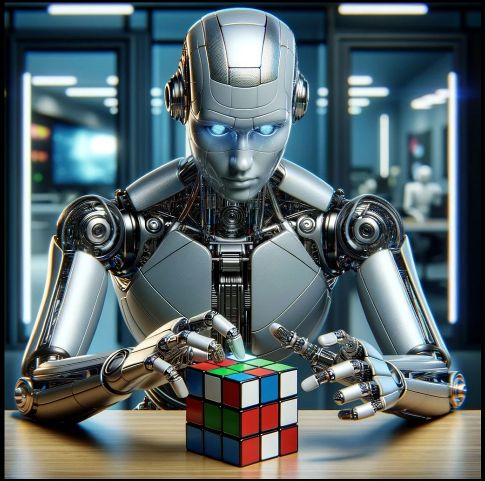
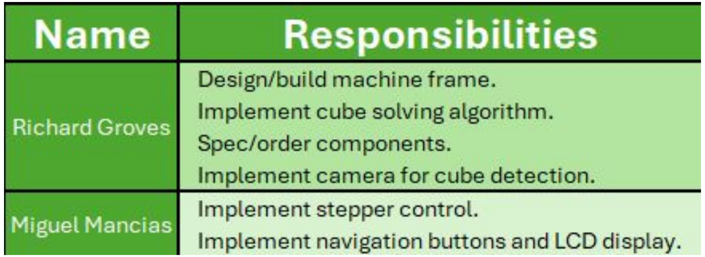
Rubik’s Cube Solving Machine



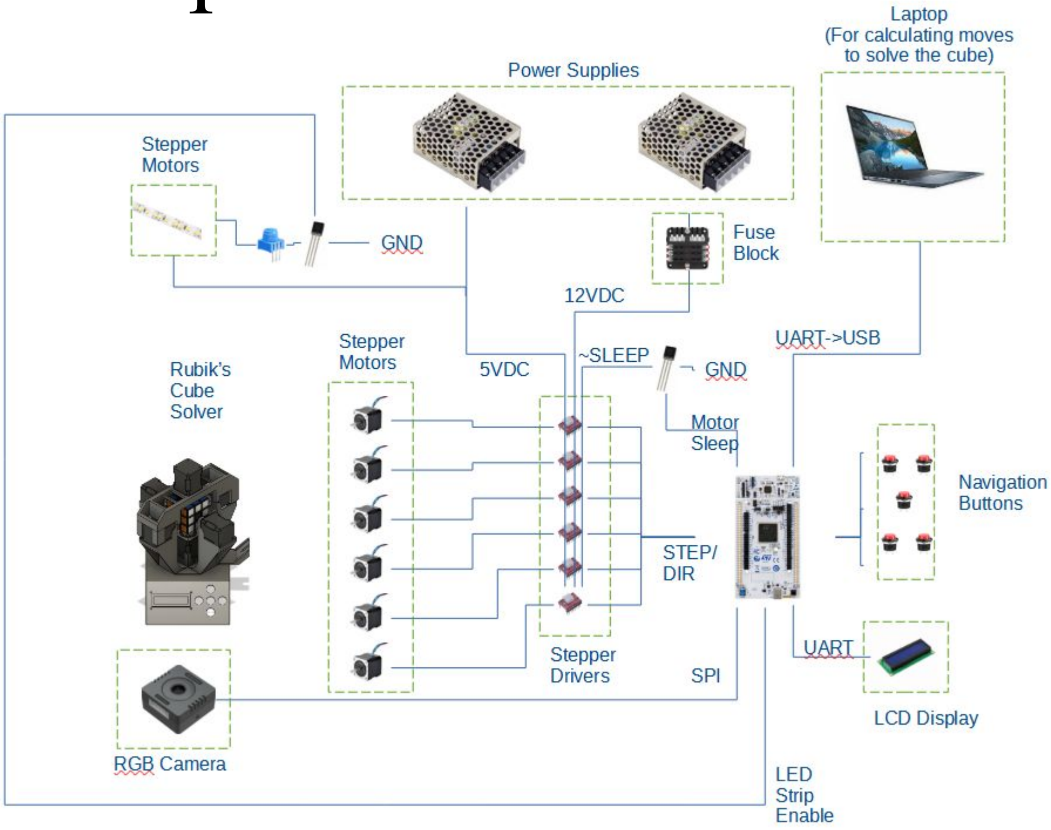
Miguel Mancias

Richard Groves

Overview & Responsibilities



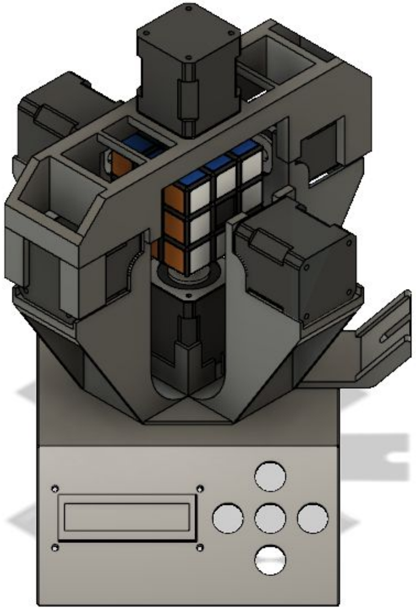
Name	Responsibilities
Richard Groves	Design/build machine frame. Implement cube solving algorithm. Spec/order components. Implement camera for cube detection.
Miguel Mancias	Implement stepper control. Implement navigation buttons and LCD display.

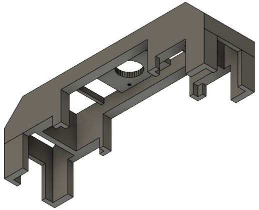


BOM

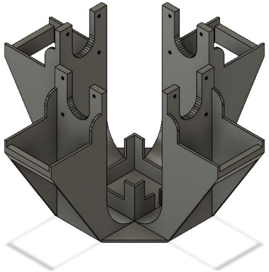
Item	Unit Price	Qty	Ext Price
0-48VDC Power Supply	$ 39.99	1	$ 39.99
5VDC Power Supply	$ 18.99	1	$ 18.99
Fuse Block	$ 9.99	1	$ 9.99
2A Fuses	$ 0.13	6	$ 0.78
Push Buttons	$ 0.48	5	$ 2.40
Transistors	$ 0.38	2	$ 0.76
LED Strip	$ 5.94	1	$ 5.94
Stepper Motors	$ 10.99	6	$ 65.94
Stepper Drives	$ 2.89	6	$ 17.34
Microcontroller	$ 21.28	1	$ 21.28
Camera	$ 34.99	1	$ 34.99
PLA Filament	$ 12.59	1	$ 12.59
LCD Display w/ Backpack	$ 24.95	1	$ 24.95
100uF Capacitors	$ 0.14	6	$ 0.86
Total			**$ 256.80**

Machine Design

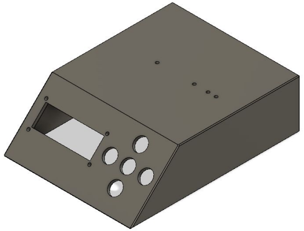




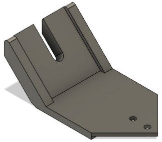
Motor Mount - Top



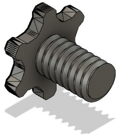
Motor Mount - Main



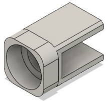
Machine Base



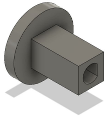
Camera Mount



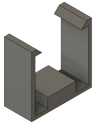
Camera Mounting Bolt



Coupler - Cube



Coupler - Motor

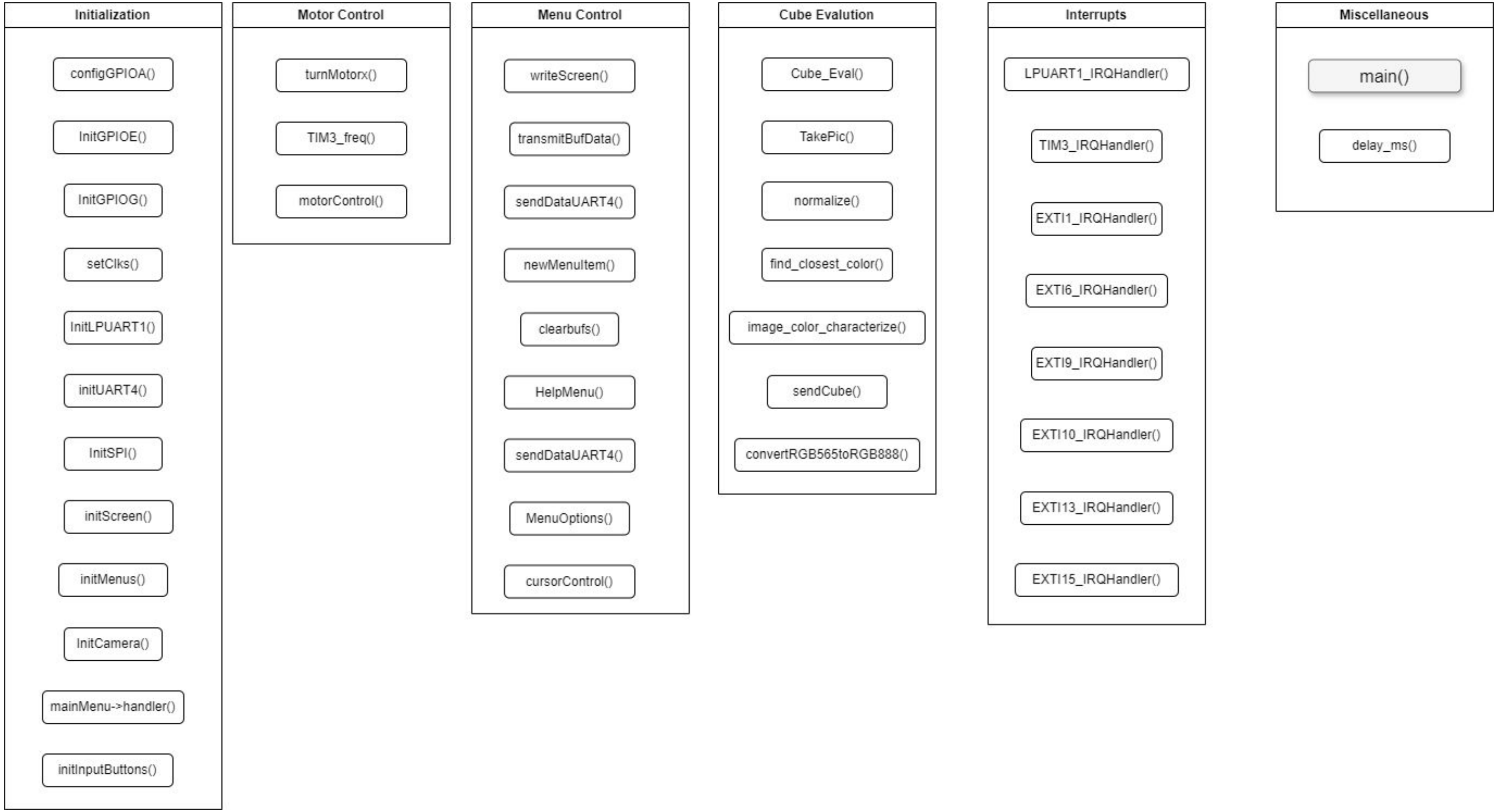


Coupler - Retainer Clip

I/O

- Motors
  - GPIOA.0-9&11-12: Step and Direction for each motor.
  - GPIOA.14: Motor Sleep.
- Comms
  - GPIOC.10: UART to LCD Display
  - GPIOE.12-15: SPI to Camera.
  - GPIOG.7-8: LPUART to Laptop
- Miscellaneous
  - GPIOC.13, GPIOE.6&10, GPIOF.1&9&15: Buttons
  - GPIOD.15: LED Strip On/Off

Function List



Initialization

- configGPIOA()
- InitGPIOE()
- InitGPIOG()
- setClks()
- InitLPUART1()
- initUART4()
- InitSPI()
- initScreen()
- initMenus()
- InitCamera()
- mainMenu->handler()
- initInputButtons()

Motor Control

- turnMotorx()
- TIM3_freq()
- motorControl()

Menu Control

- writeScreen()
- transmitBufData()
- sendDataUART4()
- newMenuItem()
- clearbufs()
- HelpMenu()
- sendDataUART4()
- MenuOptions()
- cursorControl()

Cube Evalution

- Cube_Eval()
- TakePic()
- normalize()
- find_closest_color()
- image_color_characterize()
- sendCube()
- convertRGB565toRGB888()

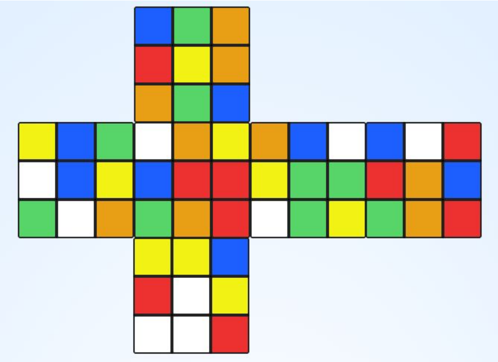
Interrupts

- LPUART1_IRQHandler()
- TIM3_IRQHandler()
- EXTI1_IRQHandler()
- EXTI6_IRQHandler()
- EXTI9_IRQHandler()
- EXTI10_IRQHandler()
- EXTI13_IRQHandler()
- EXTI15_IRQHandler()

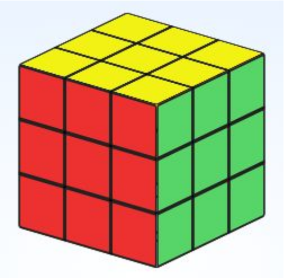
Miscellaneous

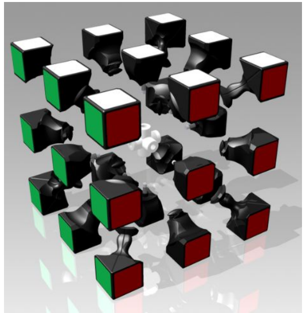
- main()
- delay_ms()

Cube Basics

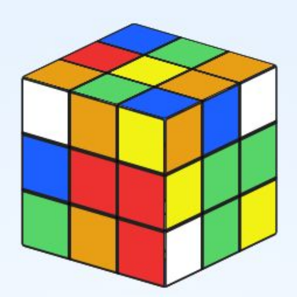


- 6 faces with 9 stickers (54 total stickers)
- 20 moving pieces (8 corners, 12 edges)
- Corner pieces have 3 possible orientations
- Edge pieces have 2 possible orientations
- 43,252,003,274,489,856,000 permutations
- All permutations can be solved within 20 moves (God’s Number)

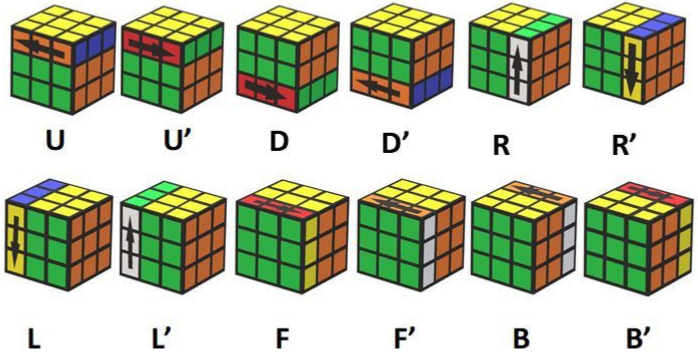




$$\mathit{Total\ Permutations} = \frac{8! \times 3^7 \times 12! \times 2^{11}}{3 \times 2 \times 2}$$

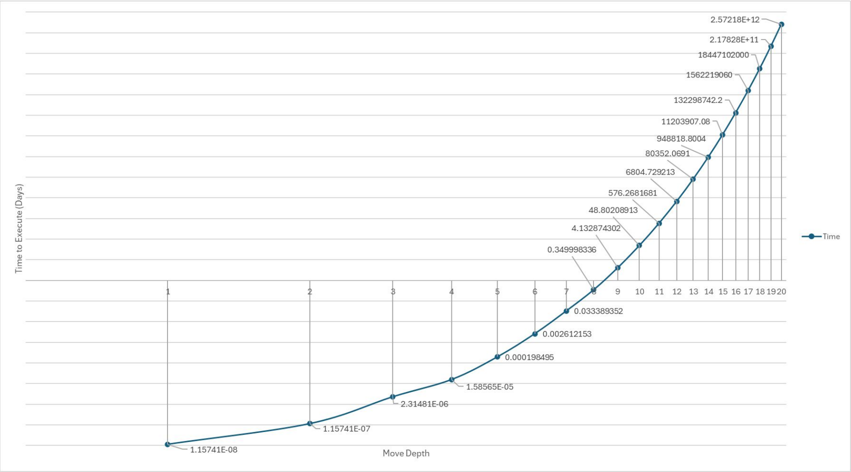


Cube Solving-
Move Codes



- Face Moves
  - L, R, F ,B, U, D(Left, Right, Front, Back, Up Down)
  - 90° Clockwise Turn
- Modifiers
  - 2
    - 180° Turn
  - ‘ (prime)
    - Counterclockwise Turn
    - Applies to 90° or 180° Turns

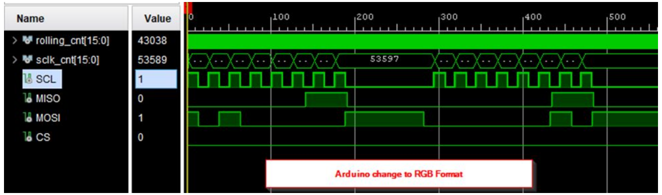
Cube Solving- Algorithms



- Iterative Deepening A* (IDA*)
  - Iterate through all 1-move solutions to the cube.
  - If the cube isn’t solved, iterate through all 2-move solutions to the cube. Etc....
- Move Pruning
  - FF+F = F’
  - F+F’ = NULL
- Kociemba’s Two Phase Algorithm
  - There is a cube state subset (G1) that all cube permutations can be moved to.
  - Phase 1 - Use IDA* to get to G1
  - Phase 2 - Use IDA* to solve the cube

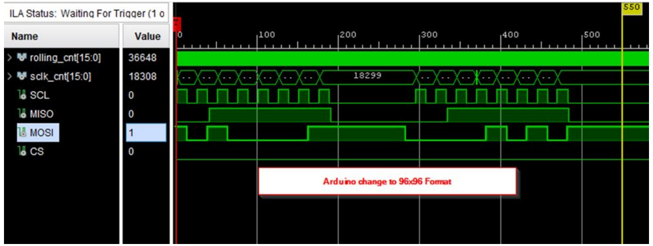
Camera-
Comms & Config

- Arucam Mega-5MP
  - SPI Protocol
  - 8MHz SCK (suggested)
  - JPEG, RGB, YUV Output formats
  - Automatic focus, brightness, contrast and saturation control
  - 12 selectable resolutions
    - From 96x96 to 2592x1944



Name	Value
rolling_cnt[15:0]	43038
sclk_cnt[15:0]	53589
SCL	1
MISO	0
MOSI	1
CS	0

Arduino change to RGB Format



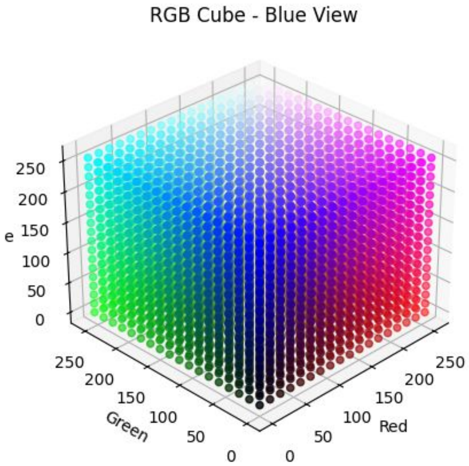
ILA Status: Waiting For Trigger (1 o

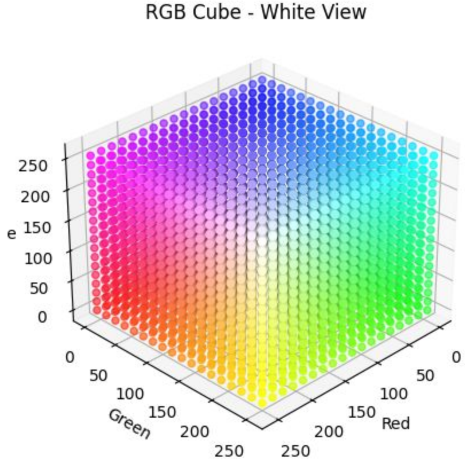
Name	Value
rolling_cnt[15:0]	36648
sclk_cnt[15:0]	18308
SCL	0
MISO	0
MOSI	1
CS	0

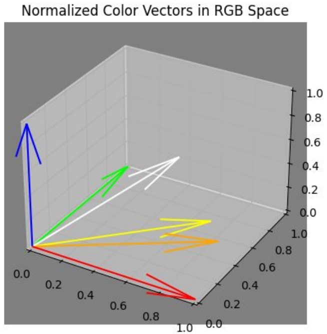
Arduino change to 96x96 Format

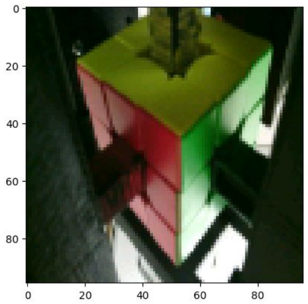
Camera-
Color Detection

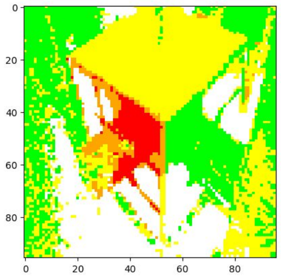
- Normalize RGB values.
- Find similarity of each pixels RGB vector with each predefined colors' vector. (dot product)
- Find the greatest similarity, and classify the pixel as the associated predefined color.

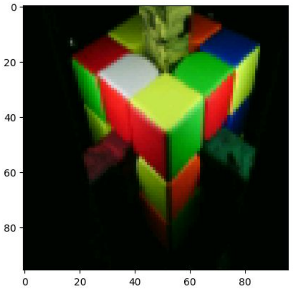

RGB Cube - Blue View

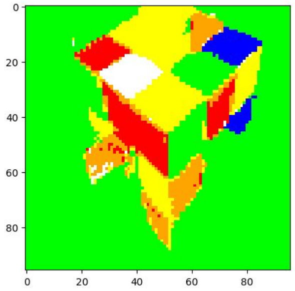

RGB Cube - White View


Normalized Color Vectors in RGB Space

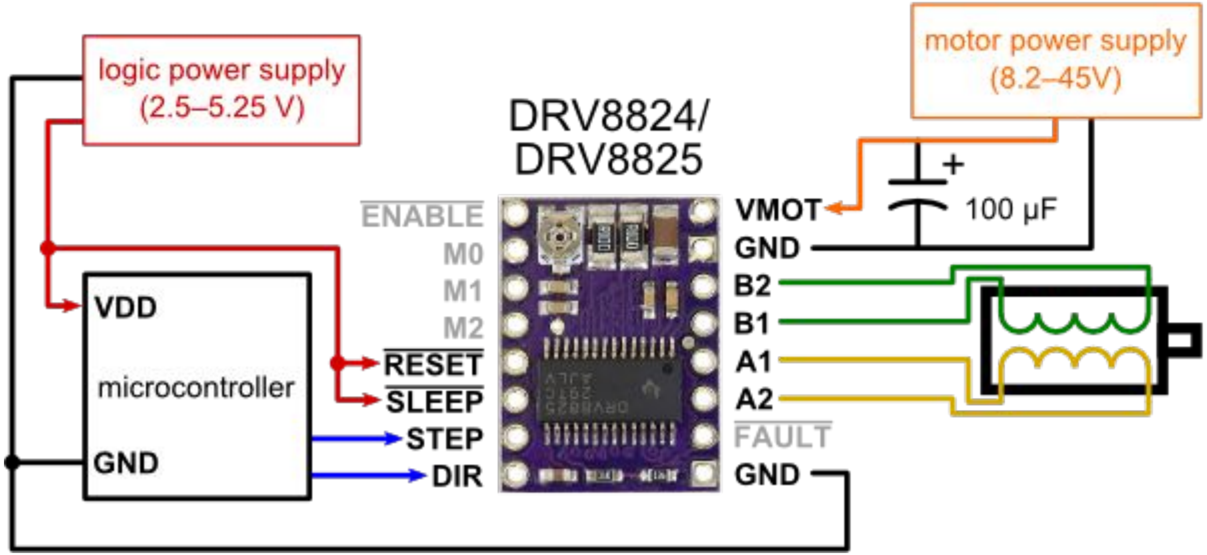




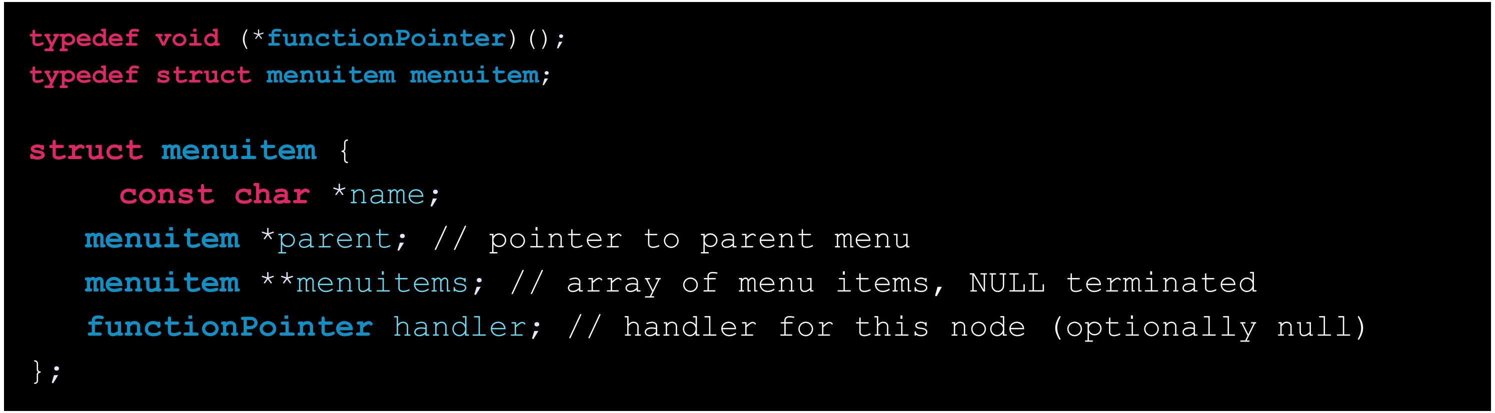




Motor Control



Menu Structure



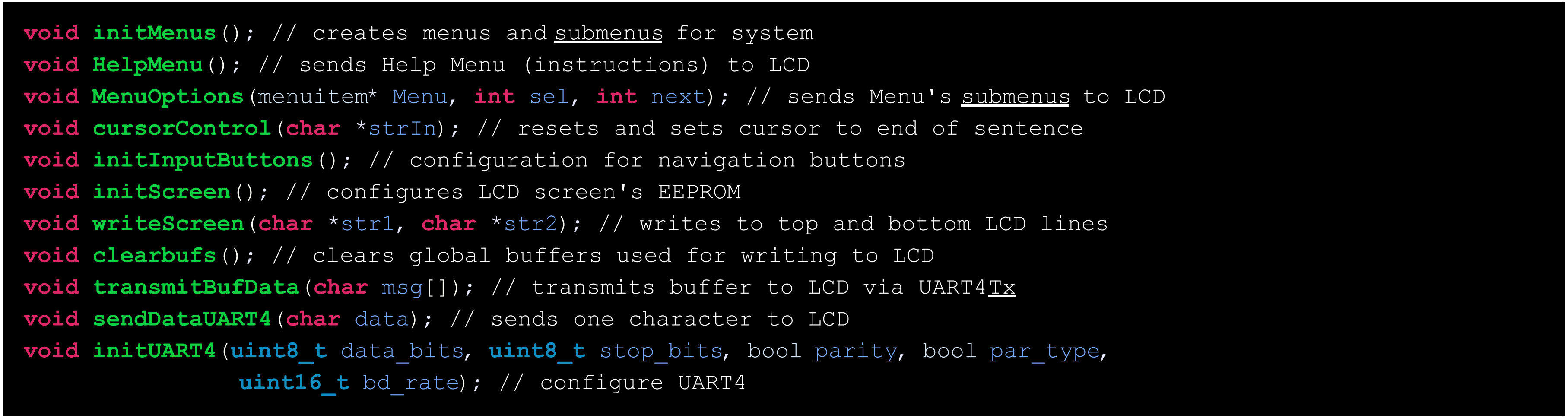
```c
typedef void (*functionPointer)();
typedef struct menuitem menuitem;

struct menuitem {
     const char *name;
   menuitem *parent; // pointer to parent menu
   menuitem **menuitems; // array of menu items, NULL terminated
   functionPointer handler; // handler for this node (optionally null)
};
```

Menu Structure

```
menuitem* newMenuItem(menuitem* parentMenu, char *strName, int numMenuItems, int arrayPos){
      /**
       *    Creates top or sub menu.
       *
       *    parentMenu:   menuitem parent, or top menu for created menu
       *    strName:      name of menu (or submenu)
       *    numMenuItems: number of submenus for created menu
       *    arrayPos:     used to indicate index of parent's submenu array
       */
      menuitem* newMenuItem = (menuitem*)malloc(sizeof(menuitem)); // assigning size to memory
      memset(newMenuItem, 0, sizeof(menuitem)); // make sure memory is set to 0
      newMenuItem->menuitems = (menuitem**)calloc(sizeof(menuitem*), numMenuItems); // assigning submenu size
      newMenuItem->name = strName;
      if (parentMenu) {
            parentMenu->menuitems[arrayPos] = newMenuItem; // assigning newMenu to parent menuitems (submenus)
            newMenuItem->parent = parentMenu; // assigning newMenu's parent
      }
      return newMenuItem;
}
```

Menu Logic



```
void initMenus(); // creates menus and submenus for system
void HelpMenu(); // sends Help Menu (instructions) to LCD
void MenuOptions(menuitem* Menu, int sel, int next); // sends Menu's submenus to LCD
void cursorControl(char *strIn); // resets and sets cursor to end of sentence
void initInputButtons(); // configuration for navigation buttons
void initScreen(); // configures LCD screen's EEPROM
void writeScreen(char *str1, char *str2); // writes to top and bottom LCD lines
void clearbufs(); // clears global buffers used for writing to LCD
void transmitBufData(char msg[]); // transmits buffer to LCD via UART4Tx
void sendDataUART4(char data); // sends one character to LCD
void initUART4(uint8_t data_bits, uint8_t stop_bits, bool parity, bool par_type,
               uint16_t bd_rate); // configure UART4
```

State	0	1	2	4	8	16	32	64	128	256
Name	Main	Help	Motor	Auto	MotorR	MotorL	MotorU	MotorD	MotorF	MotorB

Lessons Learned

Richard

- Don’t trust manufacturers to have good documentation. Even if their products are popular. (I’m looking at you Arducam😐)
- A Quality Cube = Less Headaches

Miguel

- Through-holes are not to be trusted
- Test proper connection before assuming there’s an issue with the code
- Minimize variable checks

Questions?