**Software Documentation & User Guide**

Table of Contents

**How to Save, Access, & Use Arduino Files and Libraries** 1-2

**Common Errors while Uploading to Board** 3

**Ground Station Arduino**  4-5

**Go-Kart Arduino**  6-8

**Turret Arduino** 9

**Software Applications Used** 10

**How to Save, Access, & Use Arduino Files and Libraries**

It is important that every file you create and save within Arduino IDE is saved as the same filename as the folder it is in. That being said, your Arduino file name must be placed in a folder, with the same name, and this folder should then be placed in the Arduino folder created on your computer upon downloading Arduino IDE.

For example, on my Windows machine, my Arduino folder is located in:

**This PC 🡪 Documents 🡪 arduino-1.8.9-windows 🡪 arduino-1.8.9**

The folder **arduino-1.8.9** holds a **libraries** folder. This folder is where all libraries you download and use within your project should be kept for all of your files to successfully compile.

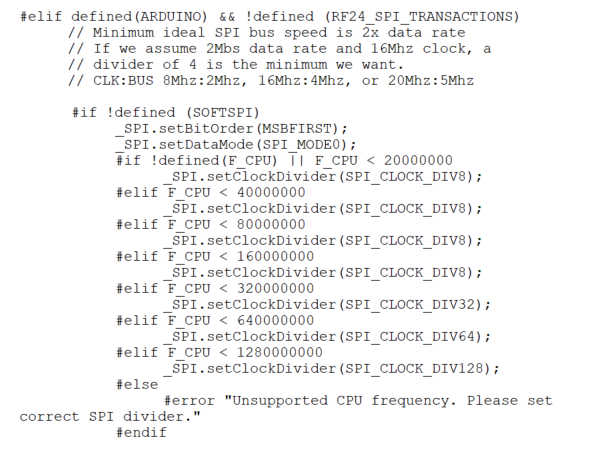
For this project, Arduino files were created for:

1. Go-Kart
2. Ground Station
3. Turret

For each Arduino file, a different library was downloaded and/or used, which are as follows:

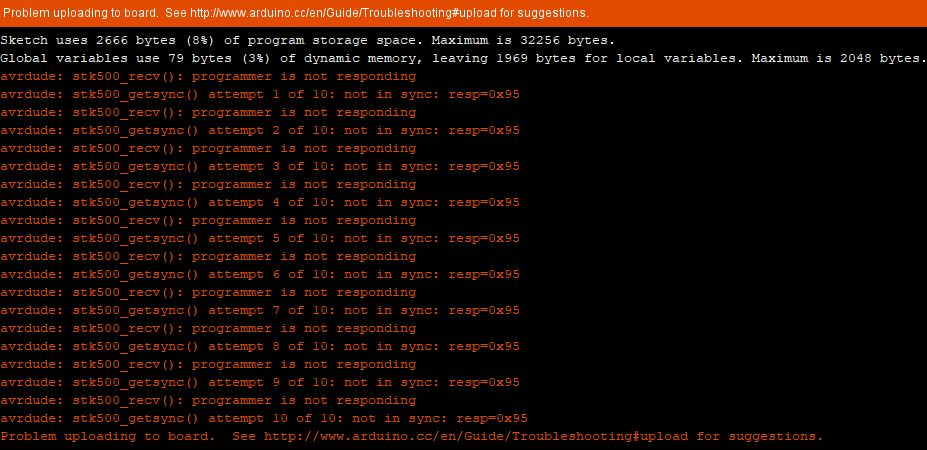
1. Go-Kart
   * Servo.h
   * SPI.h
   * nRF24L01.h
   * RF24.h
   * Wire.h
   * LiquidCrystal\_I2C.h
2. Ground Station
   * SPI.h
   * nRF24L01.h
3. Turret
   * Servo.h
   * SPI.h
   * RF24.h

Descriptions of each of the aforementioned header files and their functions are provided below:

* + Servo.h
    - Provided by Arduino (Y/N): **Y**
    - **Use:** Allows an Arduino board to control RC (hobby) servo motors
    - **What:** Servos have integrated gears and a shaft that can be precisely controlled. Standard servos allow the shaft to be positioned at various angles, usually between 0 and 180 degrees. Continuous rotation servos allow the rotation of the shaft to be set to various speeds
    - More information found on the Arduino reference page [here](https://www.arduino.cc/en/reference/servo)
  + SPI.h
    - Provided by Arduino (Y/N): **Y**
    - **Use:** Allows user to communicate with SPI devices, with the Arduino as the master device
    - **What:** SPI (Serial Peripheral Interface) is a synchronous serial data protocol used by microcontrollers for communicating with one of more peripheral devices quickly over short distances
      * Typically there are three lines common to all the devices:
        1. MISO (Master In Slave Out) – The Slave line for sending data to the master
        2. MOSI (Master Out Slave In) – The Master line for sending data to the peripherals
        3. SCK (Serial Clock) – The clock pulses which synchronize data transmission generated by the master
      * And one line specific for every device:
        1. SS (Slave Select) – The pin on each device that the master can use to enable and disable specific devices
    - More information found on the Arduino reference page [here](https://www.arduino.cc/en/reference/SPI)
  + nRF24L01.h
    - Provided by Arduino (Y/N): **N, available for download** [**here**](https://github.com/maniacbug/RF24)
    - **Use:** Developer of RF24 library has created a few more useful libraries to simplify using a NRF24L01 radio or even add new features to it
    - **What:** Newly updated/optimized NRF24L01 library for Arduino includes features/fixes from various forks and major updates
  + RF24.h
    - Provided by Arduino (Y/N): **N, available for download** [**here**](https://github.com/maniacbug/RF24)
    - **Use:** Enables user to utilize different functions associated with the nRF24L01 sensor, such as opening a reading/writing pipe
    - **What:** Optimized high speed driver for nRF24L01 Wireless Transceiver/Receiver
    - **\*NOTE\*** A change we included in our code was within the RF24.cpp file within the folder containing the .h file. The .cpp file assumes a 2Mbs data rate and a 16Mhz clock, so a divider of 4 would be the minimum necessary. We saw that our data was moving at a slow rate, and thus we divided by 8 rather than 4, as shown in the code sample to the right taken from the RF24.cpp file
  + Wire.h
    - Provided by Arduino (Y/N): **Y**
    - **Use:** Allows user to communicate with I2C/TWI devices
    - More information found on the Arduino reference page [here](https://www.arduino.cc/en/reference/wire)
  + LiquidCrystal\_I2C.h
    - Provided by Arduino (Y/N): **N, available for download** [**here**](https://github.com/fdebrabander/Arduino-LiquidCrystal-I2C-library)
    - **Use:** Allows user to control I2C displays with functions similar to LiquidCrystal library

**Common Errors while Compiling and Uploading to Board**

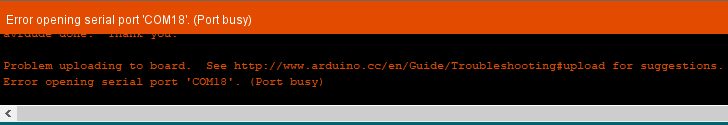
1. **avrdude: stk500\_recv(): programmer is not responding**

* This means that the board is not in sync. The sync, resp = 0x00, is a generic response that translates to the “Atmega chip is not working” on the Arduino
* This error most likely occurs if pins are connected to TX and RX while code is being upload, for this project, in terms of the Bluetooth module
* To fix this error, remove the two pins connected to TX and RX. Once they are removed, compile & upload the code to the Arduino board. **Make sure to select a port before running!**
* Only once your code has been successfully compiled & uploaded may you insert the TX and RX pins onto the board
* Also ensure that the correct COM port and board were selected under the Tools menu
* Press the reset button the Arduino a couple of times and re-upload the code
* Restart the Arduino IDE
* ****If none of the above works, consider that you may be using a “faulty” Arduino board, or a port on your laptop or device you are using has been fried (If your port is grayed out, this is most likely the situation)

1. **Correct port was not selected**

* As explained above, be sure to select the correct COM port and board under the Tools menu

1. **Error Opening serial port ‘COM\_\_’. (Port busy)**

* This is one of the simplest errors to fix
* This usually occurs when you try to upload code to an Arduino while the serial monitor is already open, or when you try to launch the serial monitor when the Arduino is talking to another software or device via the serial port
* In essence, it occurs when you try to use the Arduino serial port for two different things at the same time

**Ground Station Arduino**

**What you will need:**

1. RF Sensor (NRF24L01)
2. Thumb Joystick Module
3. Bluetooth Module

**Code:**

Include/Define

* Include SPI, RF24 and nRF24L01 .h files (header files)

Global Variables

**RF24**

* To utilize RF, radio configuration must take place. Create an RF24 object, defining the CSN and CE pins being used as the two argument
  + For example, pins 7 & 8 could be used for CSN and CE
* The Ground Station will be communicating with the Go-Kart and Turret, and thus a byte array in the form of a matrix will be needed to differentiate whether the RF is dealing with the Ground Station, Go-Kart, and Turret
  + The size of each must be 6, so for example: “turrt”, “gkart”, and “grnds”
* Set the radio number and role, both Booleans, equal to 1, to define the Ground Station as a transmitter

**Joystick**

* The up/down and left/right joystick functions will be defined as integers with the value of either 0 or 1, which represent their analog pin values
* Define the input values for both up/down and left/right as floats
* Define the output values for both up/down and left/right as integers
* Set the limits of the given joystick for up, down, left, and right

**E-Stop**

* Set the E-Stop ‘estop’ to the pin of your choice
* Create a char of ESTOP

void parseCommand(char input) { function

* For each case scenario on the Android application, so ‘1’, ‘2’, ‘3’…. ‘c’, call switch(input) and break after each case

float convertForeAftToServo(float y) { function

* Make a result variable of type int
* Call the map function, set it equal to result, then return result
  + result = map(y, Aft\_Limit, Fore\_Limit, 10, 170); /\*0 to 180\*/

float convertLeftRightToServo(float x) { function

* Make a result variable of type int
* Call the map function, set it equal to result, then return result
  + result = map(x, Left\_Limit, Right\_Limit, 10, 170); /\*0 to 180\*/

void setup() { function

**Before beginning Serial, set the E-Stop to INPUT\_PULLUP**

**After beginning Serial, begin the radio**

* Set the PA Level to HIGH, which describes the distance between the nRF24L01 receiver and transceiver

void loop() { function

**Begin by writing code to move the servos on the Turret**

* Open the writing pipe to whichever location of your byte address of the RF calls the Turret
* Analog read both the Up/Down and Left/Right joystick values and set them equal to the floats defined earlier as the input values
* Call the function convertForeAftToServo() with the input value as the parameter and set the result equal to the output value defined in the global variables as an int
  + Do the same thing, but for convertLeftRightToServo()

**Stop listening to the radio**

* Include an optional delay
* Radio write both the Up/Down and Left/Right output values

**Check to see if Serial is available**

* If so, Serial read the Bluetooth value defined as a global variable in your header file
* Call the parseCommand() function on that Bluetooth value
* Include an optional delay

**If the digital read of the E-Stop is LOW**

* Set your Bluetooth value equal to char ‘T’

**Open the writing pipe**

* Include a parameter defining the location in your byte address that addresses the Go-Kart
* Radio write. The Bluetooth value will now be written to this address and sent to the Go-Kart

**Set Bluetooth value equal to ‘2’**

**Go-Kart Arduino**

**What you will need:**

1. RF Sensor (NRF24L01)
2. Two Servos
3. Relay
4. E-Stop
5. IR Sensors

**Code:**

Include/Define

* Include Servo, SPI, nRF24L01, RF24, Wire, and LiquidCrystal\_I2C .h files (header files)

Global Variables

**RF24**

* To utilize RF, radio configuration must take place. Create an RF24 object, defining the CSN and CE pins being used as the two argument
  + For example, pins 7 & 8 could be used for CSN and CE
* The Go-Kart will be communicating with the Ground Station and Turret, and thus a byte array in the form of a matrix will be needed to differentiate whether the RF is dealing with the Ground Station, Go-Kart, and Turret
  + The size of each must be 6, so for example: “turrt”, “gkart”, and “grnds”

**IR Blasters**

* Define where the two IR pins will be on your Arduino board as integers
  + Ex. int irPin1 = 2;

**E-Stop**

* Will need code for both E-Stops, one of type int and one of type Boolean

**Millis**

* Rather than using delays, which is not good practice for large Arduino files (and really just in general!), we will use Millis() instead
* *For the IR Blasters,* set a time to begin at, customarily 0, of type unsigned long, and how long the delay will be, of type int, and make the variable names related to the IR Blasters
  + Ex. unsigned long startingIRtime = 0;
  + Ex. int delayIR = 3000;
* Create two other beginning times and the delays needed

**Servos**

* Define two servo objects
* Define a results object

**LCD Screen Configuration**

* Set up an LCD address
* Parameters will depend on what kind of LCD screen was purchased
  + In this case, the three parameters will be 0x27 for 16 chars and a 2-line display

void setup() { function

**Before beginning Serial, set up the LCD Screen and your two servos for the deployment mechanism**

* Initialize the LCD Screen and open its backlight
* Attach your two servos to pins of your choice and write each servo to position 0 after attaching to have it start at its initial position
* Now you may Serial.begin(). I recommend using 9600 bits per second as the parameter

**After beginning Serial, set the pinMode for each of the IR pins as well as the E-Stop**

* Set both IR pins to INPUT
* Set E-Stop pin to OUTPUT

**Begin the radio**

* To begin receiving, call radio.begin()
* Open the reading pipe, and set your address to Go-Kart as defined in your byte address from the global variables
* Set the PA Level to HIGH, which describes the distance between the nRF24L01 receiver and transceiver
* Start and stop listening the radio

void loop() { function

**As we started our setup() with the LCD Screen, we will handle it first within our loop function as well**

* Set the LCD Cursor to column 3, line 0, by taking in 3 and 0 as your parameters. This is the first line of your LCD Screen
* Once the cursor is set, you may lcd.print(); text of your choice

**Now, focus on your IR pins and tracking the score of the competition**

* Digital read both IR pins. Set each value to two separate Booleans
* A Boolean statement will be needed to ensure that the score does not print continuously but just once every 5 seconds by incorporating delays using millis()

**Determine what will be printed to the LCD Screen by setting pushed == false for both IR pins 1 & 2**

* If the first IR pin is set to LOW (0) and the delay in millis() is in effect
  + Digital write your E-Stop to LOW
  + Set pushed equal to true
  + Clear the LCD Screen and set the Cursor to column 10, line 1, in order to print the continuously updated score centered and on the second line
  + Now print the decremented score based on how many times the IR was hit to the LCD Screen by creating a variable, for example, ‘IRcounter’, to store the updated value
* If pushed does not == false
  + If the first IR pin is set to HIGH (1), then the IR is not being hit
  + Set pushed equal to false

**If there is a delay, which will be checked by using millios()**

* Digital write the E-Stop to HIGH
* Set pushed equal to false

**Start listening**

* If the radio is available, read the value of sent by the Bluetooth module
* If the value is equal to char ‘T’, digital write the E-Stop to LOW and include while(1)
* If the value is equal to ‘0’, move the servo with a for loop from 0 to 180 degrees and write to the servo with a delay of 20. Before moving the servo back with another for loop, be sure to include a millis() delay between the two for loops so that the defense mechanism is closed for 5 seconds before preceding
  + This will be if the RIGHT defense mechanism is pressed, and will move the RIGHT servo on the Go-Kart
* If the value is equal to ‘1’, perform the same code
  + This will be if the LEFT defense mechanism is pressed, and will move the LEFT servo on the Go-Kart
* If the value is equal to ‘s’
  + Clear the LCD Screen
  + Set the LCD Cursor to (3,0)
    - Print Score:
  + Set the LCD Cursor to (10,0)
    - Print the score derived from your ‘IRcounter’ variable
  + Set the LCD Cursor to (7,1)
    - Print “GO” to the screen
* If the value is equal to ‘x’
  + Follow the exact instructions above, however, once the LCD Cursor is set to, in this case (6,1) rather than (7,1), print “STOP”

**Write the following while loop**

while(Serial.available()) {

Serial.read();

}

**Give a delay of 10, and set the Bluetooth value to char ‘2’ to reset everything**

**Turret Arduino**

**What you will need:**

1. RF Sensor (NRF24L01)
2. Two servos

**Code:**

Include/Define

* Include SPI, RF24 and Servo .h files (header files)

Global Variables

**RF24**

* To utilize RF, radio configuration must take place. Create an RF24 object, defining the CSN and CE pins being used as the two argument
  + For example, pins 9 & 10 could be used for CSN and CE
* The Turret will be communicating with the Go-Kart and Ground Station, and thus a byte array in the form of a matrix will be needed to differentiate whether the RF is dealing with the Ground Station, Go-Kart, and Turret
  + The size of each must be 6, so for example: “turrt”, “gkart”, and “grnds”
* Set the role to 0

**Servos**

* Make two objects of type Servo for up/down and left/right
* You will once again deal with unsigned integers of Up/Down and Left/Right outputs

void setup() { function

**After beginning Serial, attach your servos**

* Parameters will be the pins you decide to attach your servos to

**Begin the radio**

* Set the PA Level to HIGH
* Open the reading pipe, reading in the Turret address from your byte address of the radio

**Starting listening**

void loop() { function

**Begin with an optional delay**

**If the radio is available**

* Read both the up/down and left/right outputs

**If the up/down output is greater than 0, write the output. Same goes for the left/right output**

**Software Applications Used**

1. Arduino 1.8.9

* Used to: Code all files
* Download: [Arduino 1.8.9](https://www.arduino.cc/en/main/software)

1. Atom

* Used to: Make .h files (header files)
* Download: [Atom](https://atom.io/)

1. Fritzing

* Used to: Make electrical schematics
* Download: [Fritzing](http://fritzing.org/download/)

1. Arduino Bluetooth Controller from Google Play Store

* Used to: Deploy defense mechanisms by clicking left or right on the Android Device
* Download: [Arduino Bluetooth Controller](https://play.google.com/store/apps/details?id=com.giumig.apps.bluetoothserialmonitor)