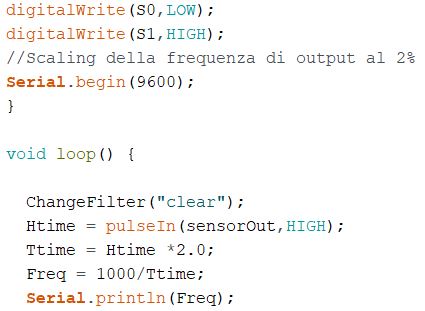
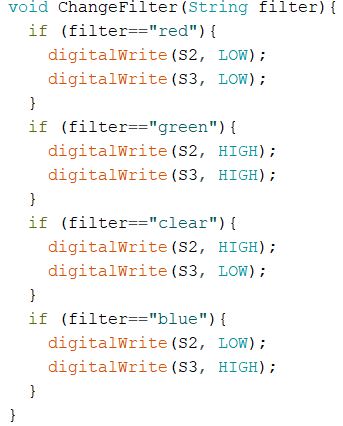
**Esercitazione 4, sensore TCS3200**

1. **Acquisizione dei dati e analisi.**

La quarta esercitazione richiedeva di settare il filtro in modalità *clear* in modo tale da verificare il funzionamento del fotodiodo al variare dell’illuminazione ambientale. Per fare ciò abbiamo elaborato uno *sketch* ausiliare. Nello sketch è presente la funzione *ChangeFilter(string*) che permette di cambiare il filtro applicato al sensore a seconda della stringa che diamo come parametro, l’implementazione è affiancata dallo *sketch*:

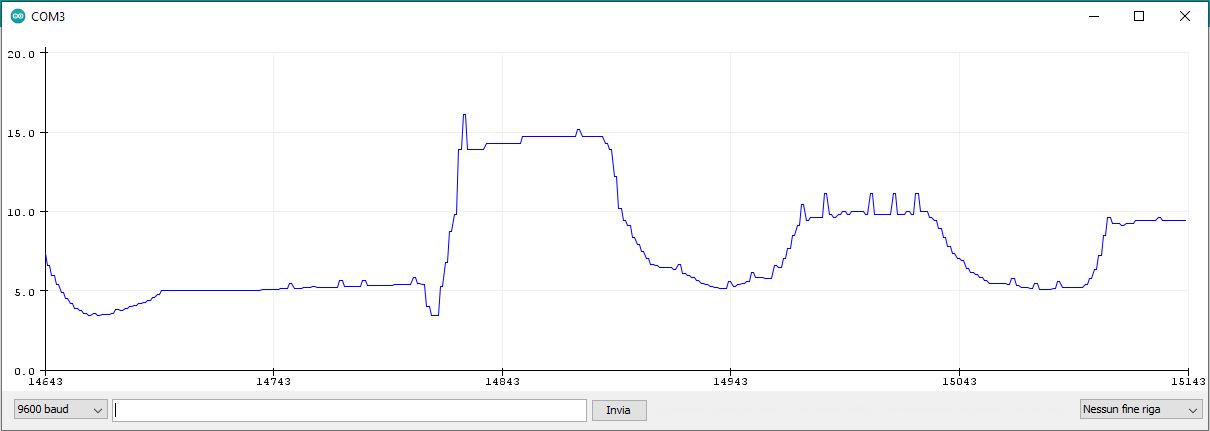


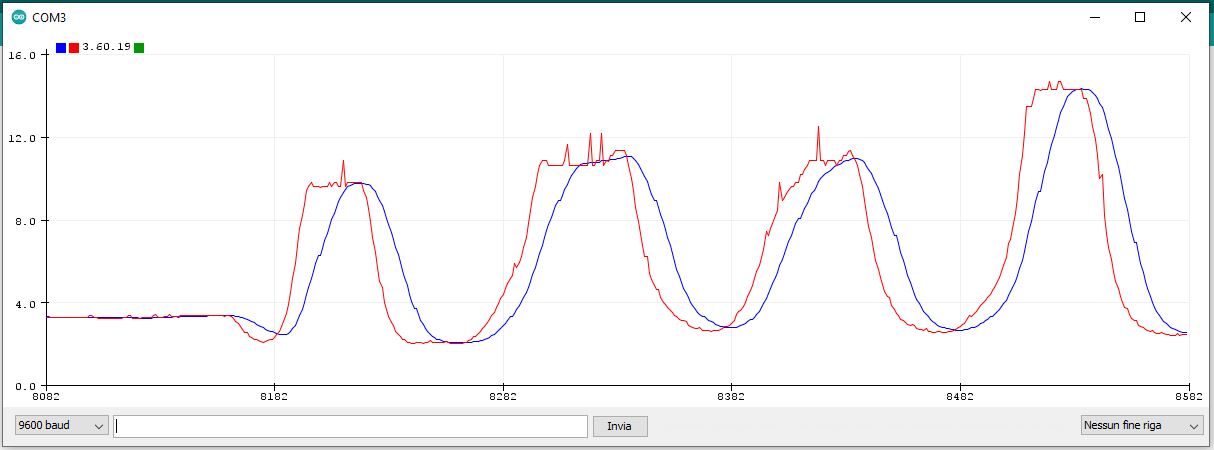
Lo scaling della frequenza è impostato al 2% per motivi che spiegherò successivamente. Il comando *ChangeFilter(“clear”)* permette di impostare il fotodiodo *clear*, quindi privo di alcun filtro. Per il calcolo della frequenza abbiamo usato la funzione *pulseIn()* che permette di acquisire il tempo in cui il segnale rimane alto durante un impulso. Dato che noi sappiamo dal datasheet di avere un *dutycycle* al 50% (“*The output is a square wave (50% duty cycle) with frequency directly proportional to light intensity.*” Pag. 1), ovvero ½ fronte alto, ½ fronte basso, quindi abbiamo ricavato la frequenza:

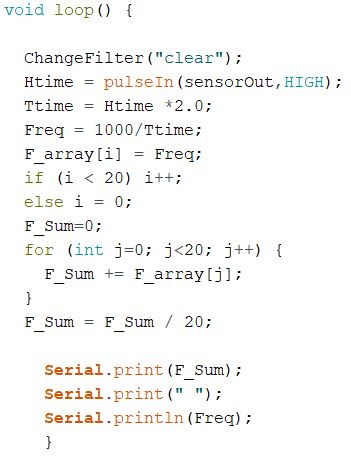
dove sensorOut, HIGH)

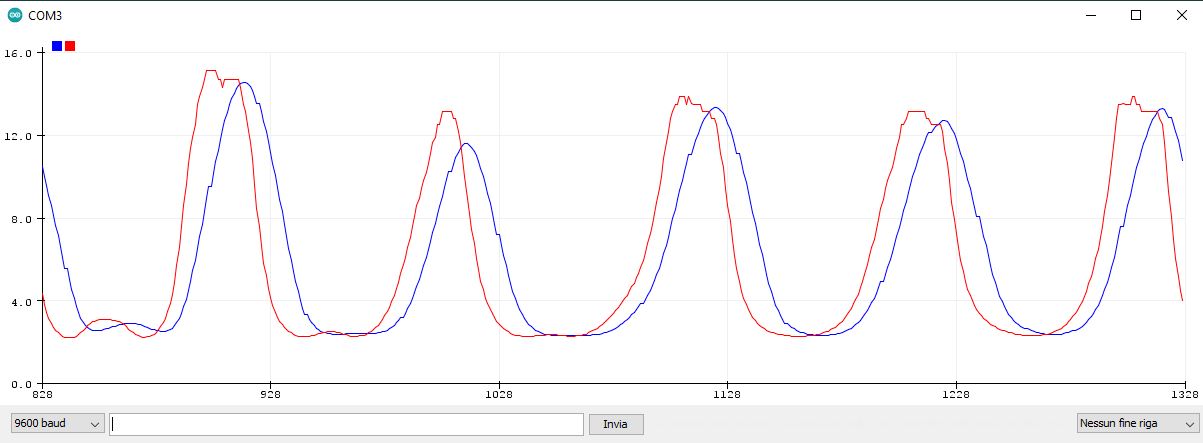
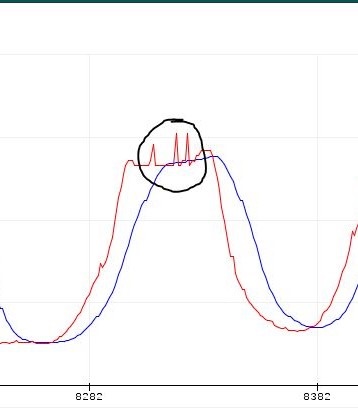
Il motivo per il quale abbiamo deciso di impostare la frequenza al 2% è perché quando abbiamo frequenze alte il nostro calcolo peggiora, in quanto *pulseIn()* ha un errore di , che noi moltiplichiamo per un fattore 2, che a basse frequenze, quindi a lunghi periodi non sono un errore significante quanto ad alte frequenze quindi a periodi corti. Per esempio, nel caso di 120 kHz avremmo un periodo di 8,3 µs, quindi di circa ¼ dell’errore di *pulseIn()*! Quindi, dato che il sensore può inviare frequenze dell’ordine di centinaia di kHz (scaling al 100%), dell’ordine delle decine di kHz (scaling al 20%) e dell’ordine di unità di kHz che ha uno scaling del 2%, abbiamo deciso di prendere questo.

Dopo aver analizzato i dati e ragionato sulla frequenza abbiamo incominciato a ragionare sul grafico e sulle eventuali interferenze che possono agire sulle frequenze, oltre all’errore del *pulseIn()*, nella pagina successiva riporto il grafico grezzo.



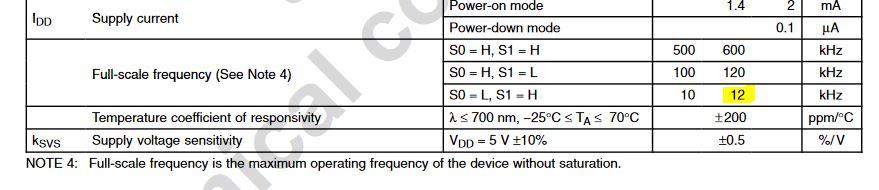
Come si nota, il grafico è gravemente alterato, la prima idea che abbiamo avuto è stato fare una media mobile usando un array di 20 elementi, dove ogni volta che il microcontrollore ripete il loop aggiunge un elemento all’array e, quanto questo è pieno, sostituisce il più “vecchio” con il più recente e ne fa la media:



Dopo che abbiamo fatto la media questi sono i risultati, ma rimane un problema, gli spikes che talvolta compaiono nel segnale grezzo. Confrontandoci con il prof. Carlini abbiamo usato la funzione noInterrupts() che permette di fermare gli interrupts che agiscono nel ciclo del programma, questi influivano la funzione pulseIn() che restituiva una frequenza più alta rispetto alla reale per alcuni punti, avendo usato questa funzione ora il segnale, a variazioni significative, non presenta più disturbi gravi :

Avendo utilizzato la media mobile, il nostro segnale ha un problema significativo, ovvero quello del ritardo, il firmware progettato non è in grado di dare risposte istantanee ma deve necessariamente aspettare tutti i 20 dati letti.

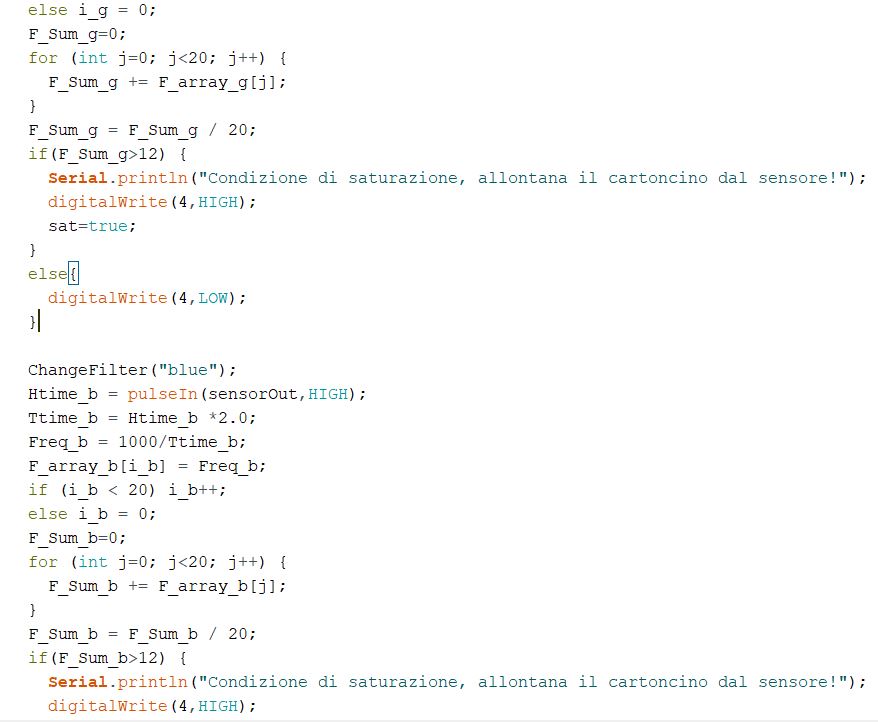
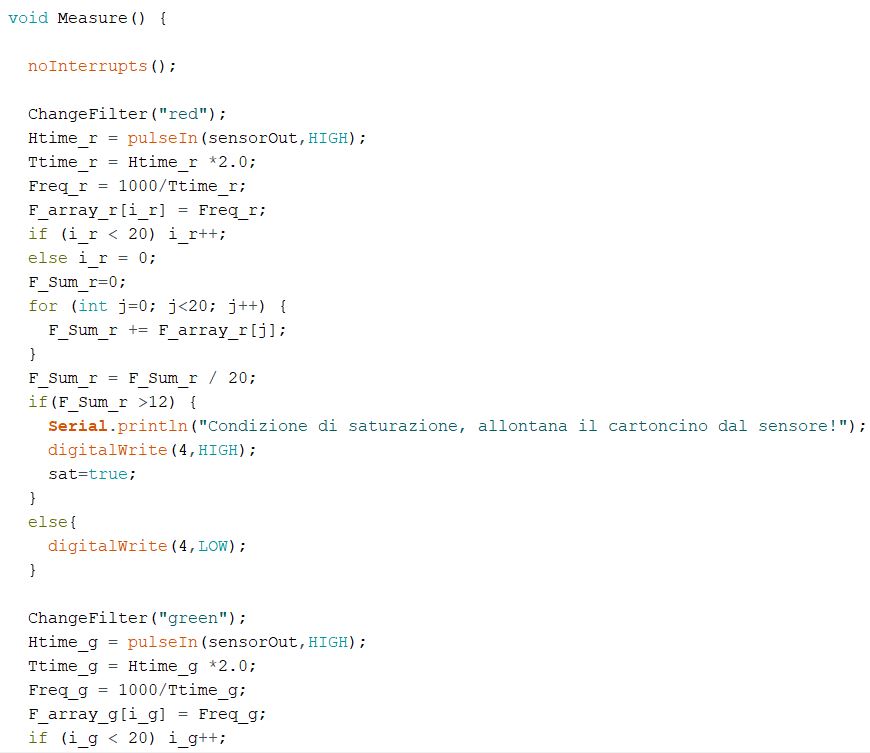
Per la gestione del valore di saturazione, leggendo il datasheet, abbiamo trovato la tabella che presentava il valore di saturazione per ciascun scaling, che nel caso del 2%, è di 12 kHz (pag. 3).



La nota 4 dice: “*Full-scale frequency is the maximum operating frequency of the device without saturation*.”, ovvero che il sensore può superare il valore di saturazione, ovvero 12 kHz, ma lavora in saturazione quindi i risultati che otteniamo non sono attendibili. Per evitare questo problema nello sketch abbiamo implementato un semplice *if* che avverte l’utente di lavorare in cui il sensore non risponde in modo coerente come nell’intervallo inferiore e non mostra la frequenza calcolata.

Per la calibrazione abbiamo aggiunto numerosi componenti a un nuovo sketch separato da quello precedente in modo da svolgere tutto il processo che necessita di diversi calcoli.

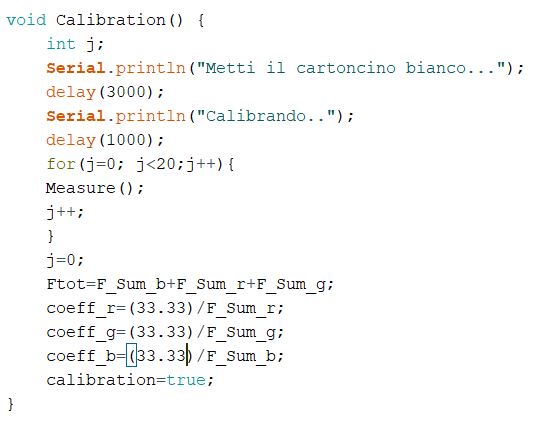
Innanzitutto per misurare, memorizzare e filtrare le tre frequenze ottenute dai tre fotodiodi filtrati abbiamo implementato la funzione Measure(), che riporto alla pagina successiva.



La funzione sembra complicata ma mette in pratica pochi concetti, il primo è quello della media mobile, tutti e tre i segnali sono mediati e poi restituiti nella relativa variabile F\_sum\_color, che è la frequenza calcolata. Il secondo concetto è quello della saturazione, il segnale F\_sum\_color non è stampato se supera i 12 kHz. Il terzo è quello del *noInterrupts(),* ovvero all’inizio della funzione tolgo gli interrupts per poi riinserirli tramite la funzione interrupts() in modo da sfruttare successivamente il monitor seriale come input e la funzione *delay(),* che sono due funzioni che lavorano grazie agli interrupts interni. In sintesi, ciò che la funzione *measure()* fa è quello di:

* Calcolare la media mobile dei valori, assegnandola a una variabile che utilizzerò successivamente.
* Controllare se il sensore opera in saturazione, effettuando un toggle sulla variabile booleana *sat* (settandola *true*). La funzione accende anche il led rosso, che poi spegnerà dopo che il sensore non opera più oltre i 12 kHz (se no potrebbe addirittura influire sulla frequenza dei fotodiodi.

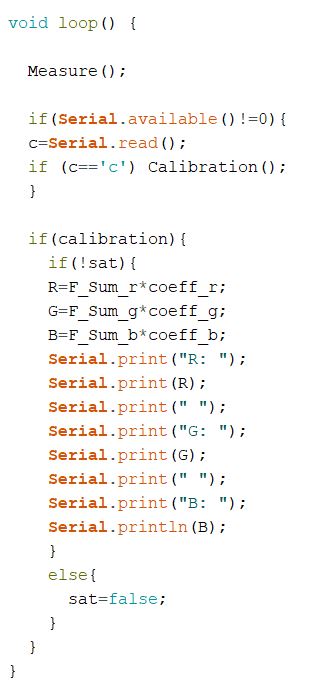
Per la calibrazione abbiamo implementato la funzione *Calibration(),* la quale non restituisce anch’essa alcun parametro ma svolge il punto cruciale della nostra esercitazione.

La funzione permette all’utente di porre il cartoncino ad una certa distanza dal sensore, dopodiché svolge 20 volte la funzione *Measure(),* in modo da calibrare il meglio possibile il “colore” bianco. Dopodiché, noi vogliamo che un certo coefficiente che sarà diverso per ogni fotodiodo restituisca 1/3 come dato:

(oppure 33%)

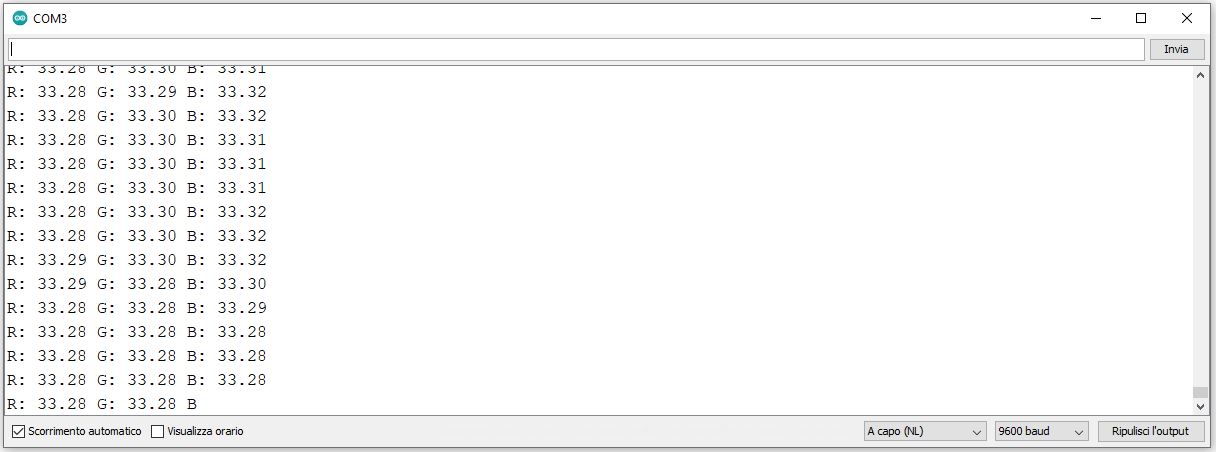
Questo permette di avere dei risultati che considerano le variazioni importanti di frequenza dei diversi fotodiodi.

Queste due funzioni vengono richiamate nel loop in modo da avere sempre delle misurazioni tramite i coefficienti calcolati. Un fattore importante è la variabile booleana *true* che permetterà di stampare i dati sul monitor seriale. Nella pagina seguente mostro il loop dello *sketch*.

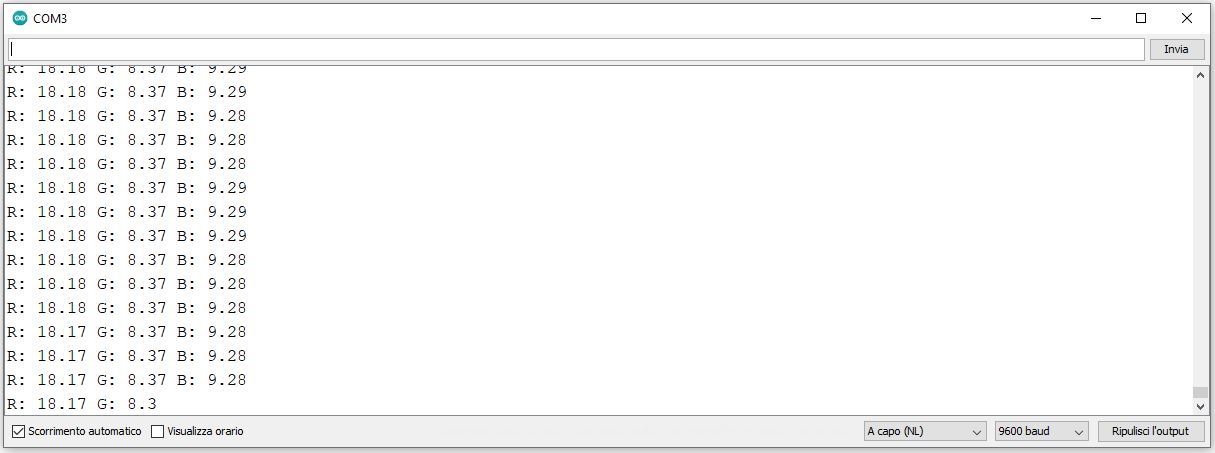
Nella funzione *loop()* l’arduino continua a effettuare la funzione *Measure()* senza però stampare alcun dato. Dopo che diamo il segnale via monitor seriale alla scheda, questa effettua la procedura di calibrazione richiamando la funzione *Calibration()*. Dopodiché, stampo tre valori: R, G, B. Questi tre valori sono le frequenze moltiplicate per il coefficiente moltiplicativo.

Di seguito mostro i risultati che abbiamo ottenuto ponendo diversi colori dopo la calibrazione del bianco.

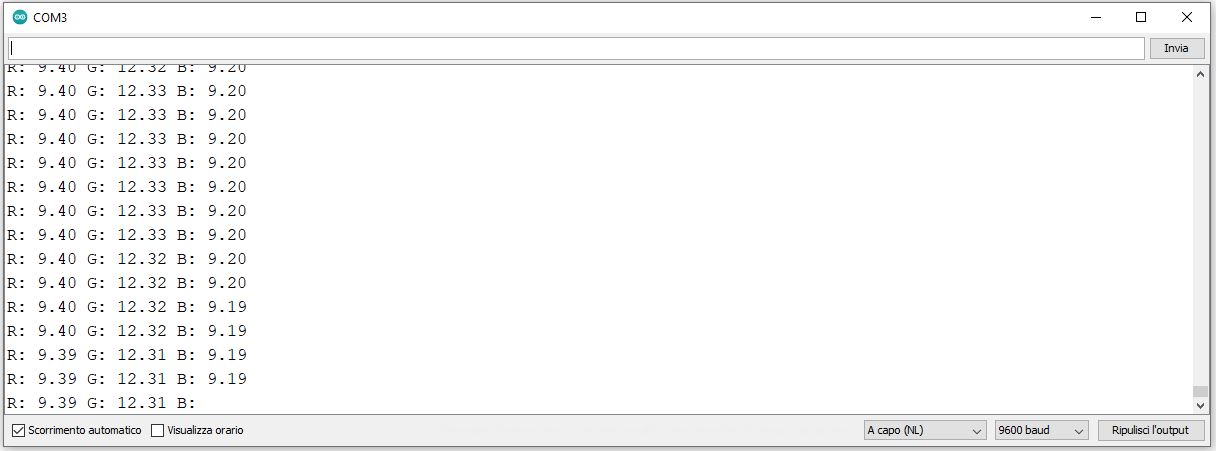
Colore bianco.



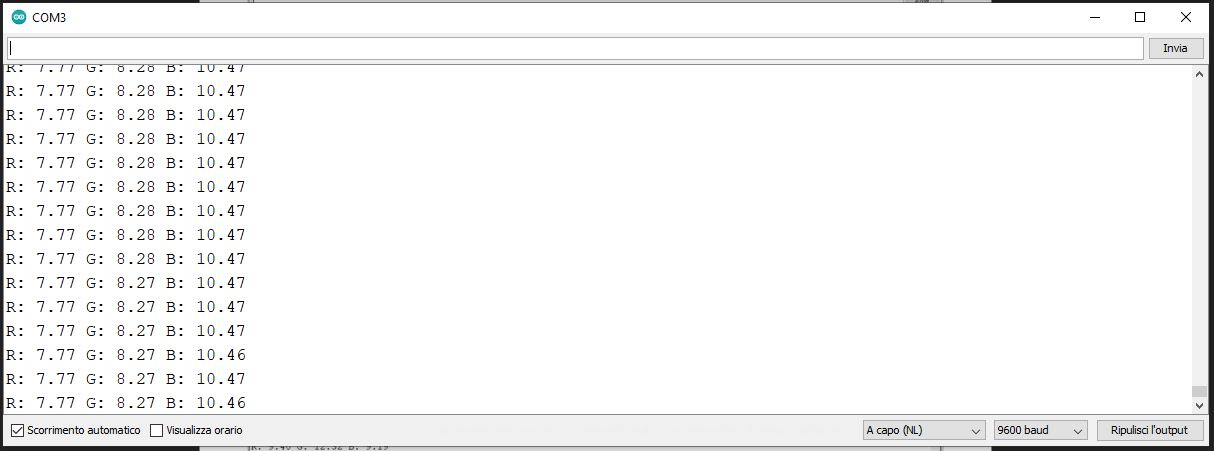
Colore rosso.



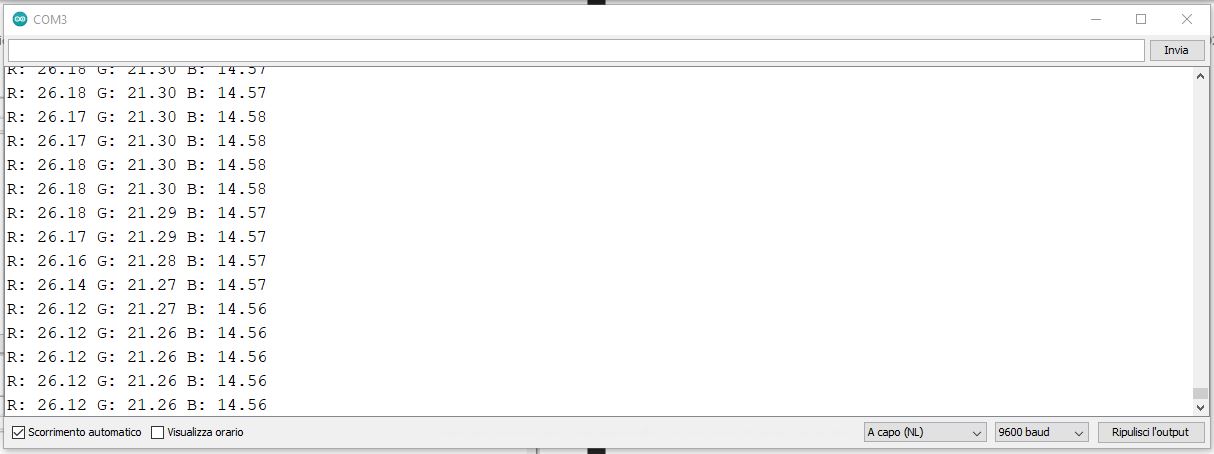
Colore verde.



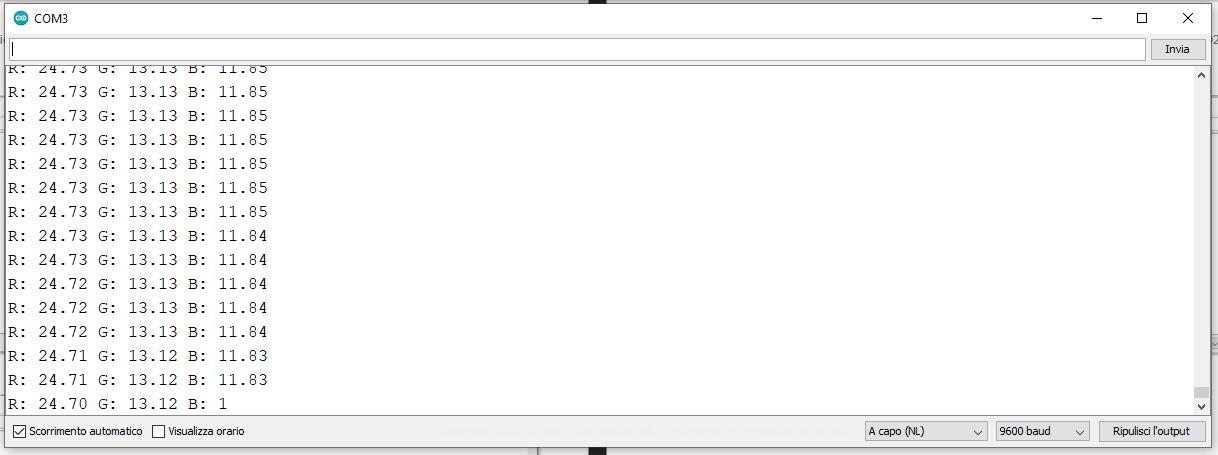
Colore Blu.

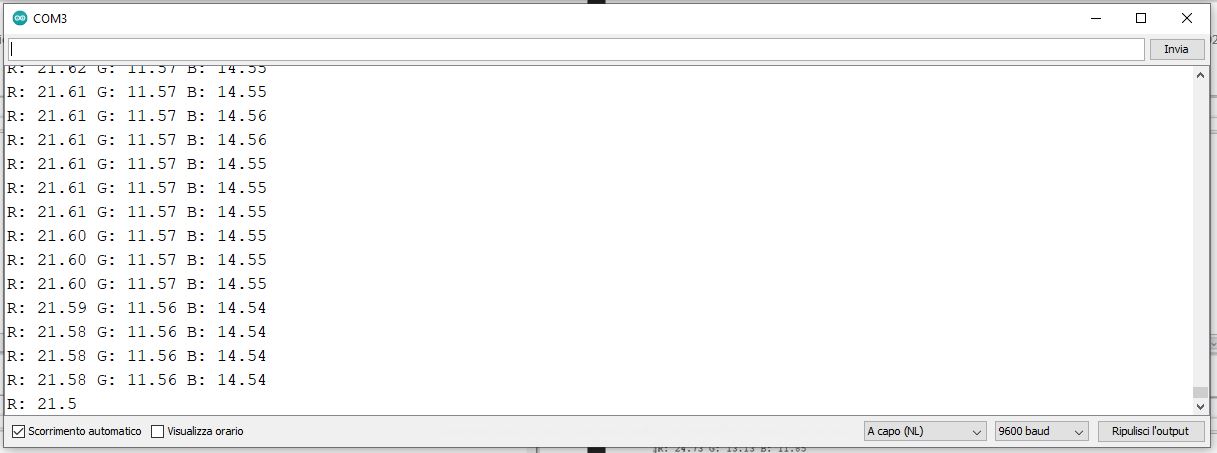


Colore giallo.



Colore Arancione.



Colore Rosa.

Per completezza ho aggiunto altri tre colori diversi dal rosso, verde e blu. I risultati sono piuttosto soddisfacenti a parte il rosa, che avendo una composizione RGB (scala 0-255 Pink: (255,192,203)) complessa, non è stato riconosciuto.

1. **Conclusioni e difficoltà incontrate.**

In conclusione, siamo soddisfatti del nostro lavoro, il sensore capisce il rosso, il verde e il blu senza troppi problemi, anche alcuni colori diversi possono essere riconosciuti, ma non perfettamente quanto i tre principali. Le maggiori difficoltà che abbiamo incontrato sono legate alla sensibilità del sensore, in quanto la frequenza cambia notevolmente, si nota anche dal tremolio della mano. Quindi abbiamo escogitato delle pseudo “costruzioni” per riuscire a tenere sempre alla stessa distanza i cartoncini, evitando piccole oscillazioni. In ogni caso, il sensore è in grado di rilevare, anche nell’eventualità di piccoli tremolii, il colore.

Per completezza riporto nelle prossime pagine l’intero sketch.

