task_done(){
  X_ADDRESS_TARGET = 0;
  Y_ADDRESS_TARGET = 0;
  Z_ADDRESS_TARGET = 0;
  F_PERCENT_SPEED = 100;
  OPCODE = 1000; //change OPCODE to 1000 when task is done
  //delay(2000);
  ready_command_lcd();

  Serial.write("X");
  Serial.print(CURRENT_X);
  Serial.write("Y");
  Serial.print(CURRENT_Y);
  Serial.write("Z");
  Serial.print(CURRENT_PAYLOAD_HEIGHT);
  Serial.write("A");
  Serial.print(CURRENT_THREAD_LEN_A);
  Serial.write("B");
  Serial.print(CURRENT_THREAD_LEN_B);
  Serial.write("C");
  Serial.print(CURRENT_THREAD_LEN_C);
```

```
  send_sensor();

  delay(1000);

  Serial.write("G"); //let matlab know when it is good to send another command.

}

void set_pins(){
  //LCD
  pinMode(2, OUTPUT);
  pinMode(3, OUTPUT);
  pinMode(4, OUTPUT);
  pinMode(5, OUTPUT);
  pinMode(8, OUTPUT);

  //STEPPERS
  pinMode(25, OUTPUT);
  pinMode(23, OUTPUT);
  pinMode(29, OUTPUT);
  pinMode(27, OUTPUT);
  pinMode(33, OUTPUT);
  pinMode(31, OUTPUT);

  pinMode(35, OUTPUT);
  pinMode(37, OUTPUT);
  pinMode(39, OUTPUT);

}

void setup() {
  Commands.begin();
  lcd.begin(16, 2);
  lcd.print("     <FP>     ");
  lcd.setCursor(2, 1);
  lcd.print("Spydercam 18");
  delay(2000);
  lcd.clear();
  lcd.print("     BETA     ");
  lcd.setCursor(0, 1);
  lcd.print("Arduino Firmware");
  delay(2000);
  init_values();
  delay(2000);
```

```
  set_pins();
  stepperA = AccelStepper(motorInterfaceType, stepPinA, dirPinA);
  stepperB = AccelStepper(motorInterfaceType, stepPinB, dirPinB);
  stepperC = AccelStepper(motorInterfaceType, stepPinC, dirPinC);

  digitalWrite(35, LOW);
  digitalWrite(37, LOW);
  digitalWrite(39, LOW);

  steppers.addStepper(stepperA);
  steppers.addStepper(stepperB);
  steppers.addStepper(stepperC);


  //sensor setup
  nfc.begin();
  lcd.clear();
  lcd.print("  Enabling NFC  ");
  lcd.setCursor(0, 1);
  lcd.print(" In Progress... ");
  uint32_t versiondata = nfc.getFirmwareVersion();
  if (! versiondata) {
    lcd.clear();
    lcd.print("     ERROR      ");
    lcd.setCursor(0, 1);
    lcd.print("   NFC Failed   ");
    while (1); // Halt
  }

  delay(1000);
  nfc.SAMConfig();
  ready_command_lcd();

  //Serial.write("G");
}

void loop() {
  if (Commands.available()) {
    task_done();
  }
}
```