Armanipulator-Firmware

Alex Westerman
Version 1.0~alpha1
Wed May 15 2019

Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

File List

Here is a list of all documented files with brief descriptions:

Armanipulator_Firmware.ino (Base file for the arduino sketch)

Controller.cpp (Source File for the Controller)

Controller.h (Contains methods for controlling the motors and parsing input)

cpu_map.h (Pin mapping and common configuration)

Class Documentation

Controller::arm_command Struct Reference

The structure used to define the overall instruction parsed from serial input.

`#include <Controller.h>`

Public Attributes

- Controller::Arm_Operation op
- double value

Detailed Description

The structure used to define the overall instruction parsed from serial input.

Definition at line 58 of file Controller.h.

Member Data Documentation

Controller::Arm_Operation Controller::arm_command::op

The operation used in the command Structure

Definition at line 59 of file Controller.h.

double Controller::arm_command::value

The value associated with the operation. If the command in op is ERROR, the value is 1 by default

Definition at line 60 of file Controller.h.

The documentation for this struct was generated from the following file:

- Controller.h

Controller Class Reference

The class-object used to interface between the serial interface and the stepper motor drivers.
`#include <Controller.h>`

Classes

- struct arm_command
  The structure used to define the overall instruction parsed from serial input.

Public Types

- enum Arm_Operation { ROTATE, GRAB, EXTEND, MICROSTEPS, ERROR }
  Used to define common commands parsed by the arm.
- enum microsteps { HALF, QUARTER, EIGTH, SIXTEENTH, THIRTY_SECOND }
- typedef enum Controller::microsteps Microsteps
  Used to placehold common Microstep configs for the motor drivers.

- typedef struct Controller::arm_command Arm_Command
  Type for the Arm_Command Struct to allow easy definition.

Public Member Functions

- Controller ()
  Constructor for the controller object In this constructor, the motors are created on the heap, and then the motor pointer vars now point to the motors on the heap. currentCmd is not initialized, because it does not need to be pointed to at the minute.

- virtual ~Controller ()
  Destructor for the controller object Has no purpose, therefore does nothing.

- void parseSerial (String rawinput)
  Translates Serial or Other Input into an Arm_Command struct object This function will take the string input, and then parse based on the first character of the string. Parsing goes as follows: -Any commands related to rotation will start with an 'r' or 'R', then followed by a double indicating revolutions -Any commands related to gripping will start with a 'g' or 'G', then followed by a 0 or 1. Other values will trigger an exception -Any commands related to extending the arm will start with an 'e' or 'E', then followed by a double indicating motor rotation. We will assert that the value is between -1 and 1 for now -Any unrecognized letter will return the ERROR enum and 1 as the value. Serial should display an error message, or trigger an exception Once the command is selected, the remaining value is parsed as a double, and then pointed to in the Arm_Command Pointer.

- void printExec ()
  Debug Function to test the parser Prints output in the serial connection based on the values in currentCmd.

- void executeCmd ()
  Runs Commands based on data pointed to by currentCmd Uses the parsing rules defined by parseSerial and then do the respective action.

- Arm_Command * getCommand ()

Getter method for the currentCmd pointer.

- void setCommand (Arm_Command *cmd)
  Setter method for the currentCmd pointer.

Detailed Description

The class-object used to interface between the serial interface and the stepper motor drivers.
Definition at line 20 of file Controller.h.

Member Typedef Documentation

Controller::Arm_Command

Type for the Arm_Command Struct to allow easy definition.

See also:
Controller::arm_command

Member Enumeration Documentation

enum Controller::Arm_Operation

Used to define common commands parsed by the arm.

Enumerator:

ROTATE	Run the Wrist Rotation Motor
GRAB	Run the Grab motor
EXTEND	Run the Arm Extender Motor
MICROSTEPS	Change the Microsteps for the driver
ERROR	Default Error Placeholder

Definition at line 28 of file Controller.h.

```
28                  {
29          ROTATE,
30          GRAB,
31          EXTEND,
32          MICROSTEPS,
33          ERROR
34      } Arm_Operation;
```

enum Controller::microsteps

Enumerator:

HALF	HALF. 1/2 Microstep Ratio
QUARTER	QUARTER. 1/4 Microstep Ratio
EIGTH	EIGTH. 1/8 Microstep Ratio
SIXTEENTH	SIXTEENTH. 1/16 Microstep Ratio
THIRTY_SECOND	THIRTY_SECOND. 1/32 Microstep Ratio

Definition at line 42 of file Controller.h.

```
42                              {
43          HALF,
44          QUARTER,
45          EIGTH,
46          SIXTEENTH,
47          THIRTY_SECOND
48      } Microsteps;
```

Member Function Documentation

Controller::executeCmd ()

Runs Commands based on data pointed to by currentCmd Uses the parsing rules defined by parseSerial and then do the respective action.

See also:

Controller::parseSerial

Definition at line 105 of file Controller.cpp.

```
105                             {
106     Controller::Arm_Operation cmdop = this->getCommand()->op;
//Extract the Arm_Operation enum stored in this controller object
107     //TODO: Check efficiency of using a switch statement instead of an if/else
chain
108     //See Jump Tables and low-level intricacies produced by AVR
109     switch (cmdop) {
//What is the command
110     case Controller::Arm_Operation::ROTATE:
//Rotate the wrist
111         if (this->getCommand()->value > 1 || this->getCommand()->value < -1) {
//Check the bounds of the value
112             Serial.println("Error: Value for Wrist Rotation commands should be
between -1 and 1");  //Print an error message and give explanation
113             return;
//Exits to the main loop if bounds are not met
114         }
```

```
  115         rotateDriver->move(currentCmd->value * 200);
//Otherwise, perform the movement
  116         break;
  117     case Controller::Arm_Operation::GRAB:
//Do the grabby hand
  118         if (this->getCommand()->value != 1 || this->getCommand()->value != -1)
{                   //Bounds checking, but stricter (Decimal values indicate a
partial oepning of the hand, which us not the best practice
  119             Serial.println("Error: Value for Grab commands should be either -1
or 1");             //Print an error message and give explanation
  120             return;
//Exits to the main loop if bounds are not met
  121         }
  122         grabDriver->move(currentCmd->value * 60);
//Do the grab (set value, motion depends on sign))
  123         break;
  124     case Controller::Arm_Operation::EXTEND:
//Extend the arm!
  125         if (this->getCommand()->value > 1 || this->getCommand()->value < -1) {
//Bounds checking, but not strict (arm should extend to variable lengths
  126             Serial.println("Error: Value for Arm Extension commands should be
between -1 and 1");   //Print an error message and give explanation
  127             return;
//Exits to the main loop if bounds are not met
  128         }
  129         extendDriver->move(currentCmd->value);
//Extend the arm
  130         break;
  131     case Controller::Arm_Operation::MICROSTEPS:
  132         //Empty for now
  133         break;
  134     case Controller::Arm_Operation::ERROR:
//He's Dead, Jim!
  135         Serial.println("Error, Unrecognized Command");
//Print out an error message
  136         break;
  137
  138     }
  139 }
```

Here is the call graph for this function:

Controller::getCommand ()

Getter method for the currentCmd pointer.

Returns:

Pointer to the Arm_Command object of the pointer

Definition at line 39 of file Controller.cpp.

```
   39                                                {
   40     return currentCmd;
   41 }
```

Controller::parseSerial (String *rawinput*)

Translates Serial or Other Input into an Arm_Command struct object This function will take the string input, and then parse based on the first character of the string. Parsing goes as follows: -Any commands related to rotation will start with an 'r' or 'R', then followed by a double indicating revolutions -Any commands related to gripping will start with a 'g' or 'G', then followed by a 0 or 1. Other values will trigger an exception -Any commands related to extending the arm will start with an 'e' or 'E', then followed by a double indicating motor rotation. We will assert that the value is between -1 and 1 for now -Any unrecognized letter will return the ERROR enum and 1 as the value. Serial should display an error message, or trigger an exception Once the command is

selected, the remaining value is parsed as a double, and then pointed to in the Arm_Command Pointer.

Parameters:

rawinput	The input from Serial or other communication method

Definition at line 47 of file Controller.cpp.

```
47                                                  {
48      Arm_Command* out = new Arm_Command();
49
50      //This is what we expect in terms of input
51      //Any commands related to rotation will start with an 'r' or 'R', then followed
by a double indicating revolutions
52      //Any commands related to gripping will start with a 'g' or 'G', then followed
by a 0 or 1. Other values will trigger an exception
53      //Any commands related to extending the arm will start with an 'e' or 'E',
then followed by a double indicating motor rotation. We will assert that the value is
between -1 and 1 for now
54      //Any unrecognized letter will return the ERROR enum and 1 as the value. Serial
should display an error message, or trigger an exception
55      if (rawinput.charAt(0) == 'r' || rawinput.charAt(0) == 'R') {
56          out->op = Controller::Arm_Operation::ROTATE;
57          out->value = atof(rawinput.substring(1, rawinput.length()).c_str());
//Extract Value from command string
58      } else if (rawinput.charAt(0) == 'g' || rawinput.charAt(0) == 'G') {
59          out->op = Controller::Arm_Operation::GRAB;
60          out->value = atof(rawinput.substring(1, rawinput.length()).c_str());
//Extract Value from command string
61      } else if (rawinput.charAt(0) == 'e' || rawinput.charAt(0) == 'E') {
62          out->op = Controller::Arm_Operation::EXTEND;
63          out->value = atof(rawinput.substring(1, rawinput.length()).c_str());
//Extract Value from command string
64      } else if (rawinput.charAt(0) == 'm' || rawinput.charAt(0) == 'M') {
65          out->op = Controller::Arm_Operation::MICROSTEPS;
66          //Special Extract Value related to enum
67
68      } else {
69          out->op = Controller::Arm_Operation::ERROR;
70          out->value = 1;
71      }
72      currentCmd = out;
73  }
```

Controller::printExec ()

Debug Function to test the parser Prints output in the serial connection based on the values in currentCmd.

See also:

Controller::parseSerial

Definition at line 75 of file Controller.cpp.

```
75                              {
76      Controller::Arm_Operation plsdo = this->getCommand()->op;
//Get the Arm Operation enum from the
77      switch (plsdo) {
//Figure out which command was extracted
78      case Controller::Arm_Operation::ROTATE:
//The command is to rotate the wrist
79          Serial.println("Command: Rotate");
//Print out the command name
80          Serial.println("Value: " + String(this->getCommand()->value));
//Print out the command value
81          Serial.println("");
//Newline to seperate output
82          break;
83      case Controller::Arm_Operation::GRAB:
//The command is to do the grabby hand
```

```
   84          Serial.println("Command: Grab");
//Print out the command name
   85          Serial.println("Value: " + String(this->getCommand()->value));
//Print out the command value
   86          Serial.println("");
//Newline to seperate output
   87          break;
   88      case Controller::Arm_Operation::EXTEND:
//The Command is to extend the arm
   89          Serial.println("Command: Extend");
//Print out the command name
   90          Serial.println("Value: " + String(this->getCommand()->value));
//Print out the command value
   91          Serial.println("");
//Newline to seperate output
   92          break;
   93      case Controller::Arm_Operation::MICROSTEPS:
   94          //Microsteps should be parsed diffenrently
   95          break;
   96      case Controller::Arm_Operation::ERROR:
//The command is an error
   97          Serial.println("Command: Error");
//Print out the command name
   98          Serial.println("Value: " + String(this->getCommand()->value));
//Print out the command value
   99          Serial.println("");
//Newline to seperate output
  100          break;
  101      }
  102 }
```

Here is the call graph for this function:

Controller::setCommand (Controller::Arm_Command * *cmd*)

Setter method for the currentCmd pointer.

Parameters:

cmd	The pointer to overwrite the current pointer of currentCmd

Definition at line 43 of file Controller.cpp.

```
   43                                                      {
   44      currentCmd = cmd;
   45 }
```

The documentation for this class was generated from the following files:

- Controller.h
- Controller.cpp

MotorConfig Class Reference

Contains constants related to motor configuration.

```
#include <cpu_map.h>
```

Static Public Attributes

- static const short MOTOR_STEPS = 200
- static const short MICROSTEPS = 1
- static const short MOTOR_RPM = 50

Detailed Description

Contains constants related to motor configuration.

Definition at line 34 of file cpu_map.h.

Member Data Documentation

const short MotorConfig::MICROSTEPS = 1 [static]

Constant used for defining the current microstep resolution

Definition at line 37 of file cpu_map.h.

const short MotorConfig::MOTOR_RPM = 50 [static]

Constant used for defining the RPM of the stepper motor (essentially speed)

Definition at line 38 of file cpu_map.h.

const short MotorConfig::MOTOR_STEPS = 200 [static]

Constant used for defining the number of motor steps per revolution of the stepper motor

Definition at line 36 of file cpu_map.h.

The documentation for this class was generated from the following file:

- cpu_map.h

Pinout Class Reference

Contains the pin definitions of the motor drivers and any other relevant pins.
`#include <cpu_map.h>`

Static Public Attributes

- static const short WRIST_ROT_STEP = 2
- static const short WRIST_ROT_DIR = 3
- static const short GRIP_STEP = 4
- static const short GRIP_DIR = 5
- static const short EXTEND_STEP = 6
- static const short EXTEND_DIR = 7

Detailed Description

Contains the pin definitions of the motor drivers and any other relevant pins.

Definition at line 14 of file cpu_map.h.

Member Data Documentation

const short Pinout::EXTEND_DIR = 7 [static]

The pin definition for the DIR pin on the motor driver responsible for arm extension motions

Definition at line 27 of file cpu_map.h.

const short Pinout::EXTEND_STEP = 6 [static]

The pin definition for the STEP pin on the motor driver responsible for arm extension motions

Definition at line 26 of file cpu_map.h.

const short Pinout::GRIP_DIR = 5 [static]

The pin definition for the DIR pin on the motor driver responsible for the grabbing motion

Definition at line 22 of file cpu_map.h.

const short Pinout::GRIP_STEP = 4 [static]

The pin definition for the STEP pin on the motor driver responsible for the grabbing motion

Definition at line 21 of file cpu_map.h.

const short Pinout::WRIST_ROT_DIR = 3 [static]

The pin definition for the DIR pin on the motor driver responsible for wrist rotation

Definition at line 18 of file cpu_map.h.

const short Pinout::WRIST_ROT_STEP = 2 [static]

The pin definition for the STEP pin on the motor driver responsible for wrist rotation

Definition at line 17 of file cpu_map.h.

The documentation for this class was generated from the following file:

- cpu_map.h

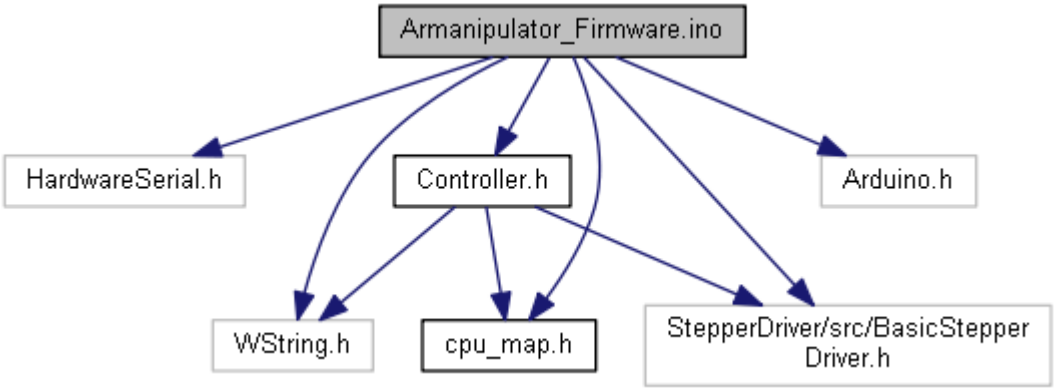
File Documentation

Armanipulator_Firmware.ino File Reference

Base file for the arduino sketch.

```
#include <HardwareSerial.h>
#include <WString.h>
#include "cpu_map.h"
#include "Controller.h"
#include "StepperDriver/src/BasicStepperDriver.h"
```

Include dependency graph for Armanipulator_Firmware.ino:



Functions

- void setup ()
  One-time initalization function The setup function is one of the required functions in an arduino sketch in order for it to work. This function serves as a "container" for all functions and operations that is to be run once, unless the RESET button is activated.

- void loop ()
  Main firmware loop The loop function is the other requried function in an arduino sketch in order for it to compile. This function serves as a contianer for all functions and operations that is to be repeated until the MCU receives no power or the RESET button is activated.

Variables

- Controller * mainController

Detailed Description

Base file for the arduino sketch.

Variable Documentation

Controller* mainController

Pointer to the Controller Object

Definition at line 13 of file Armanipulator_Firmware.ino.

Armanipulator_Firmware.ino

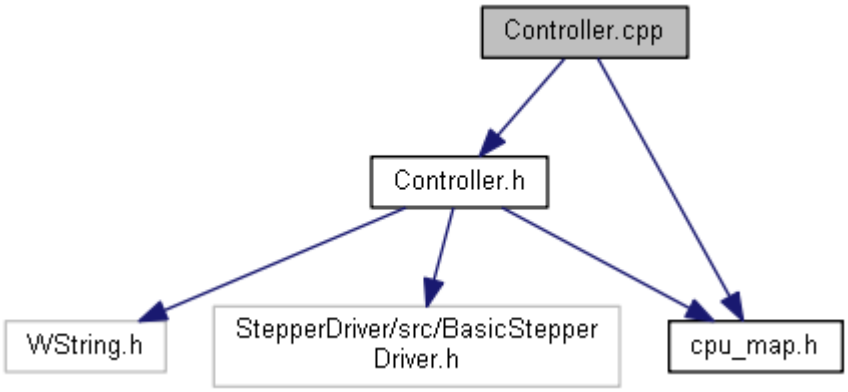
```
    1
    5 #include <HardwareSerial.h>
    6 #include <WString.h>
    7
    8 #include "cpu_map.h"
    9 #include "Controller.h"
   10 #include "StepperDriver/src/BasicStepperDriver.h"
   11
   12
   13 Controller* mainController;
   20 void setup() {
   21     Serial.begin(115200);                          //Open serial comms
   22     Serial.println("Success");                     //This implies the serial
connection opened successfully
   23     Serial.println("Enter command");               //Prompt user to enter
commands for parsing
   24     mainController = new Controller();            //Create a new Controller
object on the heap, initialize it, then have mainController point to it
   25 }
   26
   27
   33 void loop() {
   34     if (Serial.available() > 0) {               //Wait for serial input
   35         String serialIn = Serial.readString();  //Capture serial input
   36         mainController->parseSerial(serialIn);  //Parse serial input
   37         mainController->printExec();            //Print parsed serial input
(debugging)
   38         mainController->executeCmd();           //Execute the parsed input
   39     }
   40 }
```

Controller.cpp File Reference

Source File for the Controller Class Created on: May 11, 2019 Author: FaceF.

```
#include "Controller.h"
#include "cpu_map.h"
```

Include dependency graph for Controller.cpp:



Variables

- Controller::Arm_Command * currentCmd
- BasicStepperDriver * rotateDriver
- BasicStepperDriver * grabDriver
- BasicStepperDriver * extendDriver

Detailed Description

Source File for the Controller Class Created on: May 11, 2019 Author: FaceF.

Definition in file Controller.cpp.

Variable Documentation

Controller::Arm_Command* currentCmd

Pointer to the Arm_Command struct in use

Definition at line 11 of file Controller.cpp.

BasicStepperDriver* extendDriver

Pointer to the stepper driver responsible for arm extension

Definition at line 14 of file Controller.cpp.

BasicStepperDriver* grabDriver

Pointer to the stepper driver responsible for grab motions

Definition at line 13 of file Controller.cpp.

BasicStepperDriver* rotateDriver

Pointer to the Stepper Driver responsible for wrist rotation

Controller.cpp

```
    1
    7 //Include Me pls
    8 #include "Controller.h"
    9 #include "cpu_map.h"
   10
   11 Controller::Arm_Command* currentCmd;
   12 BasicStepperDriver* rotateDriver;
   13 BasicStepperDriver* grabDriver;
   14 BasicStepperDriver* extendDriver;
   15 //Bob the builder
   16 Controller::Controller() {
   17     //Init the motors
   18     //Declare Motor Pinouts
   19     rotateDriver = new BasicStepperDriver(MotorConfig::MOTOR_STEPS,
Pinout::WRIST_ROT_DIR, Pinout::WRIST_ROT_STEP);
   20     grabDriver = new BasicStepperDriver(MotorConfig::MOTOR_STEPS,
Pinout::GRIP_DIR, Pinout::GRIP_STEP);
   21     extendDriver = new BasicStepperDriver(MotorConfig::MOTOR_STEPS,
Pinout::EXTEND_DIR, Pinout::EXTEND_STEP);
   22
   23     //Start the motors
   24     rotateDriver->begin(MotorConfig::MOTOR_RPM, MotorConfig::MICROSTEPS);
   25     grabDriver->begin(MotorConfig::MOTOR_RPM, MotorConfig::MICROSTEPS);
   26     extendDriver->begin(MotorConfig::MOTOR_RPM, MotorConfig::MICROSTEPS);
   27
   28     //Init the struct as an error struct (If executeCMD is called before parseSerial,
error is returned intentionally)
   29     currentCmd = new Arm_Command{Controller::Arm_Operation::ERROR, 1};
   30 }
   31
   32 //DESTROY THE CHILD. CORRUPT THEM ALL
   33 Controller::~Controller() {
   34
   35 }
   36
   37 //Getters and Setters
   38 //Arm_Command
   39 Controller::Arm_Command* Controller::getCommand() {
   40     return currentCmd;
   41 }
   42
   43 void Controller::setCommand(Controller::Arm_Command* cmd) {
   44     currentCmd = cmd;
   45 }
   46
   47 void Controller::parseSerial(String rawinput) {
   48     Arm_Command* out = new Arm_Command();
   49
   50     //This is what we expect in terms of input
   51     //Any commands related to rotation will start with an 'r' or 'R', then followed
by a double indicating revolutions
   52     //Any commands related to gripping will start with a 'g' or 'G', then followed
by a 0 or 1. Other values will trigger an exception
   53     //Any commands related to extending the arm will start with an 'e' or 'E', then
followed by a double indicating motor rotation. We will assert that the value is between
-1 and 1 for now
   54     //Any unrecognized letter will return the ERROR enum and 1 as the value. Serial
should display an error message, or trigger an exception
   55     if (rawinput.charAt(0) == 'r' || rawinput.charAt(0) == 'R') {
   56         out->op = Controller::Arm_Operation::ROTATE;
   57         out->value = atof(rawinput.substring(1, rawinput.length()).c_str());
//Extract Value from command string
   58     } else if (rawinput.charAt(0) == 'g' || rawinput.charAt(0) == 'G') {
   59         out->op = Controller::Arm_Operation::GRAB;
   60         out->value = atof(rawinput.substring(1, rawinput.length()).c_str());
//Extract Value from command string
   61     } else if (rawinput.charAt(0) == 'e' || rawinput.charAt(0) == 'E') {
   62         out->op = Controller::Arm_Operation::EXTEND;
   63         out->value = atof(rawinput.substring(1, rawinput.length()).c_str());
//Extract Value from command string
   64     } else if (rawinput.charAt(0) == 'm' || rawinput.charAt(0) == 'M') {
   65         out->op = Controller::Arm_Operation::MICROSTEPS;
   66         //Special Extract Value related to enum
   67
```

```
    } else {
        out->op = Controller::Arm_Operation::ERROR;
        out->value = 1;
    }
    currentCmd = out;
}

void Controller::printExec() {
    Controller::Arm_Operation plsdo = this->getCommand()->op;
//Get the Arm_Operation enum from the
    switch (plsdo) {
//Figure out which command was extracted
    case Controller::Arm_Operation::ROTATE:
//The command is to rotate the wrist
        Serial.println("Command: Rotate");
//Print out the command name
        Serial.println("Value: " + String(this->getCommand()->value));
//Print out the command value
        Serial.println("");
//Newline to seperate output
        break;
    case Controller::Arm_Operation::GRAB:
//The command is to do the grabby hand
        Serial.println("Command: Grab");
//Print out the command name
        Serial.println("Value: " + String(this->getCommand()->value));
//Print out the command value
        Serial.println("");
//Newline to seperate output
        break;
    case Controller::Arm_Operation::EXTEND:
//The Command is to extend the arm
        Serial.println("Command: Extend");
//Print out the command name
        Serial.println("Value: " + String(this->getCommand()->value));
//Print out the command value
        Serial.println("");
//Newline to seperate output
        break;
    case Controller::Arm_Operation::MICROSTEPS:
        //Microsteps should be parsed diffenrently
        break;
    case Controller::Arm_Operation::ERROR:
//The command is an error
        Serial.println("Command: Error");
//Print out the command name
        Serial.println("Value: " + String(this->getCommand()->value));
//Print out the command value
        Serial.println("");
//Newline to seperate output
        break;
    }
}

//Execute Order 66
void Controller::executeCmd() {
    Controller::Arm_Operation cmdop = this->getCommand()->op;
//Extract the Arm_Operation enum stored in this controller object
    //TODO: Check efficiency of using a switch statement instead of an if/else chain
    //See Jump Tables and low-level intricacies produced by AVR
    switch (cmdop) {
//What is the command
    case Controller::Arm_Operation::ROTATE:
//Rotate the wrist
        if (this->getCommand()->value > 1 || this->getCommand()->value < -1) {
//Check the bounds of the value
            Serial.println("Error: Value for Wrist Rotation commands should be
between -1 and 1");  //Print an error message and give explanation
            return;
//Exits to the main loop if bounds are not met
        }
        rotateDriver->move(currentCmd->value * 200);
//Otherwise, perform the movement
        break;
    case Controller::Arm_Operation::GRAB:
//Do the grabby hand
```

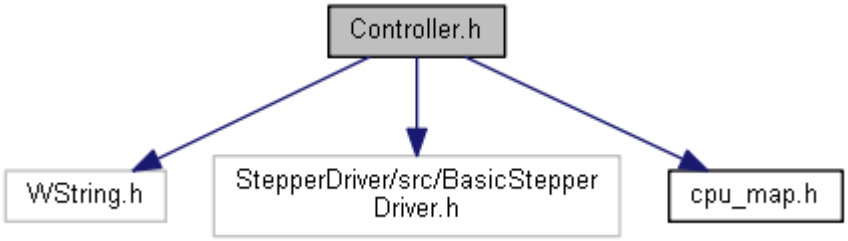
```
  118         if (this->getCommand()->value != 1 || this->getCommand()->value != -1) {
//Bounds checking, but stricter (Decimal values indicate a partial oepning of the hand,
which us not the best practice
  119             Serial.println("Error: Value for Grab commands should be either -1 or
1");              //Print an error message and give explanation
  120             return;
//Exits to the main loop if bounds are not met
  121         }
  122         grabDriver->move(currentCmd->value * 60);
//Do the grab (set value, motion depends on sign))
  123         break;
  124     case Controller::Arm_Operation::EXTEND:
//Extend the arm!
  125         if (this->getCommand()->value > 1 || this->getCommand()->value < -1) {
//Bounds checking, but not strict (arm should extend to variable lengths
  126             Serial.println("Error: Value for Arm Extension commands should be
between -1 and 1");   //Print an error message and give explanation
  127             return;
//Exits to the main loop if bounds are not met
  128         }
  129         extendDriver->move(currentCmd->value);
//Extend the arm
  130         break;
  131     case Controller::Arm_Operation::MICROSTEPS:
  132         //Empty for now
  133         break;
  134     case Controller::Arm_Operation::ERROR:
//He's Dead, Jim!
  135         Serial.println("Error, Unrecognized Command");
//Print out an error message
  136         break;
  137
  138     }
  139 }
```

Controller.h File Reference

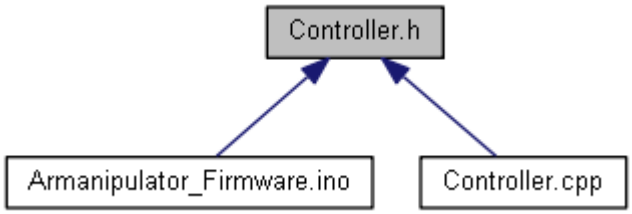
Contains methods for controlling the motors and parsing input.

```
#include <WString.h>
#include "StepperDriver/src/BasicStepperDriver.h"
#include "cpu_map.h"
```

Include dependency graph for Controller.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Controller
  The class-object used to interface between the serial interface and the stepper motor drivers.
- struct Controller::arm_command
  The structure used to define the overall instruction parsed from serial input.

Detailed Description

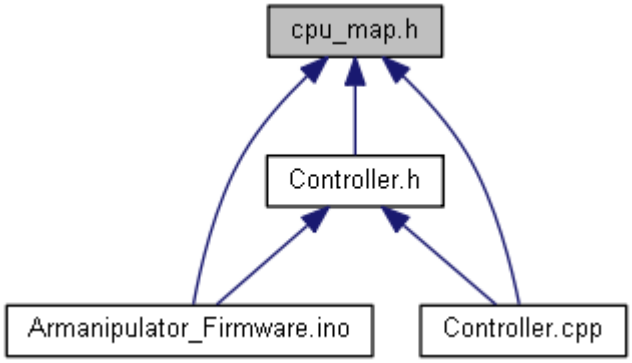
Contains methods for controlling the motors and parsing input.

Controller.h

```
  1
  6 #ifndef CONTROLLER_H_
  7 #define CONTROLLER_H_
  8
  9 //Includes
 10
 11 #include <WString.h>        //Arduino String Library (because C strings are not fun)
 12
 13 #include "StepperDriver/src/BasicStepperDriver.h"
 14 #include "cpu_map.h"
 15
 20 class Controller {
 21
 22 public:
 23
 28     typedef enum {
 29         ROTATE,
 30         GRAB,
 31         EXTEND,
 32         MICROSTEPS,
 33         ERROR
 34     } Arm_Operation;
 35
 42     typedef enum microsteps {
 43         HALF,
 44         QUARTER,
 45         EIGTH,
 46         SIXTEENTH,
 47         THIRTY_SECOND
 48     } Microsteps;
 49
 58     typedef struct arm_command {
 59         Controller::Arm_Operation op;
 60         double value;
 61     } Arm_Command;
 62
 63     //Constructor/Destructor
 69     Controller();
 70
 75     virtual ~Controller();
 76
 77     //Parser Related Methods
 89     void parseSerial(String rawinput);
 95     void printExec();
 96
 97     //Do stuff methods
103     void executeCmd();
104
105     //Getter & Setter for variables
106     //Arm_Command cmd
111     Arm_Command* getCommand();
112
117     void setCommand(Arm_Command* cmd);
118
119     //Private Vars
120 private:
124     Arm_Command* currentCmd;
125
130     BasicStepperDriver* rotateDriver;
131
136     BasicStepperDriver* grabDriver;
137
142     BasicStepperDriver* extendDriver;
143 };
144
145 #endif /* CONTROLLER_H_ */
```

cpu_map.h File Reference

Pin mapping and common configuration Created: 4/24/2019 9:38:26 PM Author: Alex Westerman
Used to define pins in relation to the stepper motor drivers.
This graph shows which files directly or indirectly include this file:



Classes

- class Pinout
  Contains the pin definitions of the motor drivers and any other relevant pins.
- class MotorConfig
  Contains constants related to motor configuration.

Detailed Description

Pin mapping and common configuration

Used to define pins in relation to the stepper motor drivers.

cpu_map.h

```
 1
 8 #ifndef CPU_MAP
 9 #define CPU_MAP
10
14 class Pinout {
15 public:
16 //Define Wrist Driver Pins
17     const static short WRIST_ROT_STEP = 2;
18     const static short WRIST_ROT_DIR = 3;
20 //Define Grip Driver Pins
21     const static short GRIP_STEP = 4;
22     const static short GRIP_DIR = 5;
24 //Define Arm Motion Pins
25
26     const static short EXTEND_STEP = 6;
27     const static short EXTEND_DIR = 7;
29 };
30
34 class MotorConfig {
35 public:
36     const static short MOTOR_STEPS = 200;
37     const static short MICROSTEPS = 1;
38     const static short MOTOR_RPM = 50;
39 };
40
41 #end
```