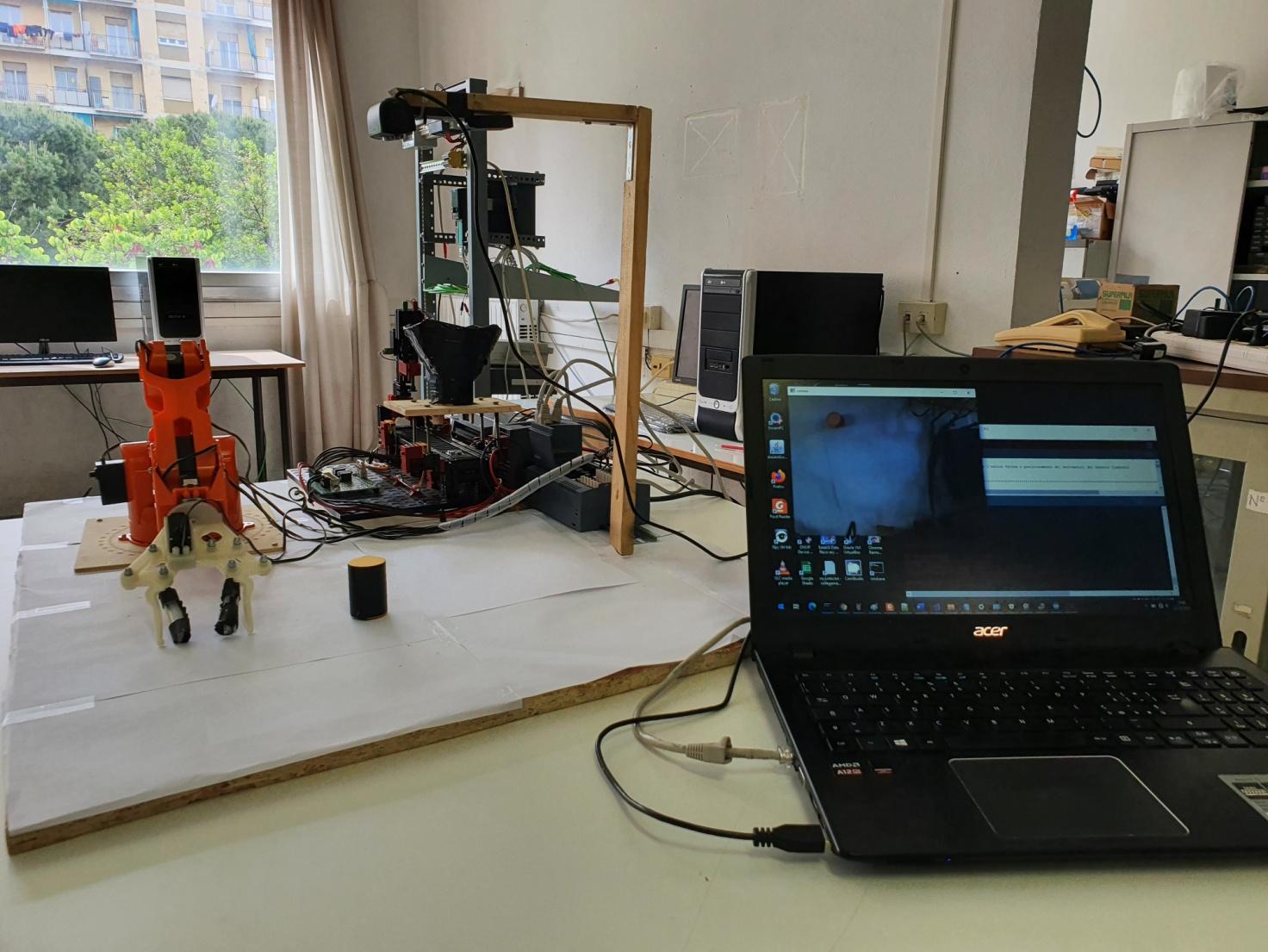
** **

**“Sistema di controllo Industria 4.0”**

**12° Edizione delle “Olimpiadi Automazione Siemens”**

****

**Gruppo di lavoro:** Bozzano Stefano, Cavallino Edoardo, Rampa Jacopo, Ravaschio Irene, Simcich Thomas

**Tutor:** Fischetti Pietro, Novelli Claudio

**INTRODUZIONE**

Gli studenti Bozzano Stefano, Cavallino Edoardo, Rampa Jacopo, Ravaschio Irene e Simcich Thomas della classe 5a Elettronica ed Elettrotecnica articolazione Automazione sez. B dell’Istituto di Istruzione Superiore Italo Calvino, hanno elaborato il seguente progetto le cui specifiche sono state concordate con l’ing. Pier Paolo Casella della TAU LOGIC via XX Settembre 16121 Genova.

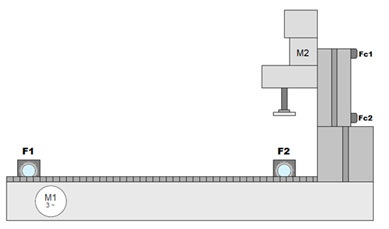
Lo scopo di questo progetto era quello di automatizzare un impianto – posizionato su un piano di lavoro in legno – costituito da:

* una telecamera TedGem USB – con una risoluzione di 640×480 – collegata ad un PC. La telecamera è montata su una base in legno – sopra al piano di lavoro – ad un’altezza di circa 42[cm] – rispetto al piano in legno – in modo tale da avere una buona visione, occupando più campo possibile e non entrando in conflitto con il braccio robotico.
* un braccio robotico costituito da:
  + una base ruotante per orientarsi orizzontalmente;
  + tre giunti che ne permettono i movimenti verticali;
  + una pinza che può essere aperta o chiusa.

I movimenti di ogni parte – e giunto – del robot è gestita da un servomotore, con una precisione di 1 grado.

La gestione del braccio è svolta mediante la *shield* per motori di Arduino, installata su un IoT2020 collegato via seriale – RS232 – al PC.

* un nastro trasportatore con 1 isola di lavoro (punzonatrice) a 24V, FISCHERTECHNIK EDUCATION. La punzonatrice è controllata dal PLC Siemens SIMATIC S7-1214C DC/DC/DC – collocato in prossimità del piano di lavoro – e collegato, insieme all’IOT2020, una rete Profinet mediante lo Switch Module Siemens;



Il processo risulta costituito dai seguenti passi:

1. A processo avviato, se la fotocamera individua un cilindro sul piano di lavoro ne rileva le sue coordinate. Attraverso di esse – sfruttando la cinematica inversa – vengono calcolati gli angoli da inviare ai servomotori del braccio robotico affinché questo si orienti con la pinza aperta nell’esatta posizione nella quale si trova il cilindro.

L’elaborazione delle coordinate del cilindro sul piano di lavoro è eseguita dal modulo software scritto in linguaggio Python ed inviate all’IoT2020.

1. Il braccio robotico preleva il cilindro dal piano di lavoro e lo deposita nel collettore, che consente di collocare il cilindro all’inizio del nastro trasportatore.
2. Il nastro, rilevato il cilindro, si avvia. Quando questo è nella zona della punzonatrice il nastro si arresta.
3. Il pezzo viene punzonato una volta – viene incrementato il conteggio dei pezzi lavorati – e ritorna alla posizione iniziale.
4. Il pezzo viene prelevato manualmente e il nastro è pronto a ricevere altri cilindri.

Il sistema prevede l’utilizzo di quattro ambienti di sviluppo: TIA Portal, Python IDE, Arduino IDE e Note red.

**TIA Portal** → KOP

Utilizzato per la programmazione del PLC che gestisce la punzonatura.

**Python IDE** → Python 2.7.6.1

Si occupa dell’acquisizione della posizione del cilindro e dei calcoli di cinematica inversa.

**Arduino IDE** → Wiring, derivato dal C++

Viene utilizzato per la gestione dei motori del braccio robotico.

**Node red** → JavaScript

Utilizzato per la gestione da remoto della punzonatura.

**Componenti:**

* **PLC:** SIEMENSSIMATIC S7-1214C DC/DC/DC

Ingressi analogici 2

Ingressi digitali 14

Tipo di ingresso analogico 0 ... 10 V CC

Tipo di entrata digitale 24 V CC

Tipo di uscita digitale Transistore

Numero di uscite transistor 10

Numero di contatori ad alta velocità 6

Input/output max. 284 DI/DQ, 51 AI/AQ

Tensione di esercizio, min. 20.4 VDC

Tensione di esercizio, max. 28.8 VDC

Corrente di carico 500 mA

Tasso di baud max. 100 Mbps

Linguaggio di programmazione utilizzato Function Block Diagram (FBD) - Diagramma a blocchi funzionali

Ladder Diagram (LD) -Diagramma a contatto

Structural Text (Testo strutturale)

Memoria dati 100 kB

Buffer di dati 480 ore

Tipo di interfaccia di comunicazione PROFINET

Temporizzatore funzionante in tempo reale Sì

Tipo di montaggio Montaggio su guida DIN

Estensibilità Sì

Backup batteria Sì

Grado di protezione IP20

Capacità programma 4 MB

* **SWITCH:** SIEMENS SCALANCE XB005

Ingressi: 1

Uscite: 4

tensione di alimentazione nominale: 24VDC

una corrente nominale: 0,07 A.

Velocità di trasmissione: dati fino a 1 Gbit/s.

Temperatura d’ esercizio: -10 °C - 60 °C

Velocità di trasferimento dati: 10/100 Mbit/s o 1 Gbit/s  
Tipo di ingresso e tipo di uscita: RJ45

Distanze fino a 26km Auto-crossover/auto-negoziazione

montaggio su guida DIN dotato di un metodo di terminazione a vite.

* **IOT2020**

• Compatibile con la maggior parte degli sketch Arduino  
• Compatibile con programmi compilati ed eseguiti sotto Yocto Linux  
• Intel Quark x 1000 (Galileo)  
• Possibilità di integrare hardware e sensori tramite Ethernet o seriale tramite adattatore USB  
• Molti protocolli disponibili e aperti come Modbus, Profinet, REST e MQTT  
• Compatibile con gli schermi Arduino e i linguaggi di programmazione, inclusi i linguaggi di alto livello come Java, C++ e Python  
• Espandibile tramite una porta mPCIe integrata  
• Node-Red, MQTT e SQLITE3 possono essere utilizzati per collegare e controllare periferiche locali e remote, come sensori su un dispositivo  
• 512 MB di RAM  
• Alloggiamento per scheda microSD incorporato  
• 1 interfaccia Ethernet  
• Nell’ IOT è presente un LED  
• Un alto grado di robustezza  
• È possibile un funzionamento integrato senza manutenzione

* **BRACCIO ROBOTICO:** TINKERKIT

21x componenti in materiale plastico

4x servomotori SR 431

2x servomotori SR 311

1x shield Arduino-compatibile

1x alimentatore 5V – 5A

1x cavo a spirale schermato

63x viti

16x guarnizioni piatte

7x dadi esagonali

2x molle

1x chiave a doppio anello esagonale

1x cacciavite Phillips

* **SPECIFICHE:**

**Braccio Robotico Tinkerkit (assemblato):**

* + Range di distanza operativa (max): 80cm
  + Capacità di carico (distanza operativa 32 cm): 150g
  + Capacità di carico (in configurazione minima): 400g
  + Altezza (max): 52cm
  + Larghezza della base: 14cm
  + Ampiezza pinza: 90mm
  + Lunghezza del cavo: 40cm
  + Peso: 792g

**Shield (comp. Arduino) per controllo servomotori:**

* + Tensione operativa: 5V
  + Consumo: 20mW
  + Corrente (max): 1.1A (M1 ~ M4) - 750mA (M5 ~ M6)

**Servomotori SpringRC SR431/SpringRC SR311:**

* + Riduzione: in metallo
  + Connettore : Tipo J (Futaba)
  + Segnale di controllo: Analogico PWM
  + Cuscinetti a sfera: 2
  + Angolo di rotazione: 180°
  + Coppia:
    - SR431: 12.2 kg-cm (@4,8V) - 14.5 kg-cm (@ 6.0V)
    - SR311: 3.1 kg-cm (@ 4.8V) - 3.8 kg-cm (@ 6.0V)
  + Velocità:
    - SR431: 0.20 sec/60° (@4.8V) - 0.18 sec/60° (@6.0V)
    - SR311: 0.14 sec/60° (@4.8V) - 0.12 sec/60° (@6.0V)
  + Dimensioni e peso:
    - SR431: 42 × 20.5 × 39.5 mm - 62g
    - SR311: 31.3×16.5×28.6 mm - 27g

È necessario utilizzare la shield apposita per l’utilizzo del braccio.

* **WEBCAM:** TEDGEM

Utilizzata con risoluzione 640 x 480

* **PUNZONATORE**

1 x nastro trasportatore bidirezionale (motore DC 24V)

1 x punzonatore (motore DC 24V)

2 x barriere fotoelettriche (NC)

2x contatti di finecorsa (NO)

**NODE RED**

Il processo di punzonatura può essere fermato attraverso il pulsante di “Stop” e ripreso attraverso il pulsante “Resume”. Infine il pulsante “Reset” serve per azzerare il conteggio dei pezzi lavorati.

All’interno del progetto l’interfaccia utente e il controllo da remoto sono svolti grazie all’utilizzo di Node-RED.

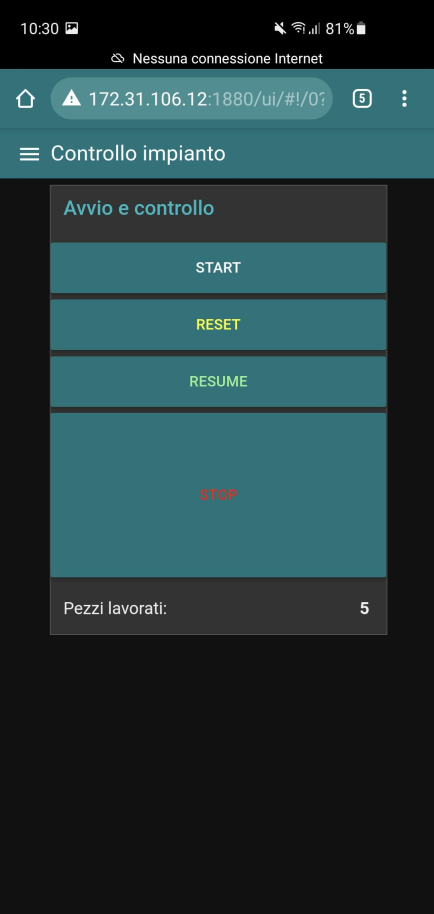
Grazie alla piattaforma possiamo visualizzare le dashboard create con Node-RED direttamente sul telefono, sul quale possiamo anche gestire il progetto, premendo i pulsanti virtuali, oppure visualizzare i grafici.

La dashboard è costituita da tab, nel nostro caso più precisamente da n. 2 tab:

1. Controllo impianto
2. Punzonatrice

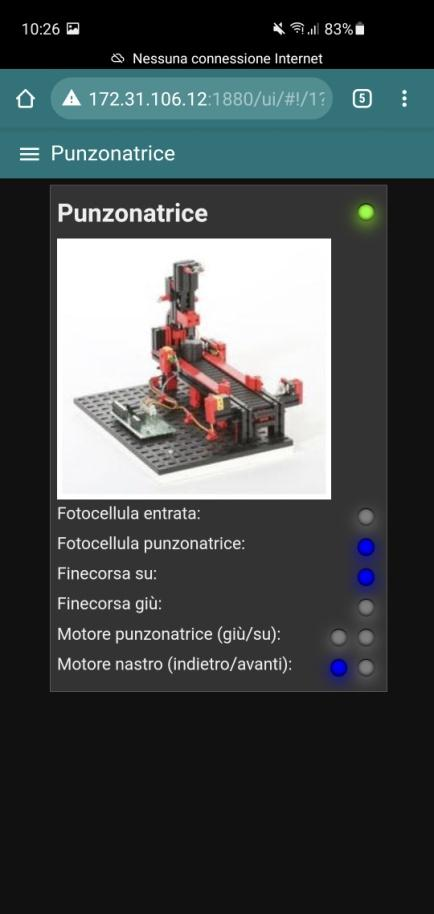
La prima pagina della dashboard – Controllo impianto – è costituita da:

* n.4 pulsanti:
  + “Start”, avvia il processo;
  + “Reset”, azzera il numero di pezzi lavorati;
  + “Resume”, riprende il processo se viene interrotto;
  + “Stop”, ferma il processo;
* n.1 nodo di testo “Pezzi lavorati”, con il corrispondente valore numerico.

******

La seconda pagina – Punzonatrice– è costituita da:

* n.1 immagine della punzonatrice;
* n.9 led con relativa sensore / attuatore.

******

I led rappresentano gli stati delle fotocellule, dei finecorsa e dei motori. I led assumono due possibili colori: grigio o blu.

Grigio rappresenta:

* nelle fotocellule, assenza del pezzo;
* nei finecorsa, finecorsa non premuto;
* nei motori, motore spento.

Il blu, invece, rappresenta:

* nelle fotocellule, pezzo rilevato;
* nei finecorsa, finecorsa premuto;
* nei motori, motore acceso.

Inoltre è presente un led di fianco a “Punzonatrice”, collegato al pulsante di “Stop”:

* se “Stop” è attivo, il led è rosso, la punzonatrice è bloccata;
* se “Stop” non è attivo, il led è verde, la punzonatrice è attiva.

I led ricevono gli stati dal PLC attraverso il blocco “S7 IN”.

I blocchi settano i valori come variabili globali.

I blocchi sono collegati ciascuno a un blocco “function”, questo blocco associa alla variabile un led e al suo stato un colore (secondo le condizioni precedentemente presentate):

* Fotocellula entrata, “H\_L1”,
* Fotocellula punzonatrice, “H\_L2”;
* Finecorsa inizio, “H\_L3”;
* Finecorsa fine, “H\_L4”;
* Motore punzonatrice:
  + giù, “HL\_5\_down”;
  + su, “HL\_5\_up”;
* Motore nastro:
  + avanti, “HL\_6\_back”;
  + indietro, “HL\_5\_forward”;

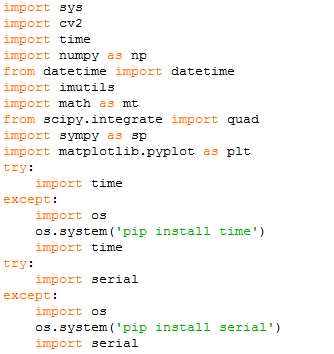
**PYTHON**

Il programma di python si occupa della acquisizione delle coordinate dalla fotocamera, le elabora e calcola, attraverso la cinematica inversa, gli angoli di ogni giunto del braccio robotico per raggiungere il pezzo con l’end effector.

Possiamo suddividere il programma in python in più parti:

* acquisizione coordinate fotocamera
* risoluzione problema della prospettiva per l’altezza cilindro
* risoluzione problema grandangolo
* cinematica inversa
* comunicazione con arduino

**LIBRERIE UTILIZZATE:**

**Sys:** modulo che fornisce l'accesso ad alcune variabili usate o mantenute dall'interprete, e a funzioni che interagiscono fortemente con l'interprete stesso

**Cv2:** libreria per la computer vision.

**Numpy:** libreria per il calcolo numerico

**Datetime:** librerie che include le funzioni e le classi per analizzare, formattare e compiere operazioni aritmetiche sulle date

**Imutils:** Una serie di funzioni utili per semplificare le funzioni di elaborazione delle immagini di base

**Math:** libreria che contiene le funzioni matematiche

**Scipy:** libreria per il calcolo scientifico

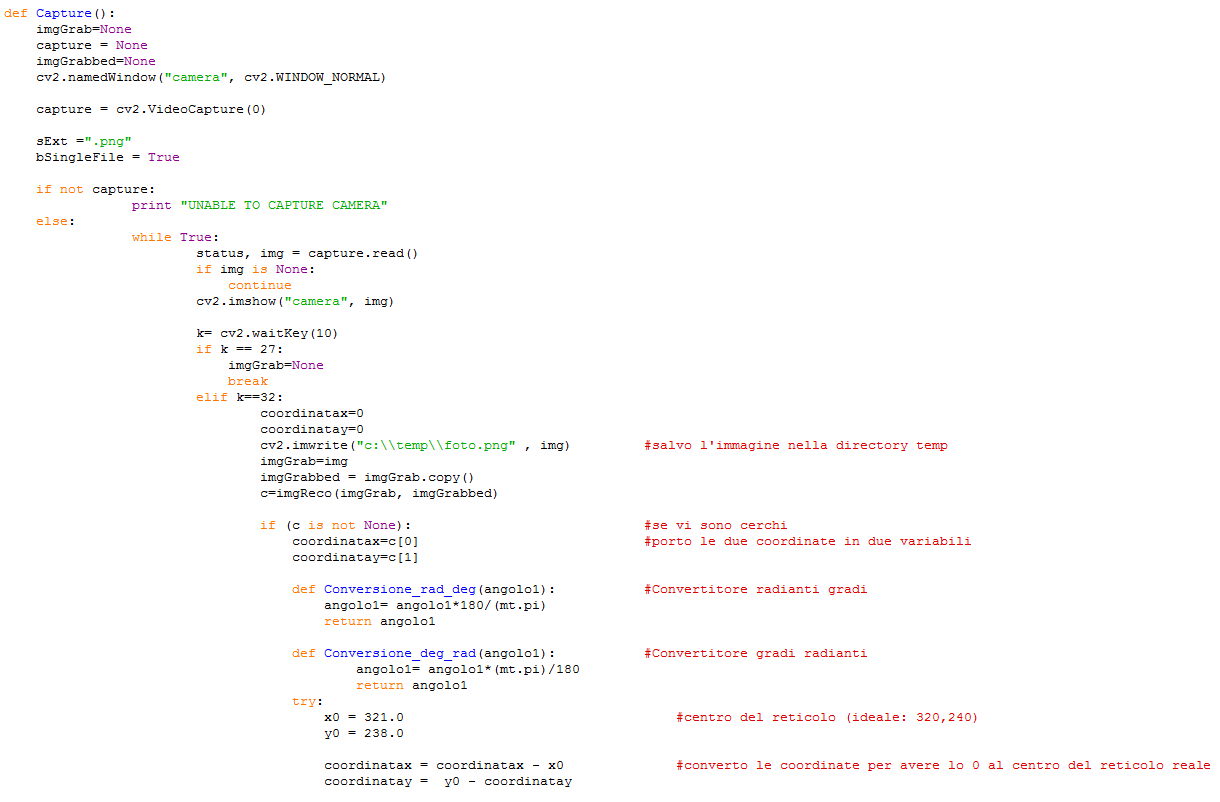
**Simpy:** Libreria per il calcolo simbolico

**Matplotlib:**  libreria grafica

**Time:** libreria che fornisce le funzioni per manipolare date e tempo.

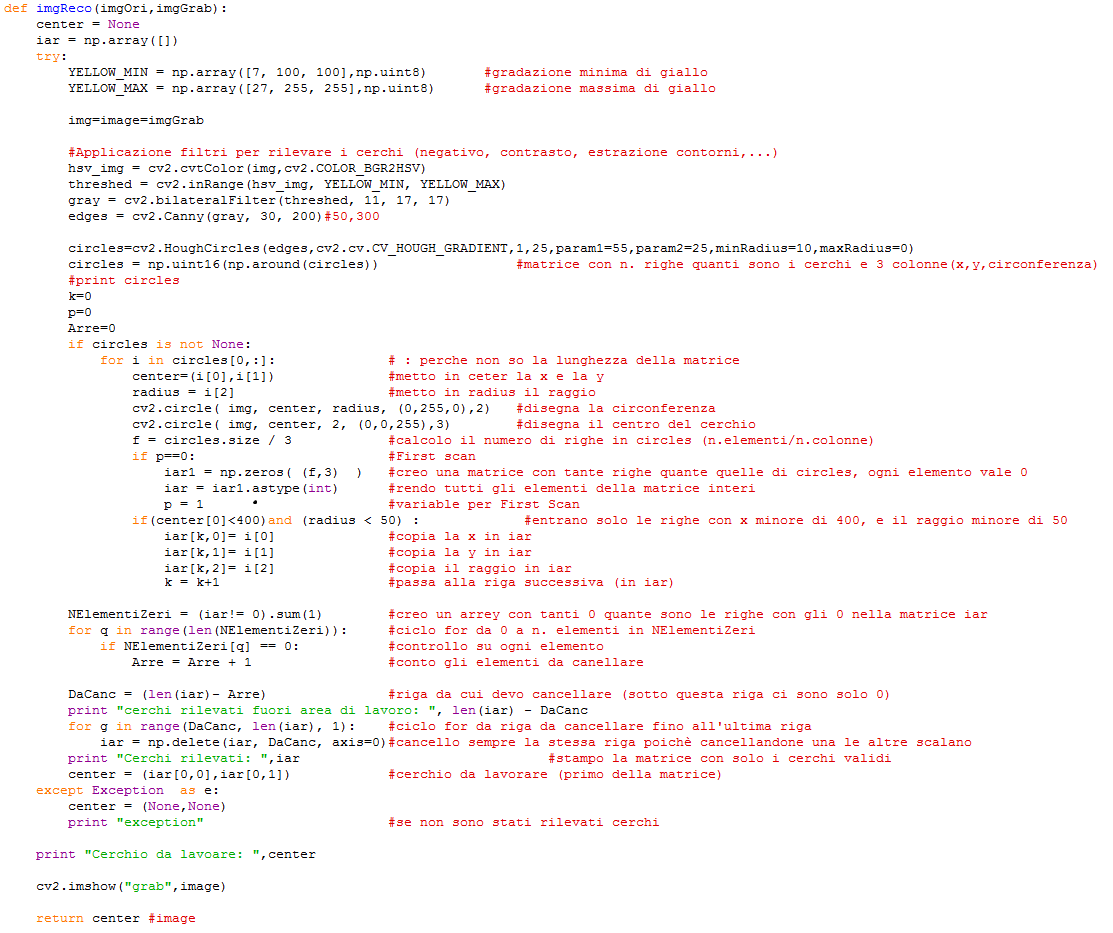
**Serial:** libreria per la comunicazione seriale

**Acquisizione coordinate fotocamera**

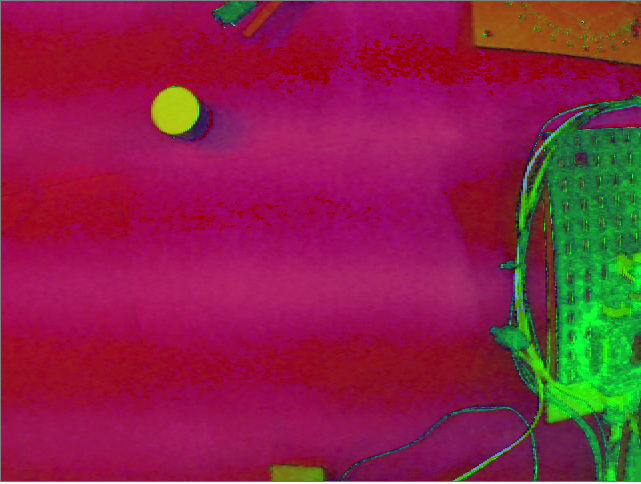
****

Questa prima parte di programma si occupa dell’inizializzazione della fotocamera e della visualizzazione della foto scattata.

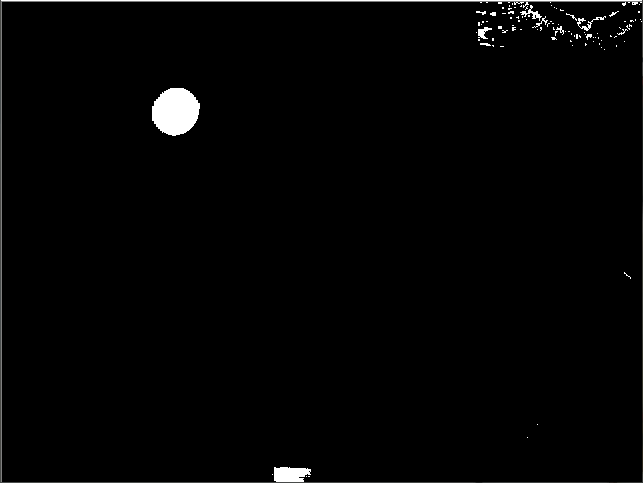
La parte di programma che si occupa realmente dell’acquisizione è sintetizzata nella funzione “ImgReco()”:



Siccome si è scelto di utilizzare dei cilindri di colore giallo all’inizio della funzione vengono settate le gradazioni di gialli accette. Per individuare il cilindro vengono applicati diversi filtri:

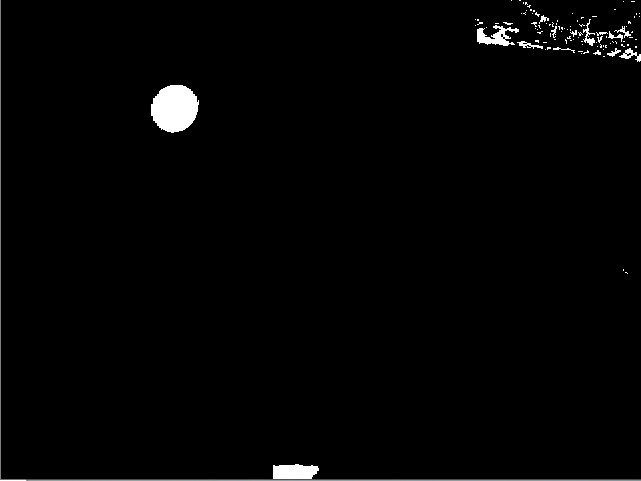
**HSV:**

Il colore viene convertito in hsv (hue, saturation, value)(tonalità, saturazione, valore). In hsv è più facile rappresentare un colore che in uno spazio colore bgr. Ciò ci facilita l’individuazione di un oggetto colorato (giallo nel nostro caso).

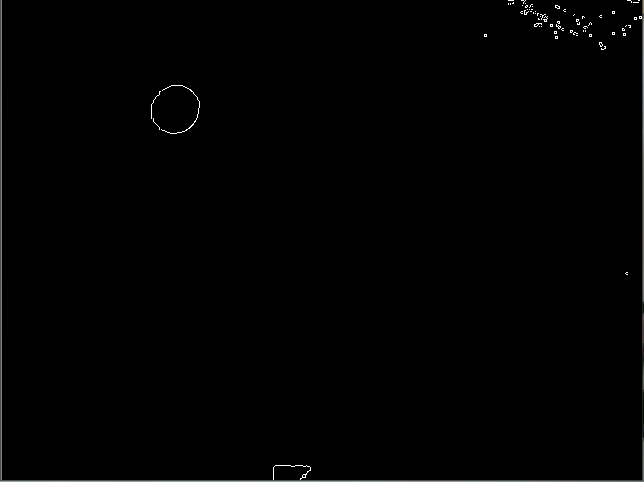


**InRange:**

Viene mantenuto solo il colore desiderato, quindi, nel nostro caso, solo le parti gialle.

**BilateralFilter:**

Si ottiene la sfocatura dell'immagine, utile per rimuovere il rumore.

**Canny:**

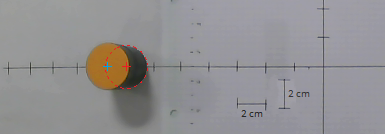
Estrae i contorni di tutte le figure estratte in precedenza.

L’immagine ottenuta può quindi essere elaborata dalla funzione **cv2.HoughCircles ()** per l’individuazione dei cerchi.

All'interno della matrice “circles” (di grandezza variabile) vengono inserite le coordinate ed il raggio di tutti i cerchi individuati (all’interno di una matrice). Successivamente viene eseguita una selezione per mantenere solo i cerchi con un raggio inferiore di 50 px e con la coordinata “x” minore di 400 px. Il controllo sulle ascisse viene eseguito poiché la webcam inquadra anche parte del punzonatore che potrebbe generare rilevazioni indesiderate.

La funzione “ImgReco” restituisce quindi le coordinate “x” e “y” in pixel del primo cerchio individuato.

* Problemi dovuti all’utilizzo della webcam:
* **PROBLEMA DELLA PROSPETTIVA PER L’ALTEZZA DEL CILINDRO**



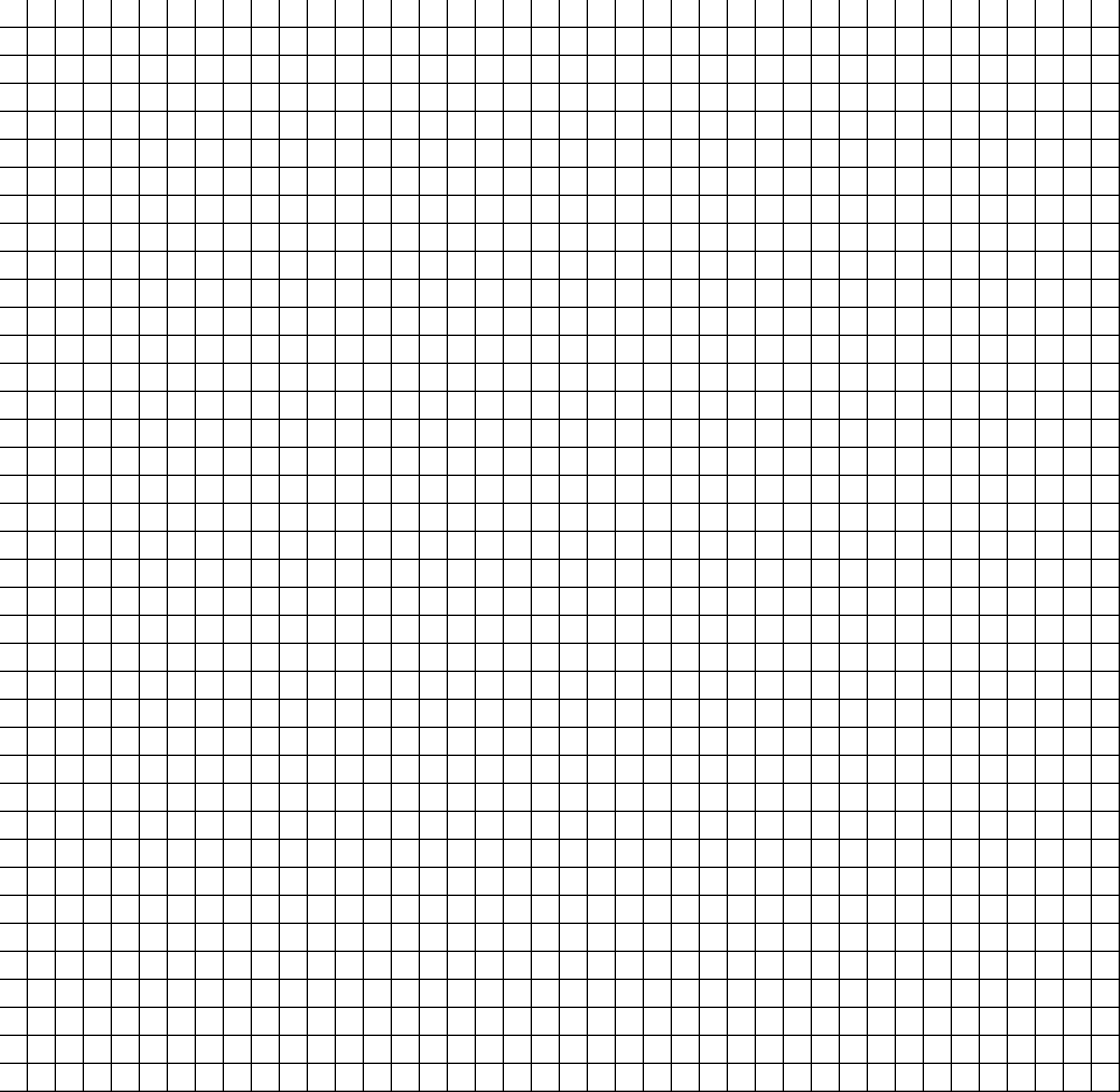
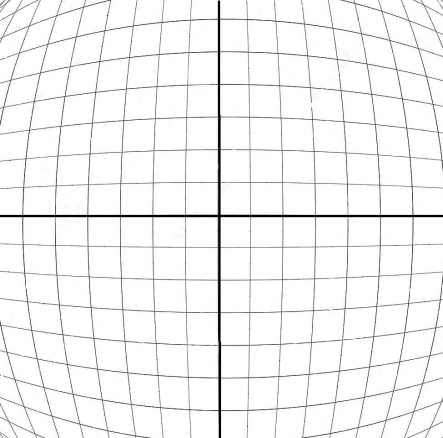
La parte di programma relativa alla telecamera individua un cerchio di colore giallo e ne restituisce le coordinate.

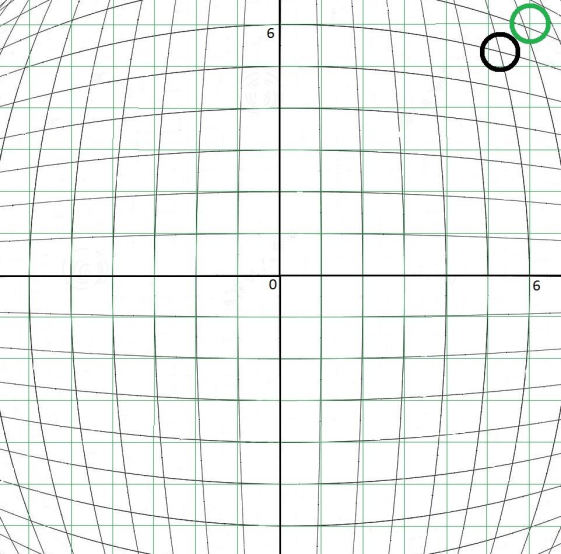
Se il cilindro non è perfettamente sottostante alla telecamera vi sarà un errore dovuto all’altezza del cilindro. Ciò che ci interessa è la posizione della base del cilindro. Come si vede in foto la base (croce rossa), si trova nella coordinata x: -14, mentre la telecamera restituirà le coordinate del cerchio giallo (croce blu) (-15???).

E’ quindi necessario compensare tale errore.

* **PROBLEMA GRANDANGOLO**

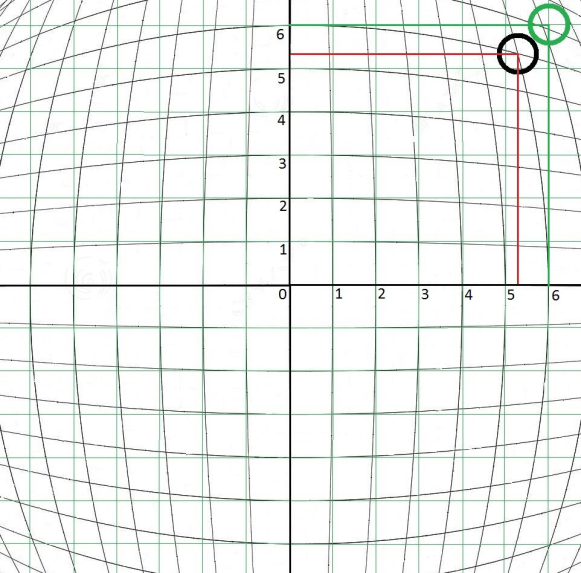
La webcam utilizzata presenta un “grandangolo”, ossia una lente che permette un maggiore angolo di ripresa, ciò comporta però una prospettiva alterata:

Se mettessimo un reticolo, come quello in figura 1, al di sotto della webcam, la lente altererebbe l’immagine distorcendo il reticolo, come in figura 2



Questa distorsione comporta un’alterazione della posizione del cilindro.

In figura vediamo in verde il reticolo “reale” con il cilindro nella posizione “reale”, in nero ciò che vede la fotocamera.

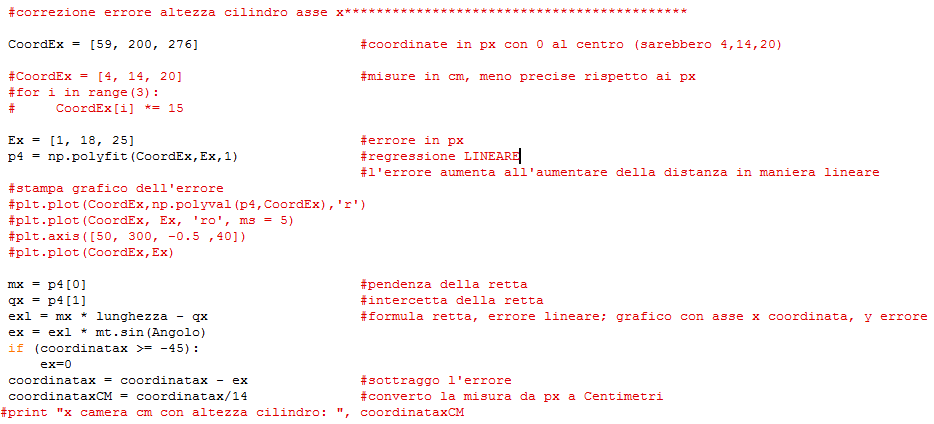


La reale posizione del cilindro è (6;6) ma la fotocamera lo individua nella posizione (5,2 ; 5,2)

Come ho risolto tali problemi:

**Problema delle prospettiva per l’altezza del cilindro**

****

****

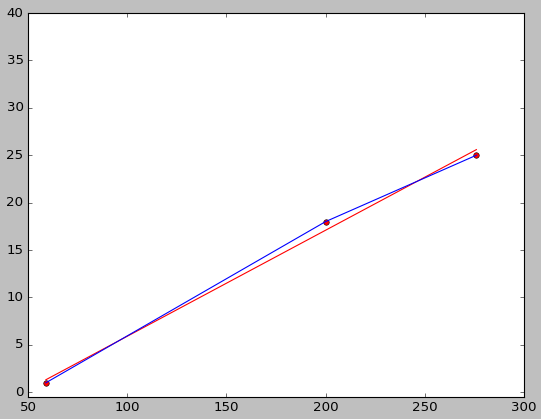
**Obbiettivo:**

Ottenere un programma che, dategli in ingresso le coordinate uscenti dal programma della telecamera, restituisca le coordinate veritiere del cilindro.

Inserendo sul grafico alcuni punti aventi come ascisse la coordinata reale e come ordinata il corrispettivo errore si nota come quest’ultimo aumenta in maniera lineare all’aumentare della distanza dal centro.

La funzione che meglio rappresenta questo andamento è la retta. Ovviamente vi è un minimo errore dovuto alla risoluzione della telecamera e alla sua posizione (non centrata perfettamente) per cui i punti non risulteranno allineati perfettamente, per questo si utilizza la **regressione lineare**.

La regressione è quella tecnica statistica utilizzata per studiare le relazioni che intercorrono tra due o più caratteri (variabili) statistici. La libreria numpy fornisce una funzione apposita che restituisce il coefficiente di ogni x, che per la retta corrispondono alla pendenza e l'intercetta.



In rosso vi è la retta ottenuta attraverso la regressione lineare.

**Otteniamo quindi la formula della retta: dato il valore “x” corrispondente alla coordinata, la funzione della retta restituisce “y”, corrispondente all’errore.**

Questo procedimento lo si esegue sia per l’asse delle x che per l’asse delle y.

E’ necessario prendere in considerazione il caso in cui il cilindro non si trovi lungo gli assi, per fare ciò si utilizza la trigonometria che ci permette di trovare l’errore prima lungo le x e successivamente lungo le y.

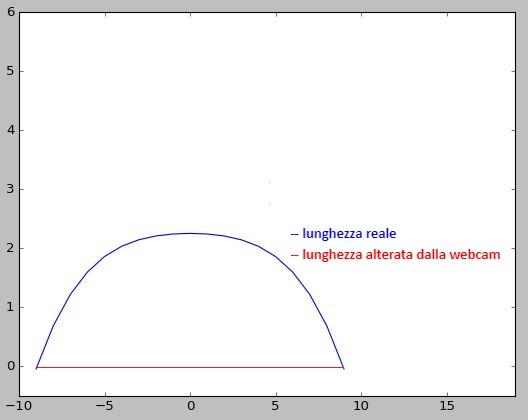
Ottenuto l’errore veritiero sarà solo necessario sottrarlo alla coordinata.

L’ultima riga di codice esegue una conversione da pixel a centimetri.

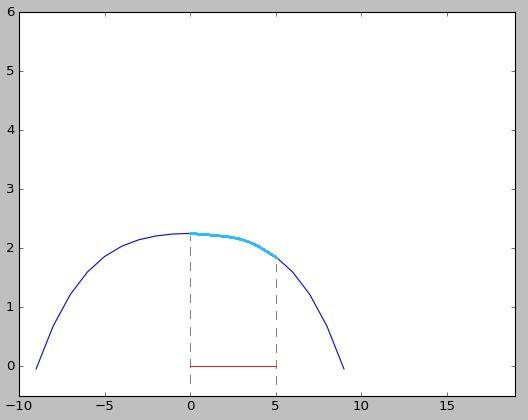
**Problema grandangolo**

Il grandangolo quindi è capace di alterare la prospettiva per catturare più superficie possibile, di fatto riesce a far stare una certa superficie in un’area notevolmente minore.

Rappresentando ciò in un grafico:



Possiamo rappresentare la realtà attraverso una curva e quello che vede la fotocamera con un segmento. L'obiettivo è trovare il corrispondente reale di ogni tratto:

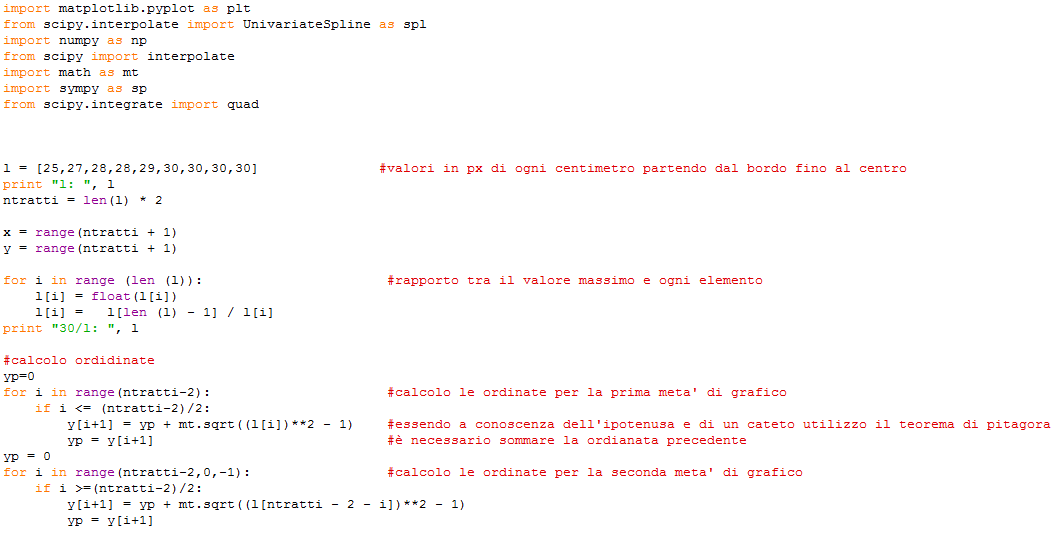
****

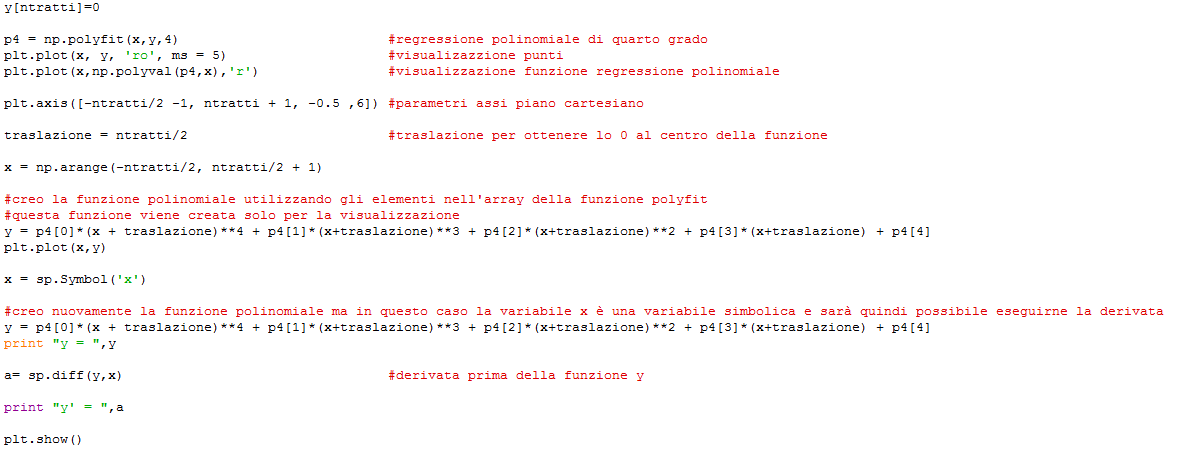
Come si vede in figura, il tratto azzurro, corrispondente alla misura reale, ha una lunghezza superiore al tratto della misura vista dalla fotocamera.

[es. una lunghezza di 5cm data dalla fotocamera corrisponderà circa 6 cm nella realtà].

**E’ quindi necessario ottenere la funzione della curva che rappresenta la realtà per misurarne successivamente la lunghezza.**

La funzione che rappresenta la realtà viene calcolata in un’altro programma:

****

****

Obiettivo del programma:

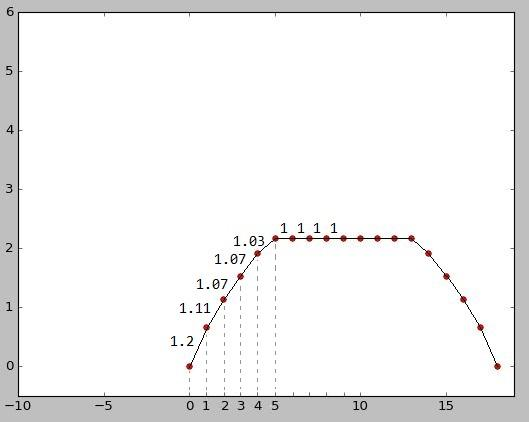
ottenere la funzione che rappresenta la realtà, affinché si possa calcolare la lunghezza del tratto della curva nel programma principale.

Siccome il grandangolo della webcam non altera l’immagine in maniera lineare, o seguendo una funzione conosciuta (es. semi cerchio o ellisse) sarà necessario ricavare tale funzione. Per effettuare ciò si prendono un numero di campioni più ampio possibile. Per velocizzare questa procedura vengono considerati solo i valori che vanno da un margine al centro del reticolo, prendendo in considerazione la simmetria della lente.

Per prendere la misura è quindi necessario:

* avere un reticolo sul piano di lavoro
* avere un programma che fornisca coordinate in px (paint)
* si inserisce nel programma la misura in px di ogni tratto corrispondente ad ogni centimetro reale

Il programma inserisce in un piano cartesiano dei punti con il seguente metodo:

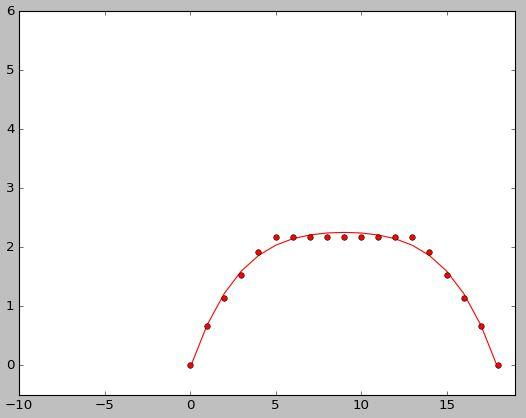


Sull’asse delle ascisse avremo i centimetri alterati. La lunghezza di ogni tratto (nero, compreso tra due punti) corrisponde al rapporto tra il valore in px del tratto più vicino allo 0 nel reticolo e la misura in px di ogni centimetro alterato.

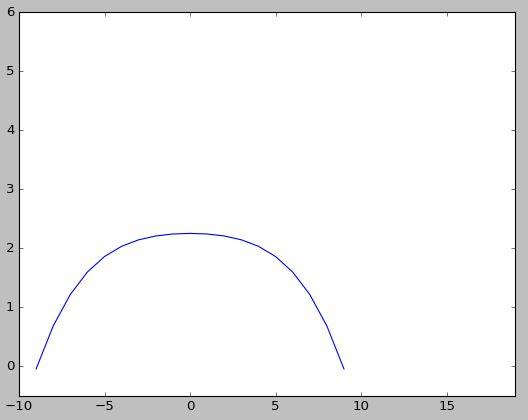
Il tratto più vicino allo 0 nel reticolo corrisponde al tratto più vicino al centro dell’immagine, che non subisce distorsione. Come si nota in figura, infatti, nella parte centrale non vi è alterazione (pendenza = 0).

Maggiore è la pendenza del tratto, maggiore sarà l'alterazione.

Ciò che si vuole ricavare è quindi la funzione che meglio rappresenta questo andamento. Viene nuovamente utilizzata la regressione. Maggiore è il grado della regressione, più la funzione creata segue l’andamento dei punti. Si è scelta una regressione di quarto grado poiché ha un buon rapporto tra complessità della funzione e precisione.



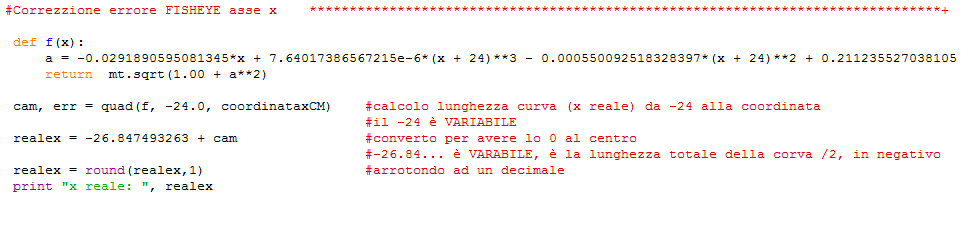
La funzione ricavata viene successivamente traslata per avere lo 0 al centro della funzione.



Si ricava quindi la funzione finale. Come vedremo in seguito, nel programma principale necessitiamo della derivata per cui viene eseguita la derivata prima attraverso l'apposita funzione della libreria sympy. Quest’ultima viene visualizzata come output per essere copiata ed incollata nel programma principale.



Inseriamo la funzione all’interno della variabile “a” del programma principale.



Questa parte del programma effettua la misura della lunghezza della curva.

Per misurare la lunghezza della curva di una funzione in un intervallo definito si utilizza la seguente formula:

****

Il dato viene calcolato tra il limite minimo e la coordinata (quindi è un valore sempre positivo), per poi essere convertito per averlo come numero compreso tra -24 e +24 (calcolato con i grafici degli esempi precedenti sarebbe tra -9 e +9 ).

**Abbiamo quindi ottenuto un programma che, dategli in ingresso la coordinata in cm della misura della telecamera, restituisce la coordinata reale.**

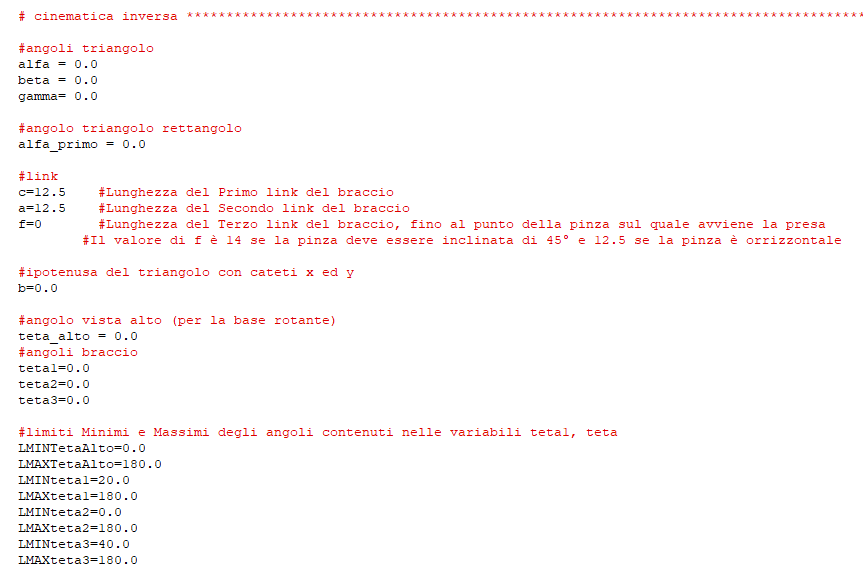
**CONCLUSIONE:**

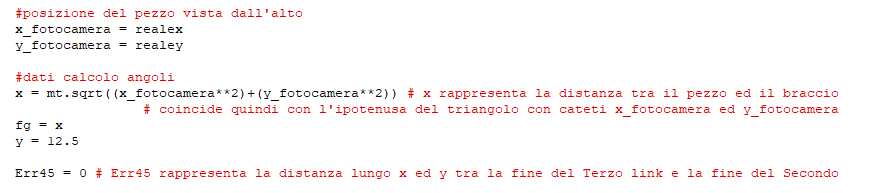
I due programmi vengono messi in sequenza nel programma principale per cui: dato in input le coordinate date dalla telecamera si avranno in output le coordinate veritiere.

**Cinematica inversa e comunicazione seriale**

Python calcola, attraverso formule trigonometriche, gli angoli che i servomotori del braccio dovranno assumere in base alla posizione del pezzo da lavorare, unisce i dati in una singola stringa e li invia tramite comunicazione seriale ad Arduino Ide.

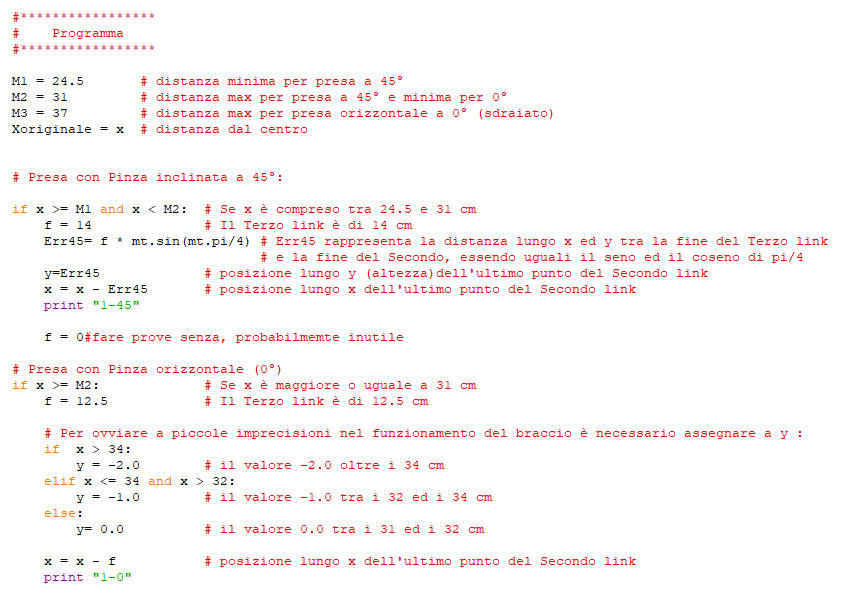
**Variabili per link ed angoli:**





Si stabiliscono i valori iniziali delle principali variabili utilizzate dal programma; appositi commenti in rosso e figure utilizzate nelle pagine successive mostrano la loro funzione.

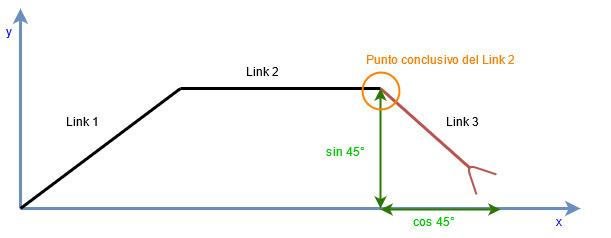
**Presa a 45° e 0°:**



**Presa con pinza a 45°:**

Se il pezzo da lavorare si trova **tra 24.5 e 31 cm** di distanza dal braccio esso viene preso con la pinza a 45°.

Conoscendo già la posizione del terzo link sarà ora necessario **calcolare la posizione**, rispetto agli assi x ed y, **del punto finale del secondo link**.



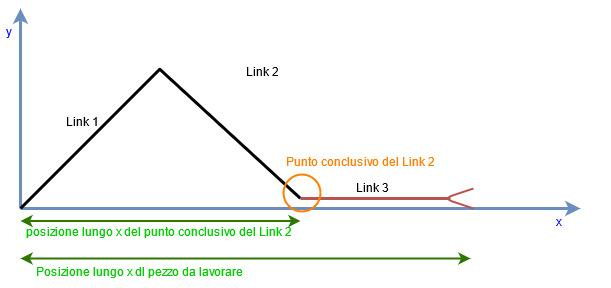
Per trovare il valore lungo **l’asse y** sarà quindi sufficiente moltiplicare la lunghezza **f** del terzo link per il **seno di 45°.**

La distanza lungo **l’asse x** sarà invece ottenuta sottraendo alla distanza tra il centro del braccio ed il pezzo da lavorare la lunghezza **f** del terzo giunto moltiplicata per il **coseno di 45°** o, essendo uguali il seno ed il coseno di 45°, sottraendo il valore di y alla distanza tra il centro del braccio ed il pezzo.

**Presa con pinza a 0°:**

Se il pezzo da lavorare si trova **tra 31 e 37 cm** di distanza dal braccio esso viene preso orizzontalmente, con la pinza a 0°.

Conoscendo già la posizione del terzo link sarà ora necessario **calcolare la posizione**, rispetto agli assi x ed y, **del punto finale del secondo link**.



Il valore lungo **l’asse y** sarà **0** ma, al fine di correggere imprecisioni imposte dalla tecnologia utilizzata, saranno assegnati valori progressivamente minori, **-1 sopra i 32 cm di distanza e -2 sopra i 34**, con l’aumentare della distanza dal braccio.

La distanza lungo **l’asse x** sarà invece calcolata **sottraendo** alla distanza tra il centro del braccio e il pezzo da lavorare la lunghezza del **terzo link**.

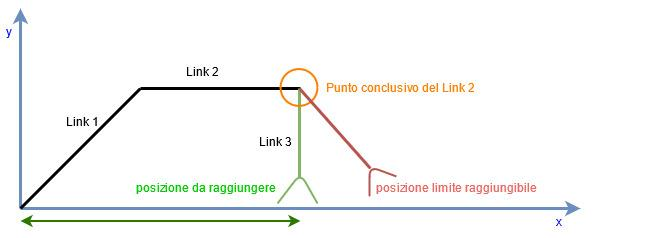
**Valore f del terzo link:**

Nel caso in cui la presa avvenga a 45° sarà utilizzata la parte finale della pinza per afferrare il pezzo e la lunghezza **f sarà di 14 cm.**

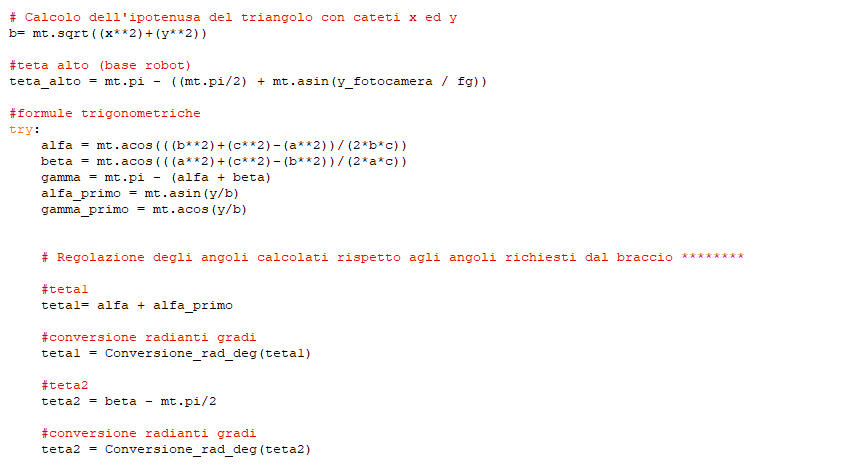
Se la presa avviene orizzontalmente il pezzo viene afferrato dalla parte centrale della pinza, risultando in una lunghezza f minore rispetto al caso precedente. Il valore di **f** viene così potato a **12.5 cm** (per ovviare ad alcune imprecisioni dovute alla tecnologia utilizzata il valore può essere portato a 13 o 13.5 cm).

**Problematiche per la presa a 90°:**

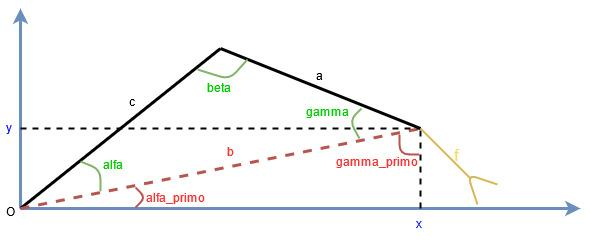
Non è possibile effettuare la presa a 90° perché per effettuarla sarebbe necessario richiedere al terzo servomotore di assumere un valore di 0°, purtroppo il valore minimo che esso può assumere è di 40°.



**Trigonometria:**



La figura sottostante rappresenta la posizione degli angoli e dei link nel braccio robotico:

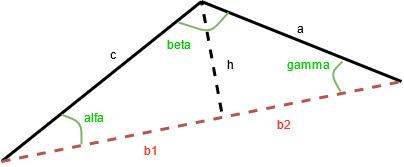


**Spiegazione dei calcoli relativi agli angoli del triangolo abc:**

Attraverso l’utilizzo del teorema di Pitagora viene calcolato l’ipotenusa b.

Consideriamo ora il triangolo abc, del quale conosciamo tutti i lati, e ricaviamone gli angoli con i seguenti passaggi:

1. Indichiamo l’altezza h e separiamo il lato b in due parti, b1 e b2, in modo che:



1. Applicando il teorema di Pitagora al triangolo ahb2 possiamo affermare che:

Essendo b2 = b – b1 riscriviamo come:

1. Ricaviamo h^2:

risolvendo:

Quindi:

1. Applichiamo il teorema di Pitagora al triangolo hcb1 :
2. Sostituiamo ad h^2 quanto calcolato nel passaggio 3 :

Sommiamo da entrambi i lati dell’uguale +b1^2 +2bb1 :

1. Possiamo quindi dire che b1 è uguale a :
2. Ricordiamo che b1 è uguale al coseno dell’angolo alfa moltiplicato per il lato c:

Quindi:

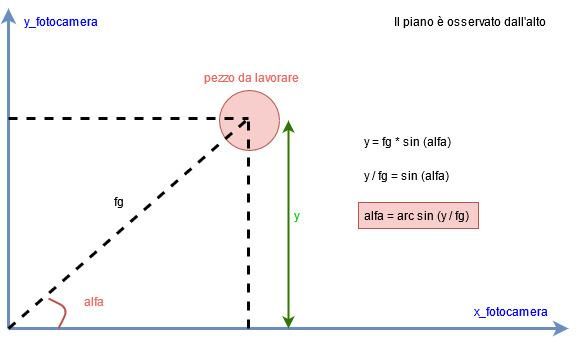
1. Ripetiamo lo stesso procedimento per gli angoli beta e gamma ed otteniamo che:

**Quindi:**

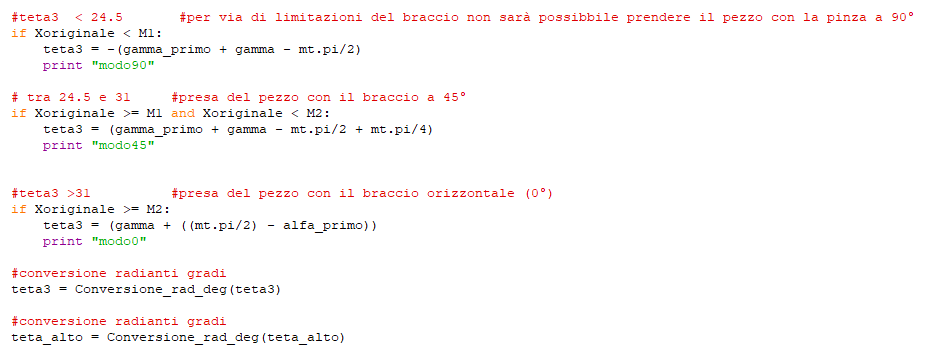
Possiamo quindi affermare che, in un triangolo rettangolo del quale si conoscono solo i lati, un qualsiasi angolo si può ricavare calcolando l’arco coseno della frazione con a denominatore il doppio prodotto dei lati che formano l’angolo considerato e a numeratore la somma del quadrato dei lati, ponendo segno negativo al quadrato del lato opposto all’angolo preso in considerazione.

1. Sarà poi necessario sommare alfa\_primo ad alfa e gamma\_primo a gamma per ottenere l’angolo completo.

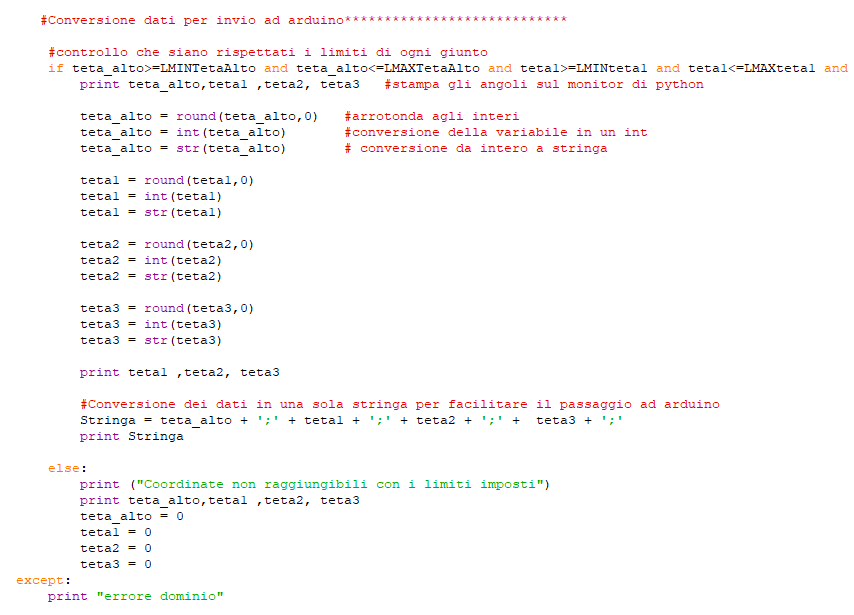
**Calcolo dell’angolo teta\_alto per la rotazione della base:**



**Regolazione degli angoli calcolati rispetto agli angoli richiesti in ingresso dai servomotori:**



**Conversione dei dati per l’invio ad Arduino:**

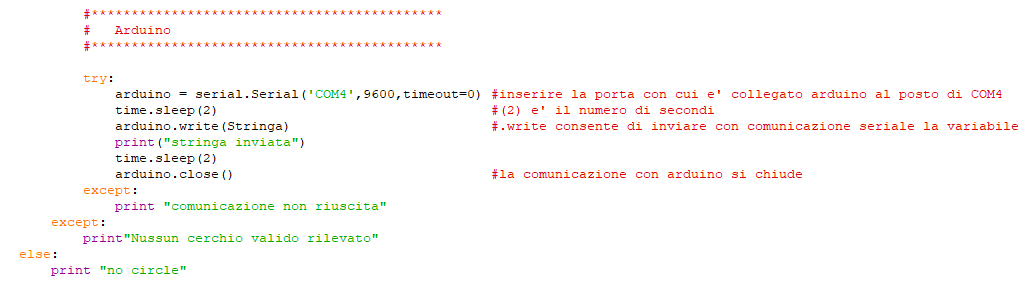


Si effettua un controllo per verificare che tutti gli angoli siano compresi entro i limiti del servomotore, se non vi sono problemi si procede all’arrotondamento del valore, che sarà convertito in un numero intero, altrimenti viene stampato sul monitor di Python il messaggio “Coordinate non raggiungibili con i limiti imposti” e i valori degli angoli vengono azzerati.

I valori interi vengono convertiti in stringhe di testo ed inseriti, separati **dal segno ;** tra loro, nella variabile **Stringa**, in modo da facilitare la comunicazione attraverso l’utilizzo di una singola variabile contenente tutti angoli da inviare ad Arduino Ide.

Nel processo attraverso la funzione print le variabili vengono stampate sul monitor di Python, in modo da poter osservare se il processo si è svolto correttamente.

**Invio dei dati ad Arduino:**



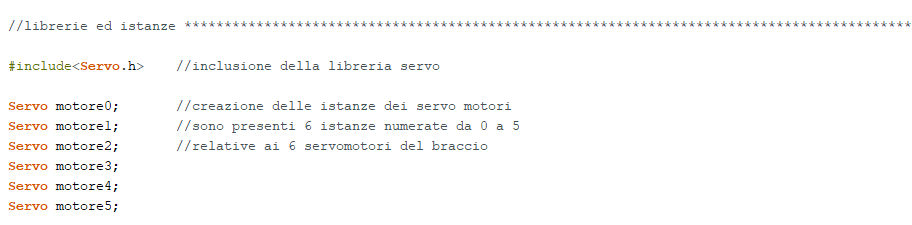
Attraverso la funzione **serial.Serial()**, contenente il nome della porta COM utilizzata e della velocità di comunicazione (in bit al secondo) comuni a Python ed Arduino Ide, viene stabilita la comunicazione seriale.

Attraverso **arduino.write()** la variabile Stringa viene inviata e, dopo due secondi, attraverso **arduino.close()**, si arresta la comunicazione seriale.

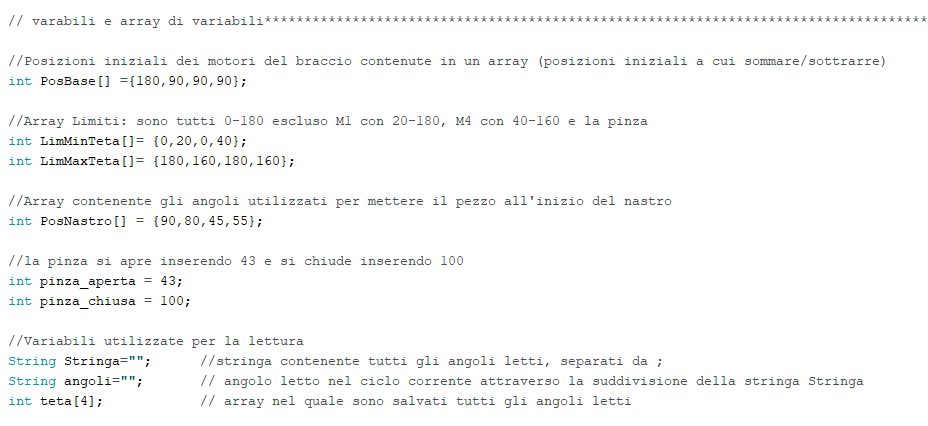
**Arduino: comunicazione seriale e controllo del braccio**

Arduino si occupa della recezione dei dati inviati da Python, della suddivisione in più variabili della stringa ricevuta e del controllo dei movimenti effettuati dai servomotori presenti nel braccio.

**Librerie, variabili:**



Viene inclusa la libreria **Servo.h**, contenente le funzioni necessarie alla gestione dei servomotori, e sono create 6 istanze per il collegamento con la libreria, che dovranno poi essere collegate ad uno specifico Pin.



Vengono create le principali variabili ed array di variabili utilizzati nel programma:

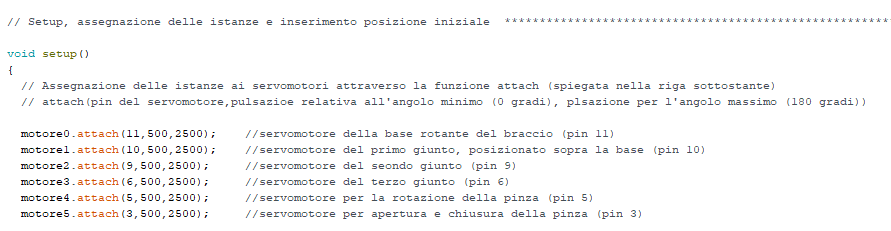
**PosBase[]** contiene gli angoli da assumere durante l’attesa dei dati inviati da Python, **LimMinTeta[] e LimMaxTeta[]** contengono i limiti minimi e massimi degli angoli, **PosNastro[]** contiene gli angoli da assumere per raggiungere il nastro sul quale il pezzo deve essere lavorato**, pinza\_aperta e pinza\_chiusa** rappresentano gli angoli per l’apertura e la chiusura della pinza.

**Stringa** rappresenta la variabile ricevuta da Python, **angoli** costituisce una locazione temporanea per i valori separati contenuti in Stringa, **teta[]** è l’array nel quale gli angoli saranno inseriti alla fine della suddivisione.

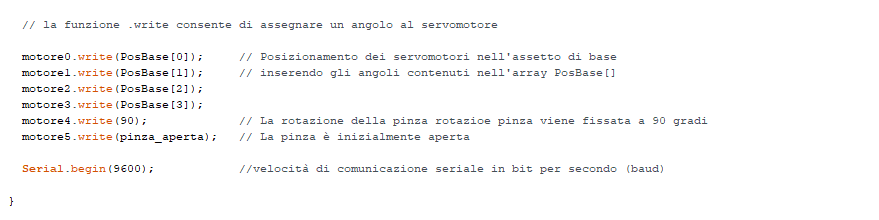
**Setup:**

La funzione setup è una fase del programma che viene effettuata una sola volta, prima dell’inizio del ciclo continuo.

In genere il setup è utilizzato per inizializzazioni, collegamenti tra pin e librerie e, più in generale, per far assumere al processo un assetto che non sarà più necessario richiamare nelle fasi successive.



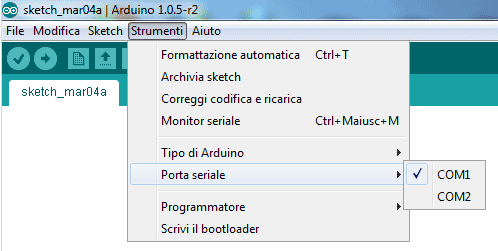
Le istanze create in precedenza vengono collegate ai Pin dei relativi servomotori e vengono assegnati i valori di pulsazione relativi all’angolo minimo ( 0 gradi) e l’angolo massimo (180 gradi)



Ai servomotori viene assegnata una posizione iniziale, mantenuta fino all’arrivo dei dati inviati da Python, attraverso i valori degli angoli contenuti nell’array PosBase[] e, attraverso la funzione **Serial.begin()**, viene indicata la velocità della comunicazione seriale (9600 bit al secondo).

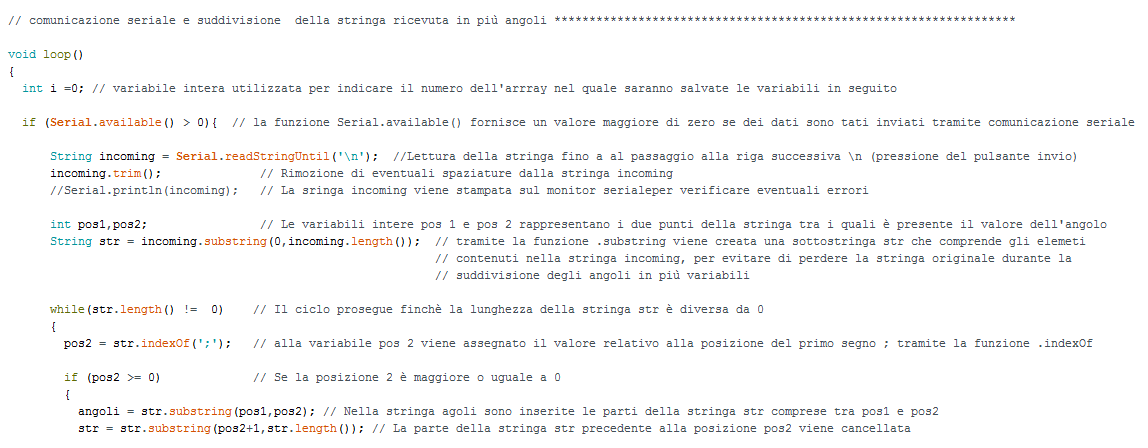
**N.B:**

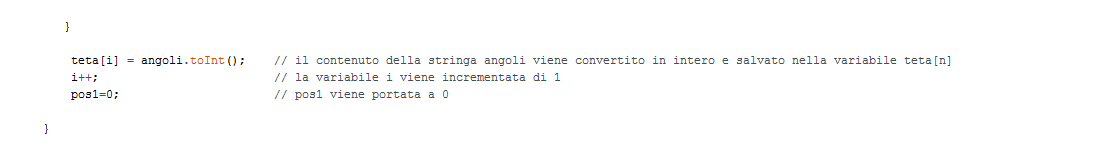
Per il corretto funzionamento della comunicazione è necessario che **Arduino Ide e Python** condividano la medesima velocità di comunicazione (9600 bit al secondo) ed utilizzino **la stessa porta COM seriale.**



**Comunicazione seriale e suddivisione della stringa ricevuta:**

Il programma **riceve** una stringa contenente tutti gli angoli da assegnare ai servomotori tramite la comunicazione seriale con Python, **rileva** i ; tra i valori nella stringa e li **separa** in più stringhe differenti, il cui contenuto sarà convertito in **numero intero** ed **inserito nell’array teta[]**.

****



**Analisi del programma:**

Attraverso la funzione **Serial.available()** si verifica se Arduino ha ricevuto variabili da Python attraverso la comunicazione seriale e, se la stringa è stata inviata correttamente.

La stringa viene letta attraverso la funzione **Serial.readStringUntil(‘\n’)** e si procede alla sua suddivisione in più variabili.

La stringa è costituita dagli angoli che sarà necessario assegnare ai servomotori del braccio separati dal segno di **punteggiatura ;** .

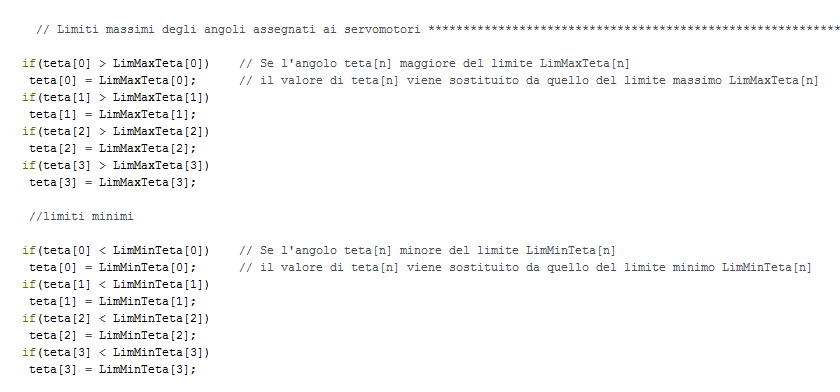
Vengono quindi selezionate due posizione nella stringa **pos1**, che rappresenta l’inizio della stringa, e **pos2** che, attraverso l’utilizzo della funzione **.indexOf()**, rappresenta la posizione del ; più vicino all’inizio della stringa.

Viene salvato nella stringa **angoli** il valore compreso tra pos1 e pos2 ed esso **viene cancellato** dalla stringa **str**.

Il procedimento prosegue fino a che tutti i valori sono cancellati e la **lunghezza della stringa str**, verificata tramite la funzione **.length()**, si è ridotta a 0.

Il contenuto della stringa **angoli** viene convertito in numero intero attraverso la funzione **.toInt()** ed è salvato nell’array **teta[].**

**Controllo dei limiti degli angoli applicabili ai servomotori:**

****

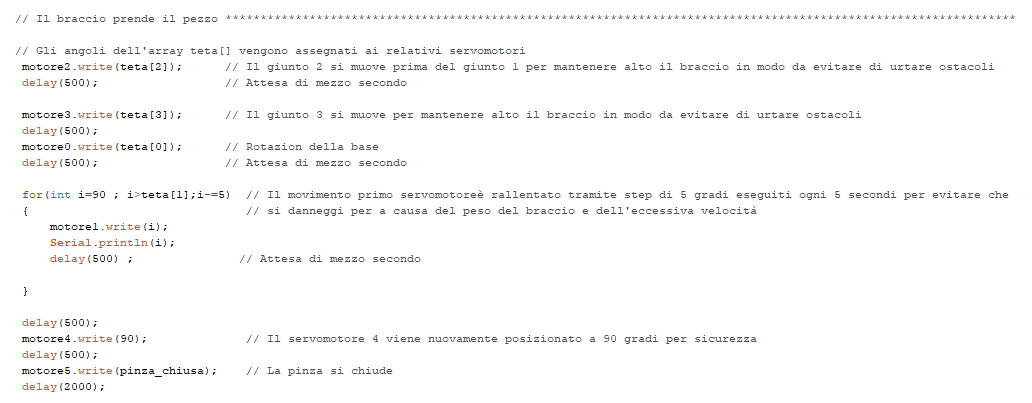
**Viene verificato** se gli angoli ricevuti sono superiori ai limiti massimi contenuti nell’array LimMaxTeta[] o inferiori ai limiti minimi contenuti nell’array LimMinTeta[], se ciò avviene il valore della di teta[] viene sostituito con quello del limite massimo o minimo.

**I limiti sono:**

* Da 0° a 180° per la base rotante
* Da 20° a 160° per il primo servomotore
* Da 0° a 180° per il secondo servomotore
* Da 40° a 160° per il terzo servomotore
* Da 0° a 180° per la rotazione della pinza
* Da 100° a 43° per apertura e chiusura della pinza



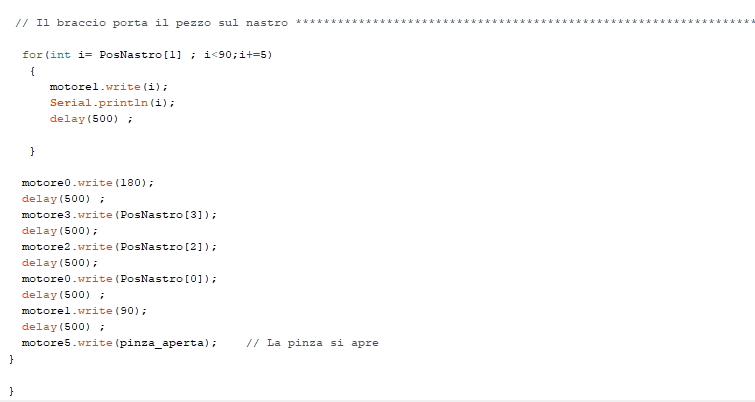
**Spostamento del braccio per la presa del pezzo e spostamento a nastro per la lavorazione:**



Per afferrare il pezzo ai servomotori sono assegnati i valori degli angoli contenuti nell’array Teta[] attrevrso la funzione **.write()** e, successivamente, la pinza viene chiusa.

Per evitare problematiche dovute alla successione troppo rapida dei movimenti vengono impostate delle brevi pause attraverso la funzione **delay(milliseconds).**

**Posizionamento su nastro:**



Per posare il pezzo sul nastro vengono assegnati ai servomotori i valori contenuti nell’array **PosNastro[]** e la pinza viene aperta, l’ordine di utilizzo dei servomotori è tale da evitare l’urto con la zona di lavorazione.

PUNZONATRICE

Questa seconda macchina ha il compito di punzonare i manufatti cilindrici, già fresati e forati, provenienti dalla catena di lavorazione e trasportati qui dal robot.

È costituita da:

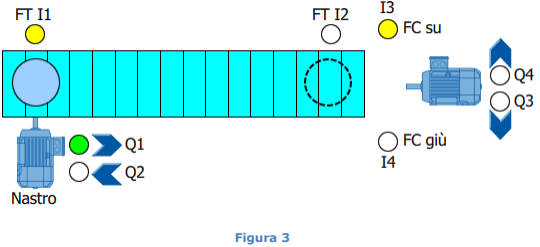
- 1 nastro trasportatore bidirezionale

- 1 punzonatore

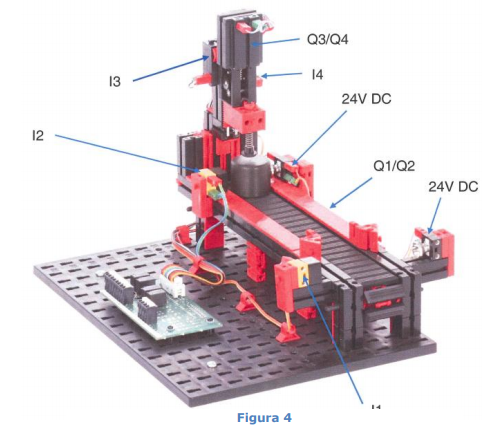
- 2 barriere fotoelettriche (NC)

- 2 contatti di finecorsa (NO)

Il nastro trasportatore consente di movimentare il pezzo dalla zona in cui viene depositato dal robot a quella dove sarà punzonato, e viceversa. Il sinottico rappresentato in Figura 3 rappresenta un esempio di stato della macchina punzonatrice. Il pezzo depositato dal robot è stato rilevato nella barriera di carico dal fototransistor I1 (luce gialla). Il motore del nastro trasportatore si è avviato (luce verde, direzione avanti) per portarlo verso la barriera di lavorazione:



Il cerchio tratteggiato indica la posizione sotto al punzonatore dove il pezzo sarà lavorato. Anche in questo caso nello sviluppo del programma che controlla la macchina punzonatrice ci siamo avvalsi di un modello in scala prodotto dalla Fischertechnik e rappresentato in Figura 4:



Per controllare questa macchina abbiamo impiegato un PLC Siemens S7- 1214C DC/DC/DC che dispone di 14 ingressi, 10 uscite digitali (a transistor) ed una connessione di rete Profinet. Le connessioni elettriche tra ingressi/uscite del PLC e del modello Fischertechnik controllato sono elencate nella seguente tabella delle associazioni:

Specifiche di funzionamento Il processo di lavorazione della punzonatrice parte dallo stato in cui il punzonatore è in alto e tutti i motori sono spenti.

1. Quando il fototransistor della barriera d’ingresso rileva il pezzo da lavorare depositato dal robot (I1 OFF), il nastro trasportatore d’ingresso si avvia in avanti (Q1 ON), portandolo verso il punzonatore;

2. quando il pezzo arriva alla barriera di lavorazione (fototransistor I2 OFF) il nastro trasportatore si arresta (Q1 OFF) e l’utensile scende per punzonare il pezzo (motore Q4 ON);

3. una volta punzonato il manufatto (finecorsa I4 ON), mentre il punzonatore risale (Q3 ON), il nastro trasportatore inverte la direzione di marcia (Q2 ON) per trasportare il pezzo verso la barriera d’ingresso;

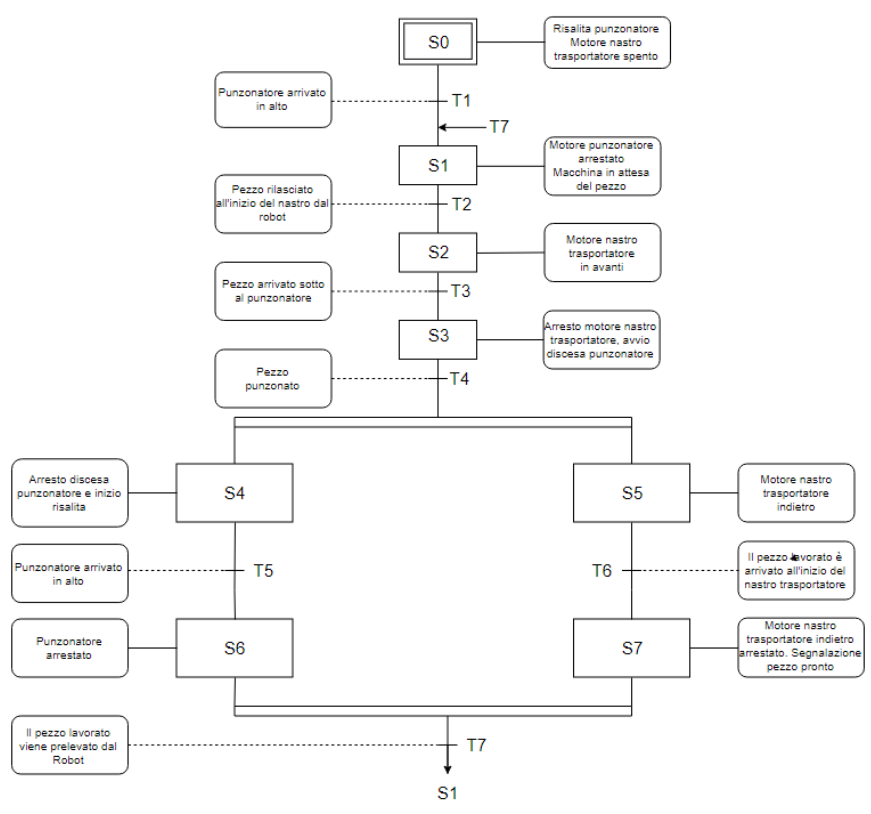
4. quando il punzonatore arriva in alto (finecorsa I3 ON) il suo motore si arresta (Q3 OFF);

5. una volta che il pezzo è arrivato alla barriera d’ingresso (I1 OFF) il motore del nastro trasportatore si ferma (Q2 OFF). Il PLC imposta a TRUE un merker di stato per segnalare al robot che il pezzo è pronto per essere prelevato;

6. quando il PLC che controlla il robot trova TRUE questo merker di stato, comanda al robot di prelevare il pezzo lavorato per portarlo verso il deposito;

7. a questo punto la catena sequenziale ricomincia dal punto 1

Anche in questo caso, il diagramma funzionale del processo di lavorazione ci consente di descrivere in modo più schematico e chiaro l’evoluzione degli stati della macchina punzonatrice:

****

Partendo da questo diagramma funzionale è stato agevole scrivere il programma di controllo della macchina punzonatrice in linguaggio KOP utilizzando ancora la metodologia batch (sequenza di passi).

