Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

Realizzazione di un sistema di controllo con auto-apprendimento per integrare l'illuminazione naturale e artificiale in un ufficio

Emilio Greco, Antonio Francesco Gentile

RT- ICAR-CS-20-02 **Maggio 2020**

Consiglio Nazionale delle Ricerche, Istituto di Calcolo e Reti ad Alte Prestazioni (ICAR)
– Sede di Cosenza, Via P. Bucci 8-9C, 87036 Rende, Italy, URL: *www.icar.cnr.it*
– Sezione di Napoli, Via P. Castellino 111, 80131 Napoli, URL: *www.icar.cnr.it*
– Sezione di Palermo, Via Ugo La Malfa, 153, 90146 Palermo, URL: *www.icar.cnr.it*

Indice generale

Introduzione

In questi ultimi anni abbiamo assistito ad un moltiplicarsi degli sforzi nella ricerca nel tentativo di risolvere il problema impellente della riduzione del consumo energetico. Allo stesso tempo la ricerca ha cercato di trovare soluzioni al fine di mantenere o addirittura aumentare lo stato di confort degli ambienti in cui viviamo o lavoriamo. Quando si parla di confort si pensa sostanzialmente a tre tipi di confort: confort visivo, confort climatico, salubrità dell'aria. Relativamente al controllo visivo sostanzialmente negli ultimi anni si è visto il delinearsi di due principali macro aree di interesse:

- lo studio di nuove tecnologie a basso consumo, come ad esempio l'illuminazione a led ed il controllo intelligente dell'illuminamento con tecniche di controllo del flusso luminoso;
- lo studio di modelli nel campo dell'edilizia pubblica e privata che mirano all'aumento dell'uso principale della luce naturale come fonte primaria di illuminamento. Questi modelli di edilizia si pongono l'obiettivo di migliorare il benessere e la salute delle persone tenendo in considerazione il naturale ritmo circadiano umano.

Questo lavoro si pone l'obiettivo di combinare, per quando possibile, i risultati prodotti da questi due filoni di ricerca. Inoltre si vuole realizzare un sistema di controllo che integri tutti e tre gli ecosistemi di controllo del confort presenti in un ambiente (luce, clima, CO2), altrimenti indipendenti, in modo che concorrono in sinergia alla definizione del confort indoor in tutti i suoi aspetti. Un approccio integrato dei tre sistemi di controllo: visivo, climatico e salubrità dell'aria ha un duplice beneficio, da un lato ha la capacità di bilanciare i diversi fattori climatici in modo sinergico per garantire un confort ottimale, dall'altro ha la capacità di riuscire ad ottimizzare la richiesta di fabbisogno energetico eliminando tutti i possibili sprechi altrimenti irrilevabili mantenendo i sistemi di controllo disgiunti.

Vista la complessità del sistema e la sua variabilità l'unico approccio a nostro avviso possibile per approntare la problematica è attraverso l'uso di metodologie di intelligenza artificiale.

Nel settore di ricerca dell'intelligenza artificiale sono stati svolti in questi ultimi anni diversi studi che hanno visto l'uso di sistemi di apprendimento automatico come elemento inserito all'interno di sistema di controllo ad anello chiuso. La logica o l'obiettivo di questo modulo integrativo è generalmente quella di apprendere i parametri di settaggio del controllore PID vero e proprio senza la necessità dell'intervento del progettista nel fare nel compiere il setup del sistema. I vantaggi di questa tecnica sono molteplici dal punto di vista operativo ed aggiungono un elemento di dinamicità o di adattabilità al controllore. La dinamicità si riferisce tuttavia alla capacità del sistema di inseguire delle derive dell'ambiente e rimodulare i parametri di controllo, non si tiene conto dell'interazione umana e delle percezioni di confort che ogni utente ha. Questo approccio se pur innovativo e molto efficiente dal punto di vista del risparmio energetico non ha ottenuto un sufficiente grado di accettabilità da parte degli utilizzatori che rimangono passivi e "subiscono" le scelte del sistema di controllo. Lo studio di questi nuovi sistemi ha quindi evidenziato un secondo problema che inizialmente era stato trascurato, ovvero ha evidenziato come ogni individuo ha una percezione dell'ambiente propria e che non è possibile definire in modo scientifico un modello di confort che vada a soddisfare contemporaneamente più utilizzatori di uno stesso sistema. Tenendo in considerazione ciò, si è pensato all' introduzione nel sistema di controllo, il feedback da parte degli

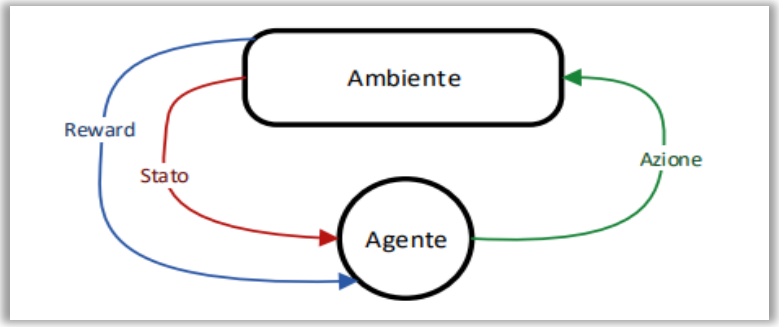
utenti, col fine di mitigare le scelte prese dal sistema di controllo. Questa scelta solleva un altro tipo di problema, cioè come far coesistere le diverse esigenze provenienti da più utilizzatori che entrano in concorrenza tra di loro ed entrano in concorrenza col sistema di controllo. Di fatti introducendo la componente energetica nel sistema di automazione, lo stesso perseguirà come obbiettivo, nella politica di scelta, il miglioramento energetico. Questa strategia o politica crea una contrapposizione, anche se minima, tra le esigenze di risparmio energetico e le esigenze di confort percepito. Per far fronte a questo tipo di problematica praticamente si fa ricorso a tecniche di gestione della concorrenza proveniente dalla teoria dei giochi o metodologie provenienti dal settore di ricerca economica come la gestione dei reclami ed il miglioramento paretiano.

Il Reinforcement Learning

L'obiettivo della ricerca in machine learning è fornire ai calcolatori l'abilità di apprendere automaticamente un comportamento sulla base di informazioni ed esempi, senza essere programmati esplicitamente per svolgere un determinato compito. In quest'ambito i metodi di Reinforcement Learning cercano di determinare come un agente razionale debba scegliere determinate azioni da eseguire a partire dalla conoscenza dello stato corrente del sistema, perseguendo l'obiettivo di massimizzare una sorta di ricompensa totale per raggiungere uno stato terminale a lui noto. La ricompensa totale è determinata sulla base di una sequenza di reward (ricompense) che l'agente ottiene nell'eseguire le singole azioni che portano al raggiungimento dello stato terminale. Il meccanismo fondamentale di tutti gli algoritmi noti del Machine Learning è il miglioramento automatico del comportamento mediante l'esperienza accumulata in fase di addestramento. Ciò vuol dire che un programma di RL dovrà scoprire, mediante ripetute prove, quali azioni permetteranno di ottenere la ricompensa maggiore.

L'idea di base che sta nella formulazione di un problema di RL è catturare gli aspetti più importanti di un problema reale, facendo interagire un agente, che è in grado di apprendere il problema in modo da fargli raggiungere l'obiettivo prefissato. Per ottenere ciò è opportuno che l'agente sia in grado di operare con l'ambiente e osservarne lo stato ad ogni istante[1]. L'agente dovrà essere in grado di eseguire azioni le quali avranno effetto sull'ambiente modificandone lo stato. Inoltre all'agente dovrà essere assegnato un obiettivo da perseguire o più obiettivi se si vuole che il sistema evolva attraversando alcuni stati desiderati e ne eviti altri indesiderati. La formulazione del problema di learning si basa dunque su questi tre aspetti: osservazione, azione e obiettivo.

[1] C'è una netta analogia con i sistemi di controllo retroazionati. Occorre individuare delle grandezze che governano il sistema, misurale, confrontarle con un valore obbiettivo e poi occorre intervenire con una attuazione per far evolvere il sistema.



La continua iterazione tra agente ed ambiente permette una continua acquisizione di conoscenza da parte dell'agente. Conoscenza che deve sfruttare in modo da massimizzare la ricompensa finale. La conoscenza acquisita non deve essere però preponderante sulle scelte dell'agente che allo stesso tempo deve esplorare nuove soluzioni in modo da scegliere azioni migliori nelle esecuzioni future. Bisogna considerare che l'agente interagisce con un ambiente stocastico, questo comporta che ogni azione dovrà essere provata più volte per ottenere una stima reale della ricompensa prevista.

Elementi del Reinforcement Learning

Oltre all'agente e l'ambiente, è possibile identificare quattro principali sotto elementi che concorrono a definire un problema di Reinforcement Learning: la reward function, la value function, la policy e, eventualmente, un modello dell'ambiente.

Abbiamo già parlato della reward function come quella funzione che associa ad una coppia stato-azione un valore numerico. Tale valore identifica nell'immediato la bontà nell'intraprendere una azione in un certo momento o in un determinato stato.

Se consideriamo il fatto che qualsiasi decisione che si prende nella vita reale richiede una valutazione preventiva tra rischi e benefici, ci si rende immediatamente conto che un modello decisionale che tenga conto della reward function come unico elemento di valutazione non porti ai risultati desiderati. In altri termini non è sufficiente per produrre una soluzione al problema che tenga conto dei rischi e dei guadagni a lungo termine.

In questo caso è necessario definire una seconda funzione, detta funzione utilità, che associa ad ogni stato un valore numerico che rappresenta la ricompensa totale che l'agente si può aspettare di accumulare nel futuro, partendo da quello stato.

Tuttavia le decisioni potranno essere prese solo sulla base di "valori stimati" (metodo bootstrap), questo perché l'obiettivo dell'agente è raggiungere un nodo terminale e massimizzare la ricompensa totale ottenuta nel raggiungerlo. La ricompensa totale sarà nota all'algoritmo soltanto al raggiungimento dell'obbiettivo. Sfortunatamente determinare i valori è più complicato che determinare i reward, questo perché i secondi vengono forniti all'agente direttamente dall'ambiente, mentre i primi devono essere stimati e ri-stimati mediante le sequenze di osservazioni dell'agente.

In problemi in cui è noto il quarto elemento che definisce un problema di Reinforcement Learning, cioè il "modello dell'ambiente", allora applicando una particolare funzione valore nota come equazione di Bellman

$$U(s) = R(s) + \gamma \cdot \max_a \sum_{s'} T(s,a,s') \cdot U(s')$$

sarà possibile ottenere una soluzione del problema risolvendo un sistema di N equazioni in N incognite.

In questo caso con il termine "modello dell'ambiente" si intende la matrice di transizione T(s,a,s') che descrive le probabilità di transizione ottenuta nel passaggio di stato tra lo stato s ed lo stato s' seguendo una certa azione a.

Se consideriamo il classico esempio del problema del lago ghiacciato (frozen lake), dove l'agente deve decidere il percorso migliore che porta da un punto iniziale (A2) ad uno finale (C2) evitando le buche (C3), se si conoscono le probabilità di transizione per ogni stato, allora è possibile giungere ad una soluzione in maniera molto semplice risolvendo l'equazione di Bellman.

Utilizzando un metodo di calcolo iterativo per la soluzione dell'equazione di Bellman, si dimostra la convergenza dell'algoritmo dopo tre iterazioni:

Il metodo risolutivo visto prende il nome di **programmazione dinamica (DP)**. È applicabile a problemi di cui è noto il modello dell'ambiente. Un limite all'applicabilità di questo approccio risolutivo consiste nel fatto che ciascuno dei passaggi iterativi contiene una scansione completa dello spazio degli stati e se il numero degli stati è alto, i tempi di convergenza sono molto lunghi.
I metodi DP convergono alla soluzione ottima con un numero di operazioni polinomiali rispetto al numero di stati m e azioni n. I metodi DP aggiornano le stime dei valori degli stati, sulla base delle

stime dei valori degli stati successori, ovvero aggiornano le stime sulla base di stime passate. Ciò rappresenta una proprietà speciale, che prende il nome di bootstrapping. Diversi metodi di RL eseguono bootstrapping, anche metodi che non richiedono un perfetto modello dell'ambiente, come richiesto dai metodi DP.
Un altro approccio per la soluzione di questi problemi, prende il nome di metodo **Monte Carlo (MC).** Considerando il fatto che quasi sempre è complicato realizzare un modello di un ambiente per le dinamiche che vi sono dietro, allora si preferisce lasciare questo compito ad un algoritmo. Individuati un certo numero di stati esaustivi e mutuamente esclusivi che descrivono un ambiente ed un certo numero di azioni necessarie per descriverne l'evoluzione, il metodo Montecarlo di fatto mira alla ricerca di percorsi (soluzione del problema) da un nodo (stato) iniziale verso i nodi terminali facendo delle stime di bontà dei percorsi ottenuti sulla base della somma totale di reward. Stima ottenuta in media negli episodi passati. Questo metodo si basa sul presupposto che l'esperienza sia divisa in episodi, e solo una volta portato a termine un episodio avviene la stima dei nuovi valori e la modifica della **policy**.
La policy in genere viene definita come una regola stocastica mediante la quale l'agente seleziona l'azione in funzione dello stato corrente. Dato che l'obiettivo dell'agente è massimizzare la quantità totale di reward ottenuta nel tempo, la policy dovrà essere greedy rispetto alla funzione valore, preferendo la scelta di azioni stimate come migliori dalla funzione valore, dove per migliore azione, si intende l'azione con valore maggiore in un dato stato.
A differenza dei metodi DP che calcolano i valori per ogni stato, i metodi Monte Carlo calcolano i valori per ogni coppia stato-azione, questo perché in assenza di un modello, i soli valori di stato non sono sufficienti per decidere quale azione è meglio eseguire in un determinato stato.
Inoltre i metodi Monte Carlo differiscono dai metodi DP per due motivi. Per prima cosa i metodi MC apprendono direttamente da campioni di esperienza, e quindi è possibile apprendere le dinamiche dell'ambiente in linea con l'acquisizione dell'esperienza da parte dell'agente in assenza di un modello.
L'apprendimento mediante **Temporal Difference (TD)** è una combinazione delle due idee. Si apprende direttamente dall'esperienza in assenza di un modello come i metodi MC e come i metodi DP, i metodi TD aggiornano le proprie stime basando il calcolo in parte su stime passate (eseguono bootstrap). Ovvero ogni scelta che viene fatta, viene "ponderata" in funzione dell'esperienza acquisita dalle precedenti iterazioni.

L'agente deve sfruttare quello che già conosce in modo da massimizzare la ricompensa finale, ma nel contempo deve esplorare in modo da scegliere azioni migliori nelle esecuzioni future.

L'agente in fase di apprendimento basa le proprie decisioni sullo stato percepito dell'ambiente, costruendo di volta in volta "il modello dell'ambiente". Per modello si intende un'entità in grado di simulare il comportamento della parte di realtà rappresentata. Per esempio, dato uno stato e un'azione, il modello è in grado di predire il risultato del prossimo stato e prossimo reward.

Affinché l'algoritmo sia in grado di costruire un modello dell'ambiente che ne descrive in tutto la sua dinamica è indispensabile che il problema così come formulato, goda della cosiddetta **proprietà di Markov.** Ovvero ogni stato che rappresenta l'ambiente non contiene memoria e quindi è possibile conoscere o determinare lo stato futuro in cui evolve il sistema a seguito di una azione basandoci solo sulle informazioni contenute nella rappresentazione dello stato attuale.

Verificato che un problema gode della proprietà di Markov, allora sarà possibile costruire ed utilizzare un modello del problema realizzato attraverso una matrice di conoscenza Q che consideri tutte le coppie stato-azione del sistema.

Algoritmo Q_Learning

Uno degli algoritmi più importanti relativi alla risoluzione di problemi di apprendimento è il Q-Learning e si basa sull'utilizzo della funzione Q(s,a). Apprendere la funzione Q equivale ad apprendere la politica ottima. Per apprendere Q, occorre un modo per stimare i valori di addestramento per Q a partire solo dalla sequenza di ricompense immediate r in un lasso di tempo. La funzione Q viene aggiornata utilizzando la seguente formula:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$

L'implementazione degli algoritmi appena trattati avviene attraverso l'utilizzo di una matrice avente un numero di righe pari al numero degli stati e un numero di colonne pari al numero delle possibili azioni. In ogni cella viene memorizzata una stima della funzione stato-azione Q(s,a).

Durante la risoluzione di un problema di Reinforcement Learning bisogna essere in grado di bilanciare la fase di esplorazione e di sfruttamento. L'esplorazione (exploration) dello spazio delle azioni permette di scoprire azioni che portano a ricompense migliori. Un agente che esplora solamente difficilmente riuscirà a convergere ad una policy ottima. Le azioni migliori vengono scelte ripetutamente (sfruttamento) perché garantiscono una ricompensa massima (reward). Il bilanciamento può essere ottenuto attraverso la tecnica ε-greedy. Questo metodo permette di scegliere l'opzione greedy per la maggior parte delle volte al fine di sfruttare l'informazione immagazzinata fino a quel momento e massimizzare la ricompensa totale e con probabilità ε di scegliere una azione in maniera casuale così da poter esplorare eventuali policy maggiormente produttive. Valori di ε troppo grandi portano a tempi di convergenza elevati mentre valori troppo piccoli non permettono di trovare la policy ottima. Il parametro α [0,1] influenza notevolmente le prestazioni degli algoritmi infatti per valori di α molto piccoli l'apprendimento è rallentato, mentre per valori di alpha troppo elevati l'algoritmo rischia di non convergere. Nell'implementazione degli algoritmi si inizia con un valore molto grande per poi farlo decrescere all'aumentare delle esplorazioni. Le prestazioni degli algoritmi presentati sono influenzati anche dalla inizializzazione della matrice Q(s,a) e dalla scelta delle ricompense nel caso di raggiungimento o non raggiungimento dell'obiettivo. Si è dimostrato come inizializzare una Q(s,a) con valori tutti diversi da zero porti ad avere una maggiore velocità di convergenza della soluzione rispetto a Q(s,a) con tutti valori nulli.

Elementi di illuminotecnica

I problemi progettuali che si presentano a chi intende occuparsi di illuminazione naturale degli ambienti risultano molto complessi. Se da una parte infatti le sorgenti artificiali mantengono costante la loro luminanza nel tempo, dall'altra quelle naturali, sole e volta celeste, presentano una luminanza variabile in funzione delle condizioni meteorologiche (cielo sereno o coperto), dell'ora e del giorno dell'anno considerati. Di conseguenza il campo luminoso naturale all'interno degli ambienti varia non solo da punto a punto ma anche nel tempo.

Il calcolo dettagliato delle condizioni di illuminazione naturale nei diversi punti di un ambiente risulta un obiettivo assai ambizioso e solo l'utilizzo di modelli matematici complessi, in grado di tenere conto in maniera corretta sia della distribuzione di luminanza del cielo nelle diverse condizioni sia dei fenomeni di riflessione multipla della luce sulle superfici, permette la descrizione attendibile delle condizioni di illuminamento.

All'interno di un ambiente chiuso, l'illuminamento naturale nei diversi punti dello spazio è determinato dal flusso di luce proveniente dalle sorgenti primarie esterne, la volta celeste, il sole, i diversi elementi del paesaggio urbano prospiciente la finestra (campo diretto), e dal flusso di luce che raggiunge il punto considerato dopo le diverse riflessioni sugli elementi che costituiscono l'involucro edilizio: pareti, soffitto, pavimento, arredamento (campo diffuso). Sulla base di queste considerazioni si è allora introdotto una grandezza sintetica e adimensionale detta fattore di luce diurna, F, in grado di descrivere le prestazioni luminose dell'involucro edilizio, la quale non dipende dal livello di illuminamento esterno, ma solo dalle relazioni geometriche tra punto considerato all'interno dell'ambiente e volta celeste.

Tale grandezza è definita come:

"il rapporto tra l'illuminamento, E, che si realizza su di una superficie orizzontale posta all'interno dell'ambiente considerato grazie alla luce proveniente dalla volta celeste (non si considera la radiazione diretta proveniente dal sole), e quello che contemporaneamente si ha su di una superficie orizzontale posta all'esterno senza alcuna ostruzione, "E_0". In base a tale definizione il fattore di luce diurna può essere calcolato con la relazione seguente:

$$F = \frac{E}{E_0} \quad (1)$$

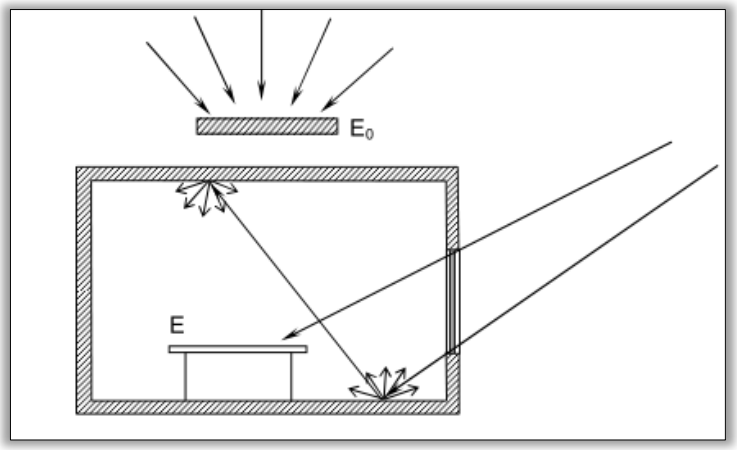
Si deve ricordare inoltre come il valore del fattore di luce diurna varia da punto a punto all'interno di un ambiente. Si introduce allora il fattore medio di luce diurna, F_{mld} , dove per medio si intende mediato su più punti di misura.

Tale parametro consente di valutare la capacità delle aperture trasparenti e dell'involucro di uno spazio chiuso di garantire condizioni di illuminazione naturale confortevoli e un accettabile sfruttamento della luce naturale.

Per raggiungere questi obiettivi esso deve essere superiore ad un certo valore, fissato come valore di soglia al di sotto del quale non sono verificate le condizioni di illuminazione naturali sufficienti alle

specifiche esigenze di benessere fisico e psicologico. Uno schema di valutazione indicativo è il seguente:

F < 0,3%	insufficiente
0.3% < F < 2%	discreto
2% < F < 4%	buono
4% < F	ottimo



Conoscendo il fattore di luce diurna è sempre possibile risalire al valore assoluto di illuminamento corrispondente all'interno dell'ambiente sul piano di lavoro partendo dall'illuminamento sul piano orizzontale all'esterno. Nella Circolare Ministeriale 13011 risulta interessante il paragrafo 1.3 riguardante l'illuminazione interna degli ambienti ospedalieri che così recita: "L'illuminazione naturale e artificiale degli ambienti di degenza e diagnostica (laboratori e terapie, visita medica) dovrà essere realizzata in modo da assicurare un adeguato livello di illuminazione con accettabili disuniformità di luminanza, la protezione dai fenomeni di **abbagliamento** ...." nel seguito al punto 1.3.02 prescrive i valori minimi dei livelli di illuminazione naturale e artificiale che sono riportati nella tabella seguente:

tipo di destinazione d'uso	**illuminamento [lux]**
piano di lavoro o osservazione medica (escluso quello operatorio)	300
piano di lavoro in spazi di lettura, laboratori e uffici	200
spazi per riunioni, ginnastica, etc.	100
corridoi, scale, servizi igienici, spogliatoi	80

Particolare cura deve essere posta per evitare fenomeni di **abbagliamento** sia diretto che indiretto, facendo in modo che nel campo visuale delle persone non compaiano oggetti la cui **luminanza superi**

rapporti di 20 volte i valori medi. Il fattore di luce diurna (punto 1.3.02 della circolare 3151) deve risultare uguale ai seguenti valori:

tipo di destinazione d'uso	F_{mid}
ambienti di degenza	0,03
palestre, refettori	0,02
uffici, spazi di distribuzione, scale	0,01

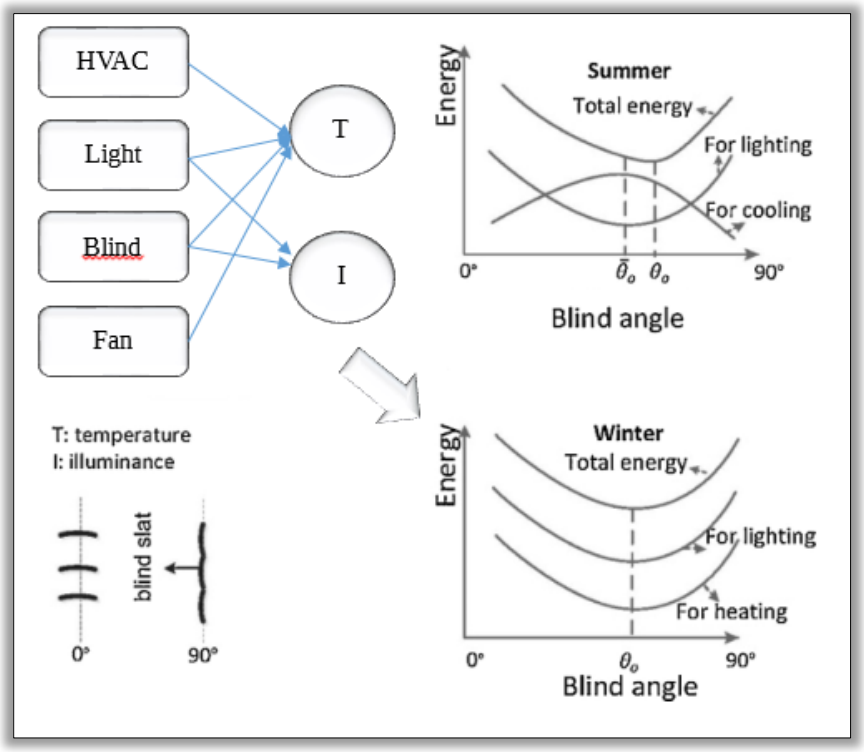
DIN EN 12464-1: Requisiti minimi raccomandati per l'illuminazione di uffici

Tipo di locale, di attività e di compito visivo	Intensità di illuminazione sulla superficie del compito visivo in lux	Valore limite UGR (limite di abbagliamento diretto)	Ra (indice di resa cromatica del corpo illuminante)
Posizionamento e copia	300	19	80
**Scrittura, lettura, elaborazione elettronica di dati	500	19	80
**Disegno tecnico	750	16	80
**Postazioni di lavoro CAD	500	19	80
Locali per conferenze e colloqui	500	19	80
Aree di reception	300	22	80
Archivi	200	25	80

Formulazione del problema

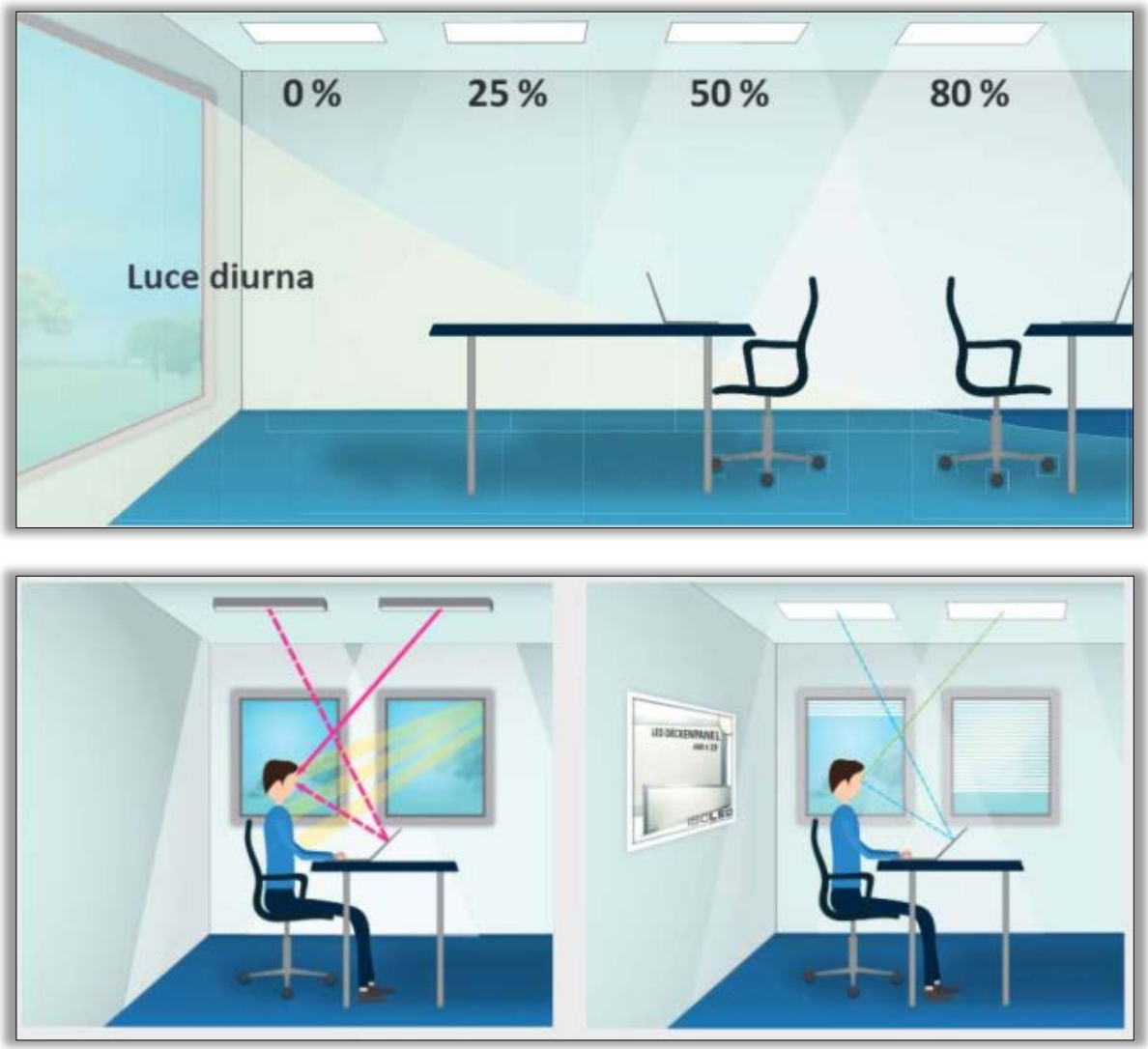
Alla luce delle considerazioni fatte nel paragrafo precedente risulta ora più chiaro comprendere come l'utilizzo di un sistema di controllo basato sul reinforcment learning sia la strada maestra per risolvere questo tipo di problemi. Detto ciò, è da aggiungere che sotto questo punto di vista sono stati effettuati diversi studi che hanno condotto a risultati soddisfacenti ma che riportano ancora alcuni problemi e margini di miglioramento. Di fatti utilizzare due sistemi automatici separati, uno per la gestione del clima ed uno per l'illuminazione non conduce necessariamente al raggiungimento di un ottimo, sia perché in molti casi non viene tenuto in considerazione l'apporto energetico dovuto all'irraggiamento esterno e sia perché sistemi separati evolvono con logiche anche in antagonismo tra loro. Diversamente, si può intuire come un approccio che integri i due sistemi sia più performante sia in termini di risparmio energetico che di soddisfazione da parte degli utilizzatori, bilanciando correttamente e contemporaneamente i parametri di controllo ad essi associati. Altro tipo di problema, riscontrato nei precedenti lavori di ricerca, è l'assenza di un feedback dell'operatore che genera un contrasto tra lo stesso sistema di controllo e la soddisfazione dell'utente. In merito a quest'ultimo, in questo lavoro viene sviluppato un sistema di controllo ad anello chiuso basato sulla soddisfazione dell'utilizzatore al fine di mediare le esigenze di risparmio energetico, più di pertinenza del sistema di controllo, con le esigenze specifiche di confort degli utilizzatori finali.

In particolare verrà usato un controller basato sul concetto di apprendimento per rinforzo che va ad implementare una strategia di controllo ottimale tenendo in considerazione il feedback dell'utilizzatore, i parametri climatici ed i set-point stabiliti dalle norme. Questo viene reso possibile attraverso un'interfaccia utente disponibile sul proprio smartphone o pc, molto semplice e intuitiva. Il sistema raccoglie le percezioni degli utenti, che insieme ai dati raccolti dai sensori vanno a determinare ed aggiornare costantemente i valori di set-poit, questo consente di delimitare la zona di confort da quella di non confort col fine di garantire un servizio personalizzato all'utente.
L'immagine seguente descrive in dettaglio il focus del progetto:



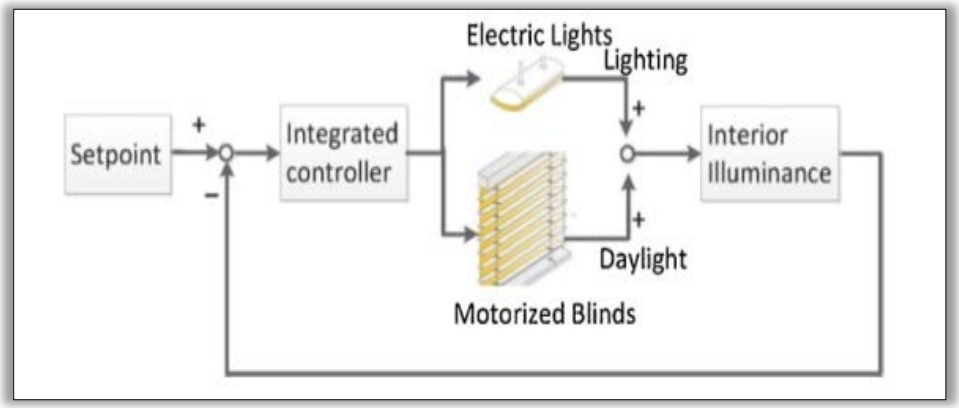
L'obiettivo che sta alla base di questo sistema è di determinare in ogni istante della giornata ed in qualsiasi stagione l'altezza di aperture e l'angolo di orientamento delle tapparelle e quindi in seconda istanza la percentuale di illuminazione artificiale che garantisca la miglior efficienza energetica e la migliore soddisfazione degli utenti. Le luci vengono regolate in modo incrementale per compensare la luce diurna qualora risulti insufficiente. Allo stesso tempo in mancanza di occupanti, in estate la tapparella si chiude per evitare l'apporto termico ed in inverso si apre al massimo per sfruttare l'effetto contrario. Questa strategia di gestione produce i risultati mostrati in figura. In alto l'effetto dell'ombreggiamento nel periodo estivo ed in basso lo sfruttamento dell'effetto serra in inverno.

La strategia di controllo dell'illuminamento, in accordo con le disposizioni normative ed in linea con le ultime ricerche in tema di confort e salubrità degli ambienti, prevede l'uso combinato di corpi luminosi con basso consumo energetico ed a elevato confort visivo in combinazione con tapparelle automatizzate che consentono una regolazione in altezza e sull'inclinazione delle doghe. Il sistema oltre a garantire un giusto apporto luminoso in funzione delle zone, impedisce fenomeni di abbagliamento, consentono il giusto grado di confort come mostrato nelle seguenti figure:

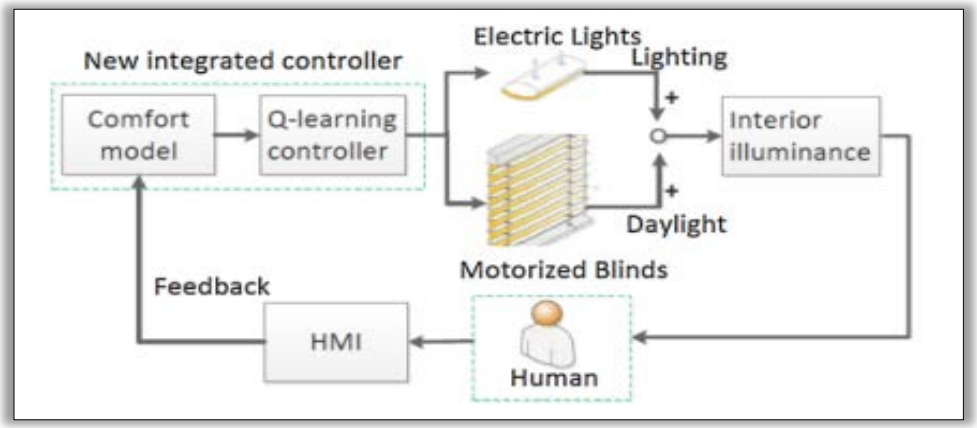


Tutto ciò è reso possibile in quando non esiste un'unica combinazione di settaggio dei vari dispositivi che permette la soddisfazione delle specifiche di confort. La possibilità di "**scegliere**" tra diverse combinazioni di regolazione della posizione delle tapparelle e dei corpi luminosi, è alla base dell'utilizzo della metodologia di Reinforcment learning. Questo perché è possibile pensare di "**demandare questa scelta**" ad un sistema di controllo. Un sistema di controllo così concepito deve essere guidato non soltanto dal soddisfacimento dei vincoli di confort visivo ma anche dal "**carico energetico**" che questa scelta comporta. Di fatti il sistema deve essere necessariamente guidato nelle scelte soppesando due elementi: la luminosità e l'energia, in modo da "dedurre" in modo automatico quale sia la giusta regolazione dei propri attuatori.
Nello schema rappresentato in Fig.1 vendono inseriti altri sistemi di controllo che si occupano del confort ambientale. Abbiamo precedentemente introdotto che i vari sistemi "si integrano" al fine di raggiungere un equilibrio rappresentato dall'ottimo energetico. Ma ancora non abbiamo chiarito come ciò avvenga.
In realtà fin qui sembrerebbe che il sistema di controllo dell'illuminamento continui ad essere un elemento a sé stante del sistema, così come il sistema di condizionamento ed areazione. Ebbene il fattore che accomuna i tre sistemi di controllo è **l'energia consumata**. Se la scelta di una particolare attivazione da parte del sistema di illuminamento produce come conseguenza l'attivazione anche del condizionatore o del sistema di ricircolo come effetto indiretto, è evidente che questa scelta è meno

consigliata rispetto ad una seconda scelta che eviterebbe questo sovraccarico mantenendo le stesse condizioni di confort visivo. Un sistema che reagisce a questo tipo di logica deve necessariamente prima acquisire attraverso l'esperienza questa informazione e tradurla in conoscenza che poi utilizzerà nel compiere le proprie scelte. Mettendo a fattor comune l'energia consumata dai vari sistemi si riesce così ad integrarli anche tenendoli fisicamente separati. Questo non toglie capacità decisionale rispetto ad un sistema completamente integrato anche dal punto di vista fisico, poiché ogni sotto sistema ha le sue specificità che non possono essere inglobate in un'unica legge di controllo. La parte in comune che può essere condivisa per armonizzare i diversi sistemi di controllo e che ne determina in modo incisivo la logica di attuazione è il fattore energetico. Naturalmente anche i sistemi di controllo HVAC e Fan dovranno essere concepiti con lo stesso approccio.
Detto ciò, vediamo ora le differenze tra un sistema di controllo tradizionale ed uno concepito secondo un approccio di RL. Un sistema di controllo tradizionale è basato su controllori PID ed utilizza setpoint fissi e normati. Un tipico schema di controllo è quello mostrato in figura:



Andiamo a delineare gli elementi che si differenziano con il nuovo sistema. Ciò che si vuole fare è sostituire il blocco setpoit con un blocco che chiameremo Confort model. Questo blocco si occuperà di definire un modello di confort secondo varie strategie che di volta in volta verranno ritenute più opportune (analisi paretiana, teoria dei giochi, etc.). Inserire nell'anello di retroazione il feedback dei sensori di luminosità e delle "sensazioni dell'utenza". Inserire al posto di un tradizionale controllore PID, un algoritmo di reinforcment learning, più in particolare il più tradizionale Q_learning. Lo schema generale di controllo pertanto sarà il seguente:



Il blocco presente nell'anello di retroazione denominato **HMI** rappresenta l'interfaccia utente. Questa assolve ad un duplice compito:

- serve a presentare una lista di percezioni che l'utente può esprimere, che poi traduce in eventi per il sistema;
- fornisce meccanismi di override del sistema di controllo in caso di emergenza, ovvero meccanismi di disattivazione del controllo e di attivazione manuale.

Il blocco **Confort model** assolve alla funzione di analizzare/elaborare i feedback ricevuti dagli utenti e dai sensori presenti nella stanza, col fine di delineare il lower limit **ML** ed il upper limit **MH** che separano la regione di confort dalla regione di non confort. L'informazione in ingresso a questo blocco viene così "classificata" e resa disponibile ai blocchi successivi del sistema.
Gli algoritmi di classificazione che possono essere utilizzati all'interno del blocco Confort model, così come quelli per stimare i set point ML e MH possono esse di diverso tipo. Tipicamente si fa ricorso ad una analisi statistica effettuata sulla serie di dati che riceve il sistema. Questa metodologia va sotto il nome di **miglioramento Paretiano** dal nome del suo ideatore, un economista Italiano Vilfredo Pareto.
"Il miglioramento paretiano consiste in una riallocazione delle risorse di un determinato sistema che migliorano la condizione di almeno un individuo senza peggiorare quella di altri, producendo quindi un aumento dell'efficienza complessiva del sistema".
Infine troviamo il blocco di controllo denominato **Q-Learning controller**. Questo blocco assolve alla funzione di "**scegliere**" l'azione più opportuna da intraprendere per riportare il sistema "in uno stato di confort". Per fare ciò, come detto prima, deve anche essere dotato di un meccanismo di "**apprendimento**" che gli consenta di essere in grado di poter intraprendere delle decisioni sulla base di ciò che si conosce. Entrambe queste funzionalità risiedono all'interno di un algoritmo di Reinforcment Learning. Tra i vari algoritmi di RL, quello che si adatta maggiormente al nostro sistema, a nostro avviso è l'algoritmo di Q-Learning, sia per la sua semplicità di implementazione che per la sua peculiarità di non inglobare la policy all'interno della regola di aggiornamento (algoritmo off-policy). L'algoritmo di Q_learning nell'aggiornare la sua base di conoscenza adotta la seguente strategia: per ogni stato che viene visitato durante la fase di esplorazione va ad aggiornare la coppia stato-azione che produce il massimo valore atteso. La matrice di conoscenza conterrà sempre come scelta migliore da perseguire, quella che produce la massima ricompensa attesa. Tuttavia, per garantire anche l'esplorazione di nuove soluzioni, la scelta dell'azione da intraprendere al prossimo step molte volte non coincide con l'azione che produce il massimo reward. In quella che viene chiamata "fase di Learning" la politica di scelta dell'azione da compiente è diversa dalla politica di aggiornamento della matrice di conoscenza. Usualmente si parla di Ɛ-greedy policy per dire che una certa percentuale di azioni che vengono intraprese vengono scelte in base alla conoscenza acquisita precedentemente, ed una parte di esse vengono scelte in modo random. Nel caso di Q-Learning, se la scelta viene intrapresa seguendo la conoscenza acquisita, l'azione che verrà selezionata sarà sicuramente quella che produrrà il massimo valore atteso, grazie alla logica di aggiornamento che ci siamo detti pocanzi. Non tutti gli algoritmi di RL usano questa priorità di scelta, questo dipende molto dal tipo di problema che si vuole risolvere. Ed esempio l'algoritmo SARSA non adotta questa politica nell'aggiornamento della matrice di conoscenza. Questo di fatti lo rende più "immune" alle perturbazioni prodotte dall'ambiente. In altre parole, se esistono degli stati del sistema che sono fortemente indesiderate (identificate con reward negativo) l'algoritmo SARSA "imparerà" a riconoscerle e compirà scelte non seguono la strategia di massimo reward come Q-Learning ma quelle che, a così dire, sono a più " basso rischio".
Oltre a questi algoritmi, diciamo classici, in letteratura troviamo una fiorente classe di algoritmi che fanno uso delle più moderne "reti neurali". Quest'ultime però operano secondo un approccio di tipo

Black box, non è possibile cioè verificare paso dopo passo come avviene l'acquisizione della conoscenza e la scelta delle azioni.
Fatte le opportune scelte progettuali, illustrate precedentemente la fase successiva nella realizzazione del sistema è la progettazione concettuale dell'algoritmo di RL.

La progettazione di un algoritmo di RL passa essenzialmente dalla definizione dei seguenti elementi:

- insieme degli stati che descrivono l'evoluzione del sistema;
- insieme delle azioni da intraprendere sul sistema per farlo evolvere;
- definizione degli obiettivi e delle ricompense / funzione di reward;
- definizione dell'albero decisionale /verifica del soddisfacimento della condizione di Marckov.

Definizione dello stato

Per stato intendiamo quella informazione minima e sufficiente a determinare l'evoluzione di un sistema che si desidera osservare e quindi controllare. Principalmente vengono identificate nello stato di un sistema tutte quelle grandezze fisiche che sono in qualche modo misurabili sia in maniera diretta, attraverso apparecchiature di misura, che in maniera indiretta, attraverso l'osservazione di altre grandezze ad esse correlate.

Per il controllo della luminosità di un ambiente si possono prendere in considerazione sia grandezze dirette, attraverso sensori di luminosità o l'abbagliamento, che grandezze indirette come la posizione delle tapparelle o delle luci. Quest'ultima, è una informazione da cui è possibile ricavare in maniera indiretta il grado di luminosità in un ambiente. Altro parametro indiretto che verrà preso in considerazione è il "confort percepito". Anch'esso parametro correlato al livello di luminosità ma che usa come strumento di misura la percezione dell'utilizzatore.

Dopo una accurata analisi, i parametri che sono stati definiti per descrivere lo stato del sistema sono i seguenti:

- **L**: **Stato delle luci**. Informazione sul numero di corpi luminosi accesi. In una prima realizzazione del sistema si è pensato all'utilizzo di tre lampadine controllabili singolarmente e che possiedono la stessa capacità luminosa, successivamente il sistema è stato sostituito con un gateway dali/mqtt per il controllo delle luci. Lo stato delle luci può essere riassunto in quattro posizioni : 0,1,2,3. Col significato di numero di lampade accese nella prima realizzazione o 0%,25%,50%,100% nella configurazione successiva.
- **B**: **Stato delle tapparell**e. Questa grandezza tiene conto sia dell'altezza che dell'angolo di apertura della tapparella. I valori che abbiamo ipotizzato ammissibili in questa prima implementazione sono i seguenti: (0,25,50,75) x (0°, 30° 60°, 90°)+100. Dove 100 descrive la condizione di tapparella tutta alzata; 0 tapparella abbassata. Questo tipo di rappresentazione è puramente semplificativa ed è utile a livello progettuale, a livello poi applicativo questa informazione deve essere "tradotta" in una sequenza di comandi tali da portare la tapparella nella posizione desiderata. Questo concetto verrà maggiormente chiarito in seguito.
- **I**: **human confort status**. Con questo tipo di informazione si va a modellare una delle tre possibili condizioni di percezione dell'utente: (a, b, c), ovvero "a" per sensazione di

abbagliamento, “b” per sensazione di buio e “c” per sensazione di confort. La zona di confort dal punto di vista concettuale può essere ulteriormente suddivisa in due zone: zona energy saving e zona energy wasted.
Una transizione che ci poti da uno stato di disconfort ad uno di confort energy save è l’obiettivo desiderato e perseguito. Diversamente non sono assolutamente consentite azioni che ci portano in uno stato di disconfort, mentre sono permesse ma rivalutabili azioni che ci portano in condizioni di confort energy wasted.

L’insieme degli stati che rappresentano l’evoluzione del sistema sarà quindi definito da:

S = {L, B, I}

Ed avrà una cardinalità pari ad |S| = 4x(4x4+1) x3= 204

Insieme delle azioni:

Gli attuatori su cui possiamo intervenire sono sostanzialmente due: le luci e le tapparelle. L’insieme delle possibili azioni sono quelle che ci consentono di intervenire su questi elementi. Anche nella definizione delle azioni sono possibili diverse scelte progettuali che presentano dei pro e dei contro. Si potrebbe pensare di considerare l’insieme delle azioni che ci consentono di passare immediatamente da uno stato ad un altro del sistema. In questa maniere si avrebbe un sistema immediatamente reattivo, ma allo stesso tempo avremmo un insieme cospicuo di azioni, pari nel nostro caso ad A= {L, B}, |A| = 68. Un numero alto di azioni ci porta però ad avere una dimensionalità più alta della matrice Q (memoria del sistema) e cosa più importante, un aumento considerevole di “tempo” necessario per la fase di addestramento dell’intelligenza. L’aumento di tempo per addestramento è dovuto al fatto che ci sono molte più opportunità di esplorazione nella ricerca dell’azione ottima. Un’altra modalità di procedere è quella di consentire al sistema di fare scelte di tipo incrementale. Potremmo considerare le seguenti azioni A= {incrementa/decrementa luci, incrementa/decrementa tapparella} |A|=4, in questo modo non solo otterremo una riduzione significativa della matrice Q, ma anche una riduzione significativa dei tempi di addestramento. Di contro, dato uno stato generico di disconfort, per poter arrivare ad uno stato di confort, il sistema dovrà, in alcuni casi, compiere una sequenza di transizioni di stato per portarsi allo stato desiderato.
Abbiamo adottato una soluzione intermedia tra le due scelte bilanciando la reattività del sistema con la necessità di avere tempi brevi di addestramento.
Lo spazio delle azioni definito per il nostro controllore sarà il seguente:

A = {null-alza, null-abbassa, null-inclinasu, null-inclinagiu, spegni-alza, spegni-abbassa, spegni-inclinasu, spegni-inclinagiu, accendi-alza, accendi-abbassa, accende-inclinasu, accendi-inclinagiu, accendi-null, spegni-null}

Albero decisionale – Marckov Decision Process Tree

Come anticipato nel paragrafo precedente, la matrice di conoscenza Q(s,a) che l'algoritmo di RL realizza per raggiungere una soluzione del problema assegnato, può essere rappresentata attraverso un albero decisionale. L'albero decisionale è una rappresentazione di un sistema costituito da uno stato iniziale, detto nodo di partenza, ed uno o più stati finali, detti nodo foglia. Le azioni o percorsi che collegano due stati del sistema sono detti archi. Il valore assegnato ad ogni cella della matrice Q(s,a), stimato dalla value function, rappresenta il peso di ogni arco. Trovare la politica ottima che risolve un problema di RL, equivale a trovare il percorso ottimo (percorso con peso massimo) tra il nodo iniziale ed il nodo finale, sull'albero decisionale. Gli algoritmi di TD-learning intervengono per "pesare" le scelte, ovvero gli archi del grafo, in funzione dei parametri che descrivono il sistema al fine di determinare il percorso a guadagno massimo. Durante questa fase di ricerca si ipotizza che il sistema rimanga immutato, ovvero che le condizioni di partenza non vengono perturbate.

Le assunzioni e le scelte progettuali che abbiamo fatto nei paragrafi precedenti ci portano a fare le seguenti constatazioni:

1. Il sistema potrebbe non evolvere in un solo step, o con una singola azione, dallo stato di "disconfort" in cui si viene a trovare ad uno stato terminale di confort energy save. La scelta del set di azioni messe a disposizione consente variazioni incrementali, pertanto sono possibili diversi alberi decisionali che, partendo dallo stesso nodo iniziale, seguano percorsi diversi per raggiungere lo stato finale procedendo con incrementi di una o l'altra variabile di sistema.
2. Non può esistere un nodo obiettivo (nodo assorbente) identificato come punto di uscita dell'algoritmo. Anche se l'algoritmo entra in uno stato di confort energy save, non può terminare la sua esecuzione per ovvi motivi e deve essere sempre messo in grado di poter valutare se lo stato raggiunto è di tipo energy save o energy wasted. Per fare ciò, si interviene con un timer che riattiva il ciclo di apprendimento e di ricerca di una nuova soluzione, qualora il controllore si porti in uno stato di confort.

Supponiamo che il sistema si trovi nello stato di "disconfort": "0-25-30-b". Lo stato rappresentato dalla stringa ha il seguente significato **0**: luci accese-**25**: % apertura tapparella-**30°**: di inclinazione tapparella- **b**: luminosità rilevata dai sensori sotto la soglia minima. Supponiamo che la luminosità misurata sia 150 lux a fronte di 200lux impostati da soglia.

Dato lo spazio delle azioni:

A = {null-alza, null-abbassa, null-inclinasu, null-inclinagiu, spegni-alza, spegni-abbassa, spegni-inclinasu, spegni-inclinagiu, accendi-alza, accendi-abbassa, accendi-inclinasu, accendi-inclinagiu, accendi-null, spegni-null}

Le azioni che possono essere considerate ammissibili considerando lo stato di partenza "0-25-30-b" saranno le seguenti:

A’ = {null-alza, null-inclinasu, spegni-alza, spegni-inclinasu, accendi-alza, accendi-abbassa, accendi-inclinasu, accendi-inclinagiu, accendi-null }

Le azioni non ammissibili saranno: **null-abbassa, null-inclinagiu, spegni-abbassa, spegni-inclinagiu, spegni-null**. Ovvero tutte quelle azioni che ci allontanano dalla possibilità di avere un incremento di luminosità nell’ambiente.

Inizialmente le nove azioni sono equiprobabili. La scelta è del tutto random. Supponiamo che venga scelta l’azione “**null-alza**”. Questa azione farà evolvere il sistema nel seguente stato:

S = 0-25-30-b S’ = 0-50-90-?

L’evoluzione è dovuto al fatto che la tapparella passa dall’altezza precedentemente impostata del 25% a quella del 50% e nel compiere questo movimento porta l’inclinazione delle doghe a 90°. Questo per il meccanismo di movimentazione della tapparella. Lo stato S’ sarà completamente definito quando sarà disponibile la misura del valore della luminosità. Supponiamo che il valore sia 710 lux, leggermente superiore al valore di soglia massimo impostato. Il passaggio quindi da S a S’ porta in un nuovo stato di disconfort con conseguente reward negativo e scelta di una nuova azione. Lo stato attuale quindi sarà S’=”0-50-90-a”. Le azioni ammissibili per questo stato saranno:

A’ = {null-abbassa, null-inclinagiu}

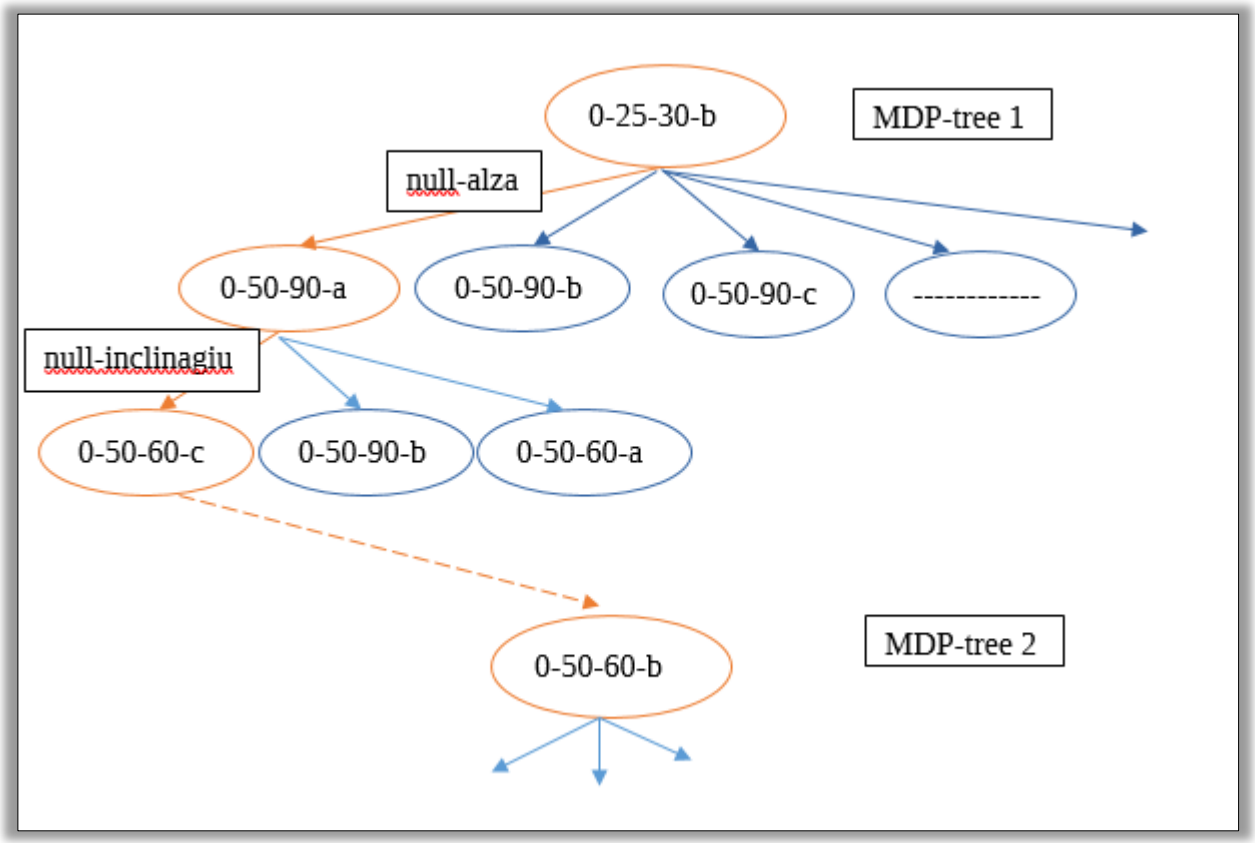
In considerazione del fatto che non è consentito spegnere le luci perché sono a 0, non è consentito alzare o inclinare in su la tapparella. Supponiamo che questa volta venga selezionata l’azione **“null-inclinagiu”,** il sistema evolverà pertanto nello stato S’’ dato da:

S’ = 0-50-90-a -> S’’ = 0-50-60-? -> S’’ = 0-50-60-c

Supponiamo che la lettura dei sensori ci dica che siamo in zona confort con 650lux. In questo caso l’azione che porta dallo stato S’ allo stato S’’ non viene aggiornata. Il passaggio da uno stato di disconfort ad uno di confort non richiede un aggiornamento immediato della matrice Q semplicemente per il fatto che si procede con la strategia delle penalità. Il valore 0 impostato di default nella matrice corrisponde di fatto al massimo valore atteso.

Mentre gli algoritmi tradizionali terminano al raggiungimento di uno stato finale, questo algoritmo permane nello stato di confort per un tempo stabilito da un timer. Supponiamo che non interviene nessun altro evento che porti il sistema in disconfort e che quindi spiri il timer. Ipotizzando che lo stato di confort individuato dal controllore “potrebbe” essere uno stato di confort energy wasted e quindi vogliamo verificare se esiste un’altra azione che ci porti in uno stato di confort energy save.
Soltanto nel caso in cui interviene un timer possiamo quindi procedere ad aggiornare lo stato S’= 0-50-90-a. Il questo caso, nonostante l’azione ci conduca in una zona di confort e che quindi non richiede nessuna penalità, dobbiamo comunque soppesare l’energia consumata e dare un feedback (negativo) al decisore su questo parametro. Aggiornata la matrice Q con il consumo energetico si continua a rimanere nello stato attuale considerato di confort. La valutazione dell’altra possibile azione “null-abbassa” a partire dallo stato S’ verrà eventualmente valutata in una successiva iterazione di apprendimento.

L’albero decisionale di Markov per il caso esaminato avrà la seguente struttura:



Reward function

La reward function è una funzione che esprime il legame esistente tra uno stato ed un’azione. Caratterizza la bontà di una azione intrapresa in un dato stato. Diversamente dalla Value function che guida l’algoritmo di controllo verso la scelta di una azione in funzione del valore atteso, la reward function rappresenta la ricompensa immediata ricevuta da quella scelta. Questo tipo di informazione deve essere data dal progettista, in qualche modo rappresenta la base di dati di partenza utilizzata dall’algoritmo per inferire nuova conoscenza. Oltre ad informazioni di “non ammissibilità” di alcune azioni, attraverso di essa vengono delineati i nodi terminali o obbiettivo dell’algoritmo e vengono identificati gli stati critici di sistema per cui si va ad istruire il decisore ad evitare azioni che vanno verso una scelta impropria.
Abbiamo già accennato nei paragrafi precedenti circa le occasioni in cui occorre definire un reward per il decisore. Di fatti, in molti casi pratici, non è possibile decidere in modo deterministico il valore di reward per tutte le azioni intraprese in ogni stato, questo implicherebbe la conoscenza dettagliata del sistema, ovvero del modello che descrive la sua evoluzione. Allora si devono individuare tutte quelle transizioni che possono essere identificate e classificate. Gli stati terminali così come gli stati critici sono i primi che richiedono una definizione del reward.
Tuttavia si cerca soventemente di trovare una rappresentazione generale di questa funzione, e nel nostro caso una possibile definizione può essere la seguente:

$$R = \rho_C r_c + \rho_e r_e$$

La prima componente di tale formulazione va a soppesare il confort ottenuto dall'azione e la seconda riguarda il risparmio energetico che viene prodotto.
Il peso che viene attribuito alle due componenti inizialmente è di 0,8 e 0,2 (80% di incidenza del fattore confort, rispetto al 20% della componente energia).

Il fattore **rc = (0,-1)**, dove 0 indica una condizione di confort ed -1 una di disconfort.

Questo valore verrà generato dal blocco "confort model" in funzione del confronto derivante tra il livello di luminosità L raggiunto e dei limiti di accettabilità calcolati (ML <= L <= MH).

Relativamente alla componente energetica **re**, questa viene ricavata come valore normalizzato dell'energia consumata dai diversi sistemi di gestione del confort e nel lasso di tempo che interviene tra il passaggio dello stato di confort e quello di spiro del timer.

Algoritmo di apprendimento

L'algoritmo Q_Learning presente in letteratura può essere descritto dal seguente pseudocodice:

1. Imposta il parametro alfa, gamma, epslon, maxstep e i premi ambientali nella matrice R.

2. Inizializza la matrice Q a zero.

3. Ripeti per ogni episodio di apprendimento:

Seleziona in modo random lo stato iniziale S_0.

Ripeti finché S_0 è terminale o si è raggiunto il numero massimo di step consentiti.

- *Seleziona l'azione a_0 dalle azioni ammissibili per S_0 con politica $\mathcal{E}$-greedy;*
- *Esegui l'azione sull'ambiente*
- *Osserva lo stato raggiunto S_1 ed il reward prodotto $R(S_1)$;*
- *Estrai dalla matrice Q il massimo valore atteso da S_1: Max [Q (S_1, all actions)];*
- *Aggiorna Q: Q (S_0, a0) = R(S_1) + Gamma * Max [Q (S_1, all actions)]*
- *Aggiorna lo stato corrente: S_0 <- S_1;*

End

End

Questo algoritmo è caratterizzato da una fase di inizializzazione dei parametri usati all'interno dell'algoritmo e della matrice Q (step 1 e 2). La matrice può essere inizializzata con valori nulli o con valori scelti in modo casuale. Per quando concerne la nostra applicazione i valori inziali di Q devono essere necessariamente pari a 0 in quando tale valore rappresenta il valore massimo atteso, ciò e

dovuto al fatto che tutti i reward prodotti dall'ambiente hanno valori negativi. Ovvero non si è deciso di procede per "premi" e "penalità" ma si considerano solo "penalità".
Relativamente allo step 3, si evidenziano due cicli innestati. Il primo ciclo definisce l'insieme di tutti gli episodi di apprendimento che vengono generati al fine di costruire la base di conoscenza rappresentata per l'appunto dalla matrice Q. Questo ciclo è indispensabile quando l'algoritmo lavora in modalità "off-line" o se vogliamo in ambiente simulato, dove si susseguono due fasi: fase di "addestramento" e fase di "produzione". Nel nostro caso questo ciclo non viene considerato perché si vuole realizzare un "**algoritmo di apprendimento online**" o real-time.
Un algoritmo di apprendimento online può aggiornarsi in modo incrementale con ogni episodio che viene osservato. Gli algoritmi di apprendimento con rinforzo come Q-learning si prestano bene ad essere utilizzati come algoritmi di apprendimento online, poiché la ricompensa per ogni azione è determinata da ciò che viene rilevato in quel momento.
Il secondo ciclo presente nell'algoritmo definisce il numero di step necessari a far convergere l'algoritmo verso un nodo terminale. L'episodio di apprendimento ha origine da un nodo iniziale e si conclude con il raggiungimento di un nodo obiettivo ed entro comunque un numero limitato di iterazioni stabilite dallo sviluppatore. Questo secondo loop nel nostro caso viene realizzato dallo stesso ciclo di controllo retroazionato descritto nello schema a blocchi precedentemente illustrato.

L'algoritmo Q-Learning on-line modificato sarà quindi il seguente:

1. Imposta il parametro alfa, gamma, epslon

2. Inizializza la matrice Q con valori da "file q.csv"

3. Inizializza il confort model con valori da file "limiti.csv"

4. Leggi lo stato attuale ***s***

5. Ripeti

*Se (**s** ∈(not confort))*

- *Calcola le azioni ammissibili*
- *Seleziona l'azione "**a**" dalle azioni ammissibili per "**s**" con politica Ɛ-greedy*
- *Esegui l'azione "**a**" sull'ambiente*
- *Osserva lo stato raggiunto **s'** ed il reward prodotto **r***
- *Estrai dalla matrice Q il massimo valore atteso da s: Max [Q (s', all actions)]*
- *Aggiorna Q: Q (s, a) =r + Gamma * Max [Q (s', all actions)]*
- *s <- s'*

else

- *Imposta timer(s)*

End

Tutti gli algoritmi di tipo Time Difference TD e quindi anche Q-Learning, aggiornano la propria conoscenza osservando l'evoluzione del sistema tra due step temporali successivi. La formulazione classica della funzione di aggiornamento come abbiamo già visto è la seguente:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$

In questo tipo di formulazione le informazioni di cui necessita l'algoritmo sono:

1. Lo stato attuale al tempo t: **s_t**, l'azione scelta nello step attuale: **a_t**, il valore attuale della coppia stato-azione nella matrice Q: **$Q(s_t,a_t)$**.
2. Lo stato futuro o al tempo t+1: **s_{t+1}**, il reguard prodotto allo stato successivo s_{t+1}: **r_{t+1}** ed il valore di Q che genera il massimo valore atteso nello stato s_{t+1}.

In ambiente simulato è possibile considerare in modo deterministico come evolve il sistema da uno stato all'altro ed è possibile anche stabilire in modo predittivo quale sarà il reward ottenibile dallo stato successivo.
Per adoperare questo tipo di approccio nel nostro caso pratico, dovremmo disporre di una formulazione del modello fisico dell'ambente in modo da predire, in funzione della posizione del sole e della caratteristica architettonica dell'ambiente, lo stato successivo ed il reward ricavato.
Tecnica certamente perseguibile poichè sono disponibili in letteratura modelli abbastanza precisi per determinare l'illuminamento solare nell'arco di una giornata. È da dire però che questo modo di procedere pone delle forti limitazioni nell'applicazione del sistema, sia perché si lega a dati previsionali e non reali e sia perché necessita di una fase di progettazione e di studio del modello previsionale.
Per svincolarci da queste restrizioni e garantire la maggiore versatilità ed adattabilità del sistema si è deciso di considerate la seguente formulazione per la legge di aggiornamento dell'intelligenza:

$$Q(s_{t-1}, a_{t-1}) \leftarrow Q(s_{t-1}, a_{t-1}) + \alpha[r_t + \gamma \max_a Q(s_t, a) - Q(s_{t-1}, a_{t-1})]$$

Rispetto alla precedente formulazione l'aggiornamento dell'intelligenza avviene sempre nello stato di esecuzione attuale ma si riferisce alla coppia stato-azione riferita al passo precedente. Posticipare l'aggiornamento di un passo ci consente di avere i seguenti vantaggi:

1. Non occorre definire modelli di predizione per il calcolo dello stato successivo ed il reward ottenuto;
2. Si lavora su dati reali e misurabili e non su dati desunti.

Algoritmo di controllo

Diversamente da altri algoritmi come ad esempio SARSA, dove la politica di scelta delle azioni viene aggiorna usando il valore Q dello stato successivo e l'azione della politica attuale, Q_learning usa una politica di apprendimento di tipo off-policy, ovvero la politica di scelta delle azioni da intraprendere è diversa dalla politica di aggiornamento della matrice Q. Questa separazione delle due funzioni: di aggiornamento e di ricerca della policy ottimale, ci permette di considerare separatamente ed in due flussi separati le due funzionalità. L'algoritmo di controllo è quel blocco del sistema che si occupa della scelta della prossima azione, bilanciando esplorazione con apprendimento.
La policy che si vuole adottare e di tipo Ɛ-greedy modificata. Il pseudocodice che descrive l'algoritmo di controllo è il seguente:

1. Imposta il parametro epslon0, definito dal progettista.

2. Calcola il parametro epsoln come epsoln0 - (days/20)

*3. Calcola l'insieme delle azioni ammissibili **A** dallo stato attuale*

*3. Calcola un numero random **x** compreso tra 0 ed 1*

4. Se x < epsoln

Esplorazione: Seleziona l'azione a in modo random dall'insieme delle azioni ammissibili A.

altrimenti

Conoscenza: Seleziona l'azione a dalla matrice Q.

Il parametro Ɛ ha un valore iniziale fissato dal progettista, tipicamente tra 0,3 a 0,8, dove il valore 0,3 indica che il 30% delle azioni sono scelte verso l'esplorazione ed il rimanente 70% dall'esperienza. Pertanto fissando inizialmente Ɛ = 0,5 ed decrementandolo ogni giorno di 0,05, dopo dieci giorni, la scelta dell'algoritmo di controllo si baserà soltanto sulla conoscenza acquisita in Q.
Questo ci consente di limitare il periodo di addestramento dell'algoritmo e di evitare azioni non desiderate da parte del controllore.
La fase di addestramento non è però one step, ovvero se interviene un "reclamo" da parte di un utente, il valore di Ɛ viene ripristinato al valore iniziale. Questo perché un reclamo sta ad indicare che ciò che era stato appreso dall'algoritmo non è più valido ed occorre una nuova fase di addestramento.

Sviluppo e Test

Le fasi che tipicamente prevedono lo studio e lo sviluppo di un nuovo sistema, richiedono tra l'altro una fase di verifica ed analisi in ambiente simulato. Ancor di più se parliamo di sistemi di intelligenza artificiale, in quest'ultimi è possibile, ed in molti casi auspicabile, effettuare una fase di addestramento dell'intelligenza in ambiente simulato per poter poi impiegare tale conoscenza in ambiente reale, evitando che il sistema evolva liberamente verso stati indesiderati ed anche potenzialmente pericolosi. Nel nostro caso non si evidenziano particolari criticità che inducono ad adottare un approccio di questo tipo, al contrario si vuole enfatizzare la caratteristica del sistema di apprendere un modello dell'ambiente con l'iterazione diretta. È tuttavia possibile effettuare i test, se

non si dispone di tutta la strumentazione per farlo in ambiente reale, utilizzando opportuni software di simulazione specifici per la progettazione di sistemi di illuminotecnica per gli edifici. I software appena citati, richiedono l'inserimento di alcuni parametri relativi all'architettura dell'ambiente, tra cui: la disposizione verso i punti cardinali, la dimensione e la collocazione delle aperture finestrate e dei corpi luminosi presenti. Vista la forte tipizzazione del caso di studio, risulta evidente che un solo test per un particolare ambiente non è sufficiente per validare il modello stesso. Il nostro test in ambiente simulato, si limita pertanto a testare il comportamento dell'algoritmo inviando degli specifici input e verificando gli output prodotti, nonché l'aggiornamento prodotto nella matrice di conoscenza. Si tratta quindi di una verifica step by step dell'algoritmo. È necessario quindi definire anticipatamente la sequenza di input da inviare ed il risultato atteso da parte del sistema. In particolare si tratta di definire quello che viene chiamato in gergo "caso di studio".

Per la realizzazione del nostro algoritmo, l'ambiente di sviluppo, l'ambiente di esecuzione/produzione e l'ambiente di simulazione coincidono con la stessa piattaforma. La piattaforma che si è deciso di utilizzare è NODE-RED, strumento nato per l'IoT che consente di definire logiche applicative complesse attraverso l'uso di un sistema guidato ad eventi. Offre la possibilità di collegare tra loro diversi dispositivi (con eventuali relativi sensori ed attuatori), oltre a API e servizi online per poter realizzare sistemi/scenari altamente integrati.

Il modello event-driven usato e l'esecuzione asincrona delle operazioni di I/O, consentono di sviluppare soluzioni scalabili ed efficienti per l'analisi real-time di flussi di dati, con un forte orientamento all'Internet of Things. Le applicazioni Node-RED sono definite come "flow" (flusso) composti da "blocchi" chiamati "nodi". Ogni nodo ha una parte "nascosta" la quale contiene l'implementazione del processo e dell'elaborazione eseguita dal relativo nodo e una parte "visibile", descritta attraverso codice HTML, per quanto riguarda la parte visuale e di configurazione. Tutti i nodi del programma sono dotati di porte d'input e di output attraverso le quali possono essere instaurate le connessioni tra gli stessi. In Node-RED sono evidenti tutti i concetti tipici della flow-based programming. Tutti i linguaggi di programmazione sono creati facendo riferimento a specifici paradigmi con i quali si stabiliscono lo stile, le modalità, i componenti fondamentali e l'interazione tra questi per la realizzazione del software. Quando parliamo di Flow-Based Programming ci riferiamo ad un paradigma di programmazione.

Questo paradigma stabilisce che un'applicazione sia costituita da una serie di blocchi dei quali dobbiamo conoscerne il tipo di elaborazione eseguita ma non necessariamente la corrispondente implementazione interna; per questo motivo essi possono essere considerati come "black box". Tali blocchi utilizzano una o più porte (sia in ingresso che in uscita) per poter essere collegati tra loro attraverso delle connessioni in modo da costituire una rete nell'ambito della quale comunicano scambiandosi dati sotto forma di messaggi. I blocchi possono essere riutilizzati senza alcun cambiamento interno per poter sviluppare altri programmi che richiedano le medesime elaborazioni in determinati contesti. Abbiamo dunque la possibilità di avere più blocchi che eseguono le medesime elaborazioni nello stesso programma, ciascuno in modo indipendente dagli altri anche in termini di memoria di lavoro e di risorse utilizzate (a meno di casi particolari in cui potrebbero sussistere anche delle risorse condivise).

Il programma non è più una sequenza di istruzioni ma è caratterizzato da un insieme di flussi di dati che vengono scambiati tra i blocchi in maniera completamente asincrona. L'esecuzione viene

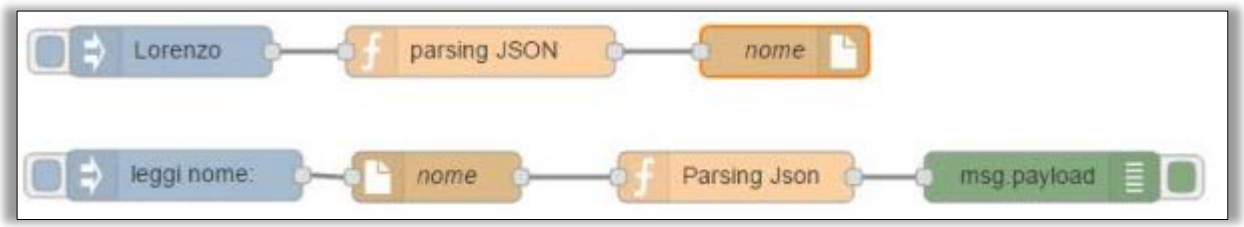
garantita da un software noto come “scheduler” che può o meno sfruttare le caratteristiche della piattaforma in cui si trova per garantire anche l’eventuale parallelismo nell’esecuzione dei processi su più flussi. Il concetto di flussi separati è determinato proprio dal fatto che l’applicazione può avere più ingressi indipendenti attraverso i quali far fluire i dati attraverso i blocchi sfruttando le relative connessioni. Data la sua natura basata sul concetto di rete (blocchi e connessioni), la programmazione a flussi è molto spesso realizzata con tool di tipo visuale tra i quali spicca Node-RED con la sua focalizzazione al mondo dell’Internet of Things. Va sottolineato che i blocchi di ciascun flusso sono comunque caratterizzati da codice da eseguire, per cui lo sviluppo di un’applicazione prevede due livelli: quello visuale e quello implementativo per ciascun blocco.

Conservazione dello stato

Esistono numerose soluzioni per salvare le impostazioni locali, le variabili del sistema o i parametri di configurazione. Molti, infatti, sono i nodi di collegamento verso sistemi di archiviazione on-line o verso database di varia natura. La scelta di una o dell’altra soluzione dipende naturalmente dalla mole, dalla nauta dei dati, dell’hardware a disposizione, dal tempo di risposta del sistema.

Per quanto riguarda la conservazione e/o la gestione di alcune variabili di sistema o di alcuni parametri come le soglie di controllo, abbiamo ritenuto opportuno proporre l’archiviazione su file locali. All’interno dei file locali possono essere conservate le informazioni utili al funzionamento del sistema, che possono essere poi recuperate ad esempio all’avvio dello stesso. Il valore delle variabili di controllo del sistema, conservato all’interno di variabili locali o globali del programma, viene archiviato in memoria locale per poi essere nuovamente recuperato ad ogni riavvio. La scelta dell’utilizzo dei file è guidata anche dal fatto che node-red offre una modalità di gestione estremamente semplice, proponendo due nodi: quello di scrittura e quello di lettura. Le modalità di scrittura sul file possono essere di due tipi: modalità append e modalità overwrite. Nella modalità overwrite ogni messaggio in arrivo al nodo andrà a cancellare il valore precedente ed a sovrascrivere il nuovo valore, mentre nella modalità append ogni messaggio arrivato verrà accodato su una nuova linea all’interno del file. Per poter conservare oggetti JSON su un file, occorrerà prima convertirli nello stesso formato del file. Per i file di testo viene adoperata la funzione JSON.stringify(oggetto).

Di seguito un esempio di un flusso che archivia una coppia variabile:valore all’interno di un file di testo ed un flusso che legge i parametri nel file e li mostra a video.



Dopo aver visto come rendere persistenti dei dati all’interno di un file, vediamo ora come conservare dei dati durante elaborazioni successive del sistema. Sappiamo che in questo genere di sistema,

all’arrivo di un evento (messaggio) si può innescare un flusso di elaborazione, che si conclude quando non vi è più nessun messaggio che circola tra i nodi. In realtà il programma non termina mai, rimane in uno stato di standby fintanto che non arriva un nuovo messaggio. Se vogliamo mantenere memoria di alcuni valori derivanti da elaborazioni precedenti dobbiamo poggiarci a delle variabili dette di contesto (o di ambiente). Il nome deriva dal fatto che il valore assunto da queste variabili è visibile e modificabile solo in porzioni ben delineate del programma. Per questo esistono le variabili che hanno come campo di visibilità solo un nodo, variabili che invece sono interrogabili e modificabili da tutti i nodi presenti in un flusso o più in generale nella stessa scheda e variabili che sono visibili in tutto il sistema.

All’interno di un nodo è possibile salvare il valore di alcune variabili per poter essere utilizzate in elaborazioni successive, come ad esempio è possibile conoscere quanti elaborazioni ha eseguito un nodo aggiornando all’arrivo di ogni messaggio una variabile contatore.

var count = context.get('count')||0;

count += 1;

context.set('count',count);

msg.count = count;

Per accedere alle variabili di contesto di un nodo si usano i metodi: context.get('nomevariabe') per leggere il valore e context.set('nomevariabile',valore) per scrivere la variabile.

Se il valore della variabile contatore che abbiamo visto nel paragrafo precedente, lo vogliamo rendere disponibile anche per altri nodi del flusso o della finestra allora occorrerà utilizzare l’oggetto flow al posto dell’oggetto context. La sintassi rimane invariata, l’oggetto flow avrà i metodi get e set per leggere e scrivere su una variabile.

var count = flow.get('count')||0;

Una ulteriore estenzione della vesibilità di una variabile è data attraverso l’uso di variabili globali, visibili da più programmi o più flussi. La sintassi è la seguente:

Le variabili globali del sistema sono definite dalla sintassi:

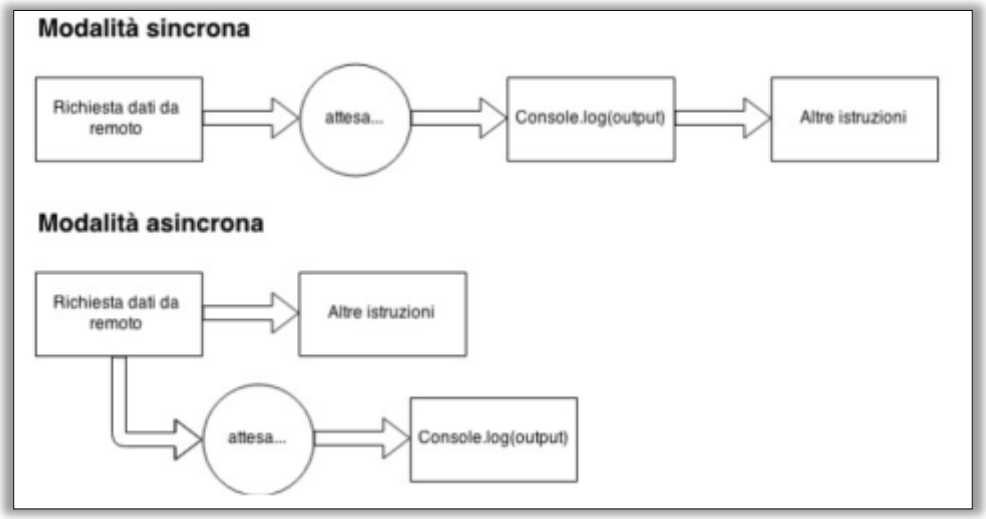
var count = global.get('count')||0;

Modello di concorrenza non-blocking Asincrono

Il modello su cui si basa il sistema è quello event-driven, anziché il classico modello a processi o thread concorrenti. Ciò significa che si eseguono azioni solo al verificarsi di un evento, rimane in sleep fino alla notifica del completamento quindi si riattiva per eseguire istruzioni contenute in una funzione callback. Questo meccanismo asincrono è abbastanza noto per chi usa le chiamate a servizi remoti (microservices): ad esempio, quando un’istruzione richiede un valore ad un server remoto, come nel caso di una chiamata Ajax, l’esecuzione del codice non si ferma in attesa del risultato ma prosegue, “saltando” l’istruzione che effettua la chiamata. Tale istruzione rimane in uno stato di “limbo”, perché è stata eseguita ma non completata: nel frattempo vengono eseguite le istruzioni

successive. Quando la chiamata remota viene completata il controllo dell'esecuzione torna all'istruzione che era stata sospesa.

Modello di concorrenza non-blocking Asincrono



Nella programmazione JavaScript si fa spesso ricorso ad un modello di programmazione concorrente in base al quale possiamo pensare che diverse attività possano avvenire virtualmente in parallelo e possano comunicare tra di loro in maniera asincrona. Pensiamo, ad esempio, al verificarsi di eventi: questi si possono verificare in maniera indipendente dall'esecuzione del flusso principale del nostro programma e, virtualmente, possono verificarsi più eventi contemporaneamente. Se abbiamo previsto gestori di eventi (flussi di controllo) per più eventi, ci aspettiamo che questi vengano eseguiti immediatamente al verificarsi del relativo evento. In realtà le cose non stanno proprio così. Il modello di concorrenza di JavaScript è diverso da quello di altri linguaggi come ad esempio C o Java. Mentre infatti nei linguaggi di programmazione che supportano la concorrenza una porzione di codice di un thread può essere interrotta per mandare avanti l'esecuzione di un altro thread, in JavaScript tutto avviene in un unico thread.

Il modello di concorrenza in base al quale abbiamo l'illusione che più thread siano in esecuzione è quello dell'event loop: ogni evento inserisce un messaggio in una coda che viene elaborata sequenzialmente dal runtime di JavaScript in un ciclo infinito. In pratica, un engine JavaScript non fa altro che verificare la presenza di messaggi nella coda ed eseguire il codice dell'eventuale gestore (flusso o regola) per passare poi al messaggio successivo. È importante aver chiaro che il codice eseguito tra un messaggio ed il successivo viene eseguito senza interruzioni. Qualsiasi evento che si verifica durante l'esecuzione di un ciclo dell'event loop non può interromperlo.

Comprendere il modello di concorrenza su cui si basa JavaScript è importante per capire il motivo di certi comportamenti e per poter scrivere codice efficiente. Quindi due eventi non saranno mai trattati esattamente nello stesso momento. Come gli eventi arrivano essi vengono aggiunti a una coda di eventi e gestite in un ciclo di base molto veloce. Tuttavia una volta che un flusso viene avviato non

vi è alcuna garanzia che verrà eseguito fino al suo completamento. Se per esempio abbiamo due flussi composti dai seguenti blocchi attivati da due eventi diversi: 1-2-3-4 e ABCD

l'arrivo dei due ingressi potrebbe causare la sequenza di esecuzione: 1-A-2-B-3-C-4-D.

È molto difficile ottenere la sequenza 1-2-3-4-ABCD - come se gli eventi non fossero concorrenti. Dopo aver compreso il meccanismo dell'event loop occorre soffermarci su un'altra importante considerazione: in JavaScript le operazioni di input e output non sono bloccanti, differentemente da quanto accade in altri linguaggi.

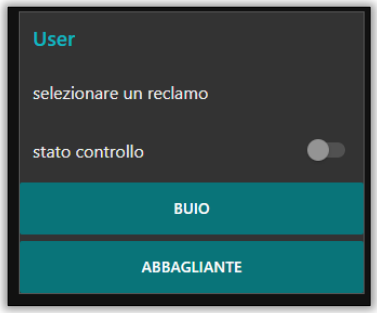
Il principio è abbastanza semplice: tutto ciò che riguarda l'esecuzione di codice JavaScript puro viene eseguito all'interno di ciascun ciclo dell'event loop e non è interrompibile; ogni interazione che coinvolge l'ambiente di esecuzione viene eseguita con l'approccio ad eventi tramite la coda di messaggi. Quindi richieste come il caricamento di dati dal server o la scrittura su un database sono operazioni asincrone il cui completamento inserirà una richiesta di esecuzione della relativa funzione di callback nella coda dei messaggi.

Questo aspetto è uno dei punti di forza di JavaScript che consente di ottenere elevate prestazioni nella gestione di più richieste contemporanee pur mantenendo un unico thread di elaborazione, come avviene, ad esempio, nella gestione delle richieste HTTP in node.js.

Blocco HMI: Flusso di gestione reclami utenti

L'elemento innovativo che distingue questo progetto dai precedenti lavori è l'utilizzo di un nuovo approccio detto "humancentric" per la risoluzione del problema di controllo. Questo approccio, basato per l'appunto sull'interazione diretta con l'utente come già accennato, è reso possibile attraverso l'inserimento sull'anello di retroazione del sistema di controllo del blocco chiamato Human Machine Interface: HMI.
Il blocco HMI ha la funzione d'interfacciamento del sistema di controllo verso gli utenti ed implementa le seguenti funzioni:

- presenta una lista di percezioni che l'utente può esprimere;
- fornisce meccanismi di override del sistema di controllo in caso di emergenza, ovvero meccanismi di disattivazione del controllo e di attivazione manuale.



Una prima implementazione che è stata realizzata è raffigurata nella figura precedente, dove nella sezione in alto viene mostrata l'informazione di ausilio all'utente riguardante lo stato in cui si trova

il sistema insieme alle informazioni sui comandi che vengono azionati. Nella stessa sezione troviamo un interruttore che consente di impostare lo stato di funzionamento del sistema.

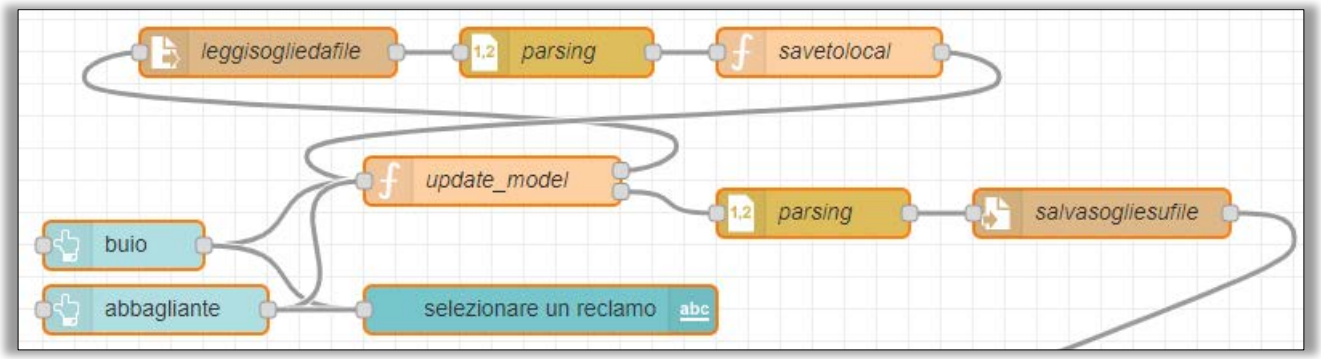
L'interfaccia consente di selezionare due posizioni che rappresentano il seguente stato del sistema:

- **sistema in controllo**. In questa modalità il controllo è lasciato al sistema automatico;
- **sistema non in controllo**. Questa modalità serve a disattivare l'algoritmo di controllo e consentire una gestione manuale dei dispositivi. Utile in caso di emergenza o di manutenzione/pulizia degli apparati.

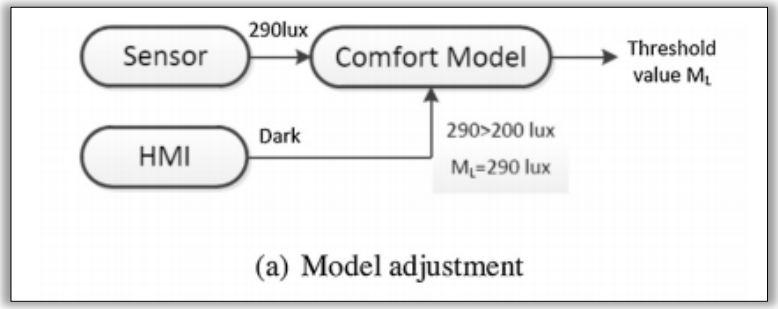
In basso, nella stessa finestra, sono disponibili due pulsanti che identificano i due tipi di **reclamo/percezione** che sono concessi all'utente:

- **Buio**. L'utente cliccando su questo pulsante invia un reclamo al decisore che provvederà, secondo la logica stabilita dal blocco "confort model", a ridefinire i parametri del sistema verso un settaggio che garantisca un miglior apporto di luce;
- **Abbagliante**. Selezionando questo bottone si invia un reclamo per indurre il sistema a trovare un settaggio dei parametri che riducano l'apporto di luce.

Il flusso che implementa questo meccanismo è il seguente:



In Azzurro troviamo i nodi di comando "Buio" ed "Abbagliante" rappresentati nella dashboard attraverso i bottoni corrispondenti. Ancora in blu troviamo il campo di testo "selezionare un reclamo" che serve a visualizzare nell'interfaccia HMI il comando selezionato dall'utente. L'evento così generato dai due pulsanti in ingresso al flusso confluisce nel nodo funzione "update_model". In questa prima release si utilizza un approccio semplificato per approcciare il "confort-model". Il modello di aggiornamento del modello di confort segue il seguente schema:



Descrizione: il questo esempio i sensori di luminosità presenti nell'ambiente leggono un valore di luminosità pari a 290lux. Supponiamo che il valore di soglia corrispondente al limite inferiore "ML" presente a sistema sia di 200lux, pertanto in queste condizioni il modello di confort genererà un segnale/evento di "confort". Se nelle stesse condizioni interviene l'utente con un reclamo di "Buio", allora l'"update_model" non farà altro che aggiornare il setpoint ML del sistema con il nuovo limite. In questo esempio il valore di ML viene aggiornato al valore 290lux.

L'algoritmo che svolge questa funzione è il seguente:

```
var ml =flow.get('ML');
var mh = flow.get('MH');
var lx = flow.get('lux');
flow.set('epsoln',0.5);
if(ml!==undefined && ml!==null){
    if (msg.payload=='buio' && lx>ml){
        flow.set('ML',lx);
    }
    if(msg.payload=='abbagliante' && lx<mh){
        flow.set('MH',lx);
    }
    return[null,{payload:[flow.get('ML'), flow.get('MH')]}];
}else{
    return[msg,null];
}
```

Le prime tre istruzioni dell'algoritmo prelevano il valore ML, MH e LUX dalle variabili d'ambiente precedentemente valorizzate. Nel caso in cui le variabili ML e MH risultassero indefinite allora si procede alla loro acquisizione leggendole dal file di persistenza presente nel sistema.

Notiamo in questo caso che il nodo presenta due "uscite", ovvero definisce una funzione che produce due valori. In questo caso l'istruzione "return[msg,null]" produrrà un valore "null" sull'uscita 2 mentre replicherà il messaggio al suo ingresso sull'uscita 1. Se sull'uscita 1 verrà generato un messaggio diverso da null, questo azionerà la lettura del file "soglie.csv" e l'inizializzazione delle variabili ML ed MH.

Eseguita la lettura dei limiti da file ed effettuato l'aggiornamento dei limiti secondo i reclami impartiti dall'utilizzatore, il passo successivo è salvare le informazioni modificate su file garantendone la persistenza. Pertanto nello stesso flusso viene inserito il nodo "**salvasogliesufile**" che provvederà a salvare i valori aggiornati nel file locale "soglie.csv".

Blocco HMI: Miglioramento Paretiano

La seconda implementazione del blocco Confort model fa ricorso ad una analisi statistica effettuata sulla serie di dati che il sistema riceve attraverso il feedback degli utenti. Questa metodologia va sotto il nome di **miglioramento Paretiano** dal nome del suo ideatore, un economista Italiano Vilfredo Pareto.

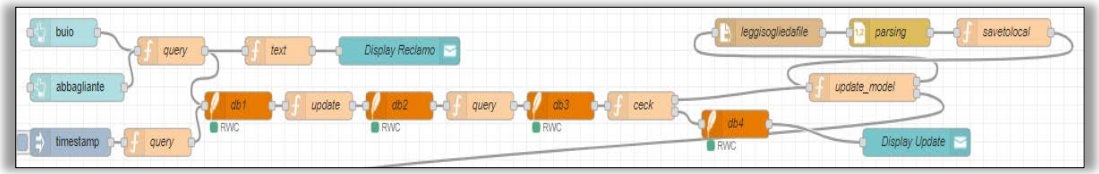
Efficienza paretiana (o Pareto-efficienza)

Una situazione S è **Pareto-efficiente** (o **efficiente in senso paretiano**) se non è possibile accrescere il benessere di alcuno dei soggetti coinvolti, se non riducendo il benessere di qualcun altro di loro.

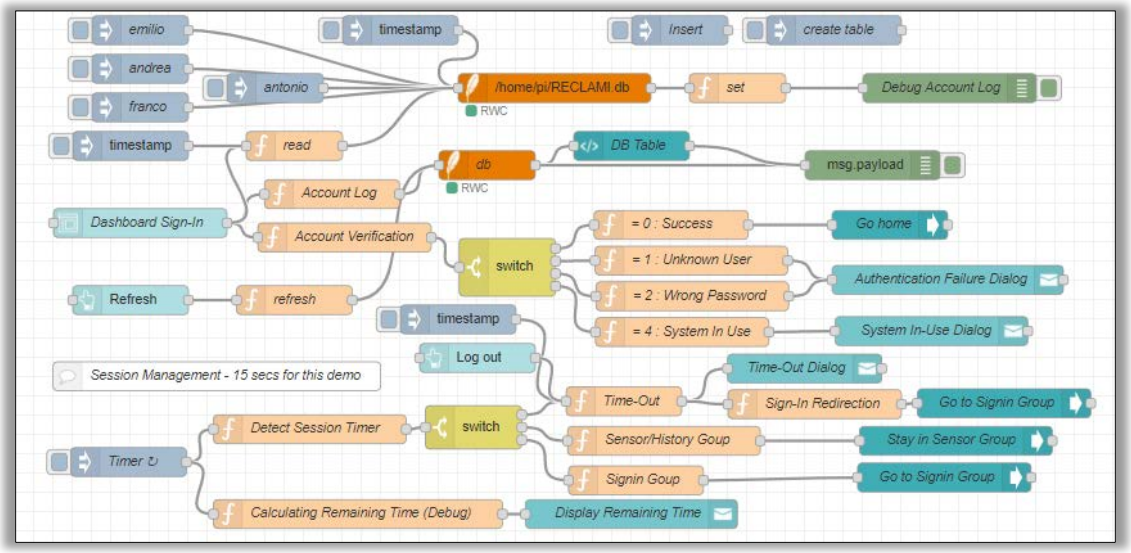
Una situazione S′ è un **miglioramento paretiano** rispetto alla situazione S, se è possibile trasformare S in S′, il benessere di *almeno uno* dei soggetti coinvolti è maggiore in S′ che in S e *nessuno* ha minor benessere in S′ che in S. Un miglioramento paretiano è **debole** se nella nuova situazione il benessere di qualcuno dei soggetti coinvolti è lo stesso di prima (ma non è escluso che, invece, tutti godano di un miglioramento), è **forte** se v'è, invece, un incremento di benessere per *tutti* i soggetti coinvolti.

Possiamo, pertanto, riformulare la definizione precedente come segue: una situazione S è Pareto-efficiente se non esistono miglioramenti paretiani deboli rispetto ad essa.

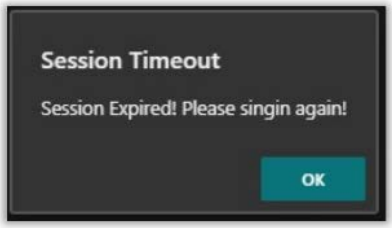
Il classificatore è stato realizzato attraverso il seguente flusso:

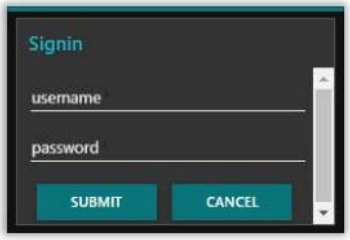


Innanzitutto è stato necessario gestire il login di ogni utente ed assegnargli una sessione di lavoro, questa funzionalità è stata implementata attraverso il seguente programma:



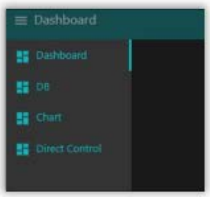
Il Flusso Session Management verifica periodicamente la variabile di sessione per ogni utente che ha effettuato il login, se la sessione è scaduta visualizza una notifica e reindirizza l'output verso la pagina di login :

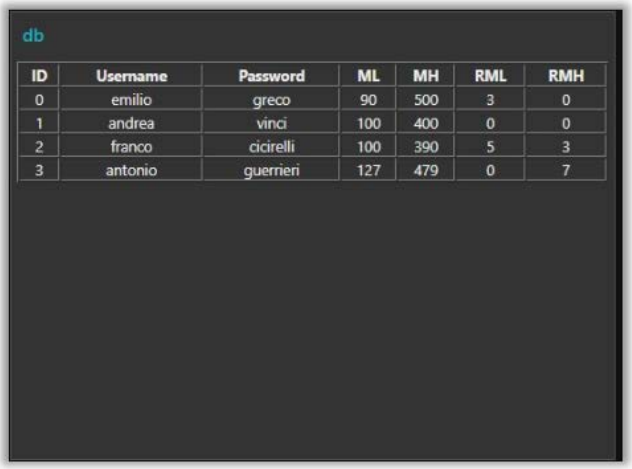




Il flusso "Dashboard Sign-in" inizializza una tabella HTML che espone i dati contenuti nel database e provvede a verificare le credenziali degli accessi. Al termine di ogni login l'utente viene reindirizzato alla home page.

Per visualizzare il numero dei reclami ed i setpoint per utente, dal menu principale si arriva al tab DB:





ID	Username	Password	ML	MH	RML	RMH
0	emilio	greco	90	500	3	0
1	andrea	vinci	100	400	0	0
2	franco	cicirelli	100	390	5	3
3	antonio	guerrieri	127	479	0	7

I pulsanti "buio" ed "abbagliante" in input al flusso ricevono i reclami dagli utenti e innescano una notifica immediata di presa in carico da parte del sistema attraverso un allert "Display Reclamo". La prima operazione che esegue l'algoritmo è l'update dei contatori presenti nel DB ed eventualmente un aggiornamento del valore del limite associato al singolo utente.

Se ad esempio la condizione iniziale dell'operatore "emilio" è la seguente:

ID	Username	Password	ML	MH	RML	RMH
0	emilio	greco	90	500	3	0

A seguito di un reclamo di buio se il livello di lux è minore del valore di soglia precedentemente salvato viene incrementato solo il contatore RML, altrimenti viene aggiornato anche il valore ML.

ID	Username	Password	ML	MH	RML	RMH
0	emilio	greco	151.77492	500	5	0

Successivamente ad ogni Update del DB, viene eseguita una verifica se vi sono le condizioni per poter effettuare un aggiornamento dei limiti di controllo del sistema. Il blocco “ceck” riceve come risultato di una query tutte le tuple del DB, verifica se vi è almeno metà degli utenti che ha eseguito un reclamo di “buio” o di “abbagliante” e se complessivamente il numero di reclami per ogni classe è maggiore o uguale a 10 procede con l’aggiornamento dei limiti.

db

ID	Username	Password	ML	MH	RML	RMH
0	emilio	greco	90	500	3	0
1	andrea	vinci	100	400	0	0
2	franco	cicirelli	100	390	5	3
3	antonio	guerrieri	127	479	0	7

In questo esempio abbiamo 2 su 4 utenti che hanno espresso almeno un reclamo su “buio” ma il numero di reclami non è sufficiente per procedere all’aggiornamento di ML, mentre per i reclami su “abbagliante” è soddisfatta sia la condizione sul numero di utenti che sul numero di reclami. In questo caso si procede ad aggiornare il valore di MH.

In questo esempio il valore di MH scelto è il minimo dei valori MH espressi dagli utenti che hanno notificato dei reclami di tipo “abbagliante”, mentre non vengono considerati gli altri utenti. Questa scelta esprime il concetto di eseguire un “miglioramento paretiano debole”, pertanto il nuovo valore scelto sarà MH=390. Fortunatamente il valore scelto rappresenta anche un miglioramento paretiano “forte” poichè se viene soddisfatto l’utente “franco” di conseguenza verranno soddisfatto tutti gli altri utenti.

Ipotizziamo che gli utenti “emilio” e “franco” raggiungono un numero di reclami pari a 10, in questo caso l’aggiornamento della soglia del sistema al valore ML=100 (il valore massimo di tutti i reclami), comporta un miglioramento paretiano debole e non forte perchè l’utente “antonio” verrebbe escluso da questa scelta.

Oltre all’aggiornamento dei limiti di controllo del sistema, il blocco Confort model provvede al reset dei contatori (RML, RMH) per ogni Update ed al riavvio della fase di addestramento per la matrice

Q.

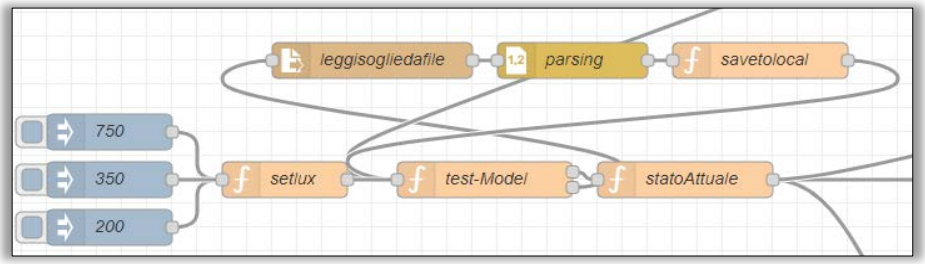
Blocco Confort Model: Flusso di gestione dello stato

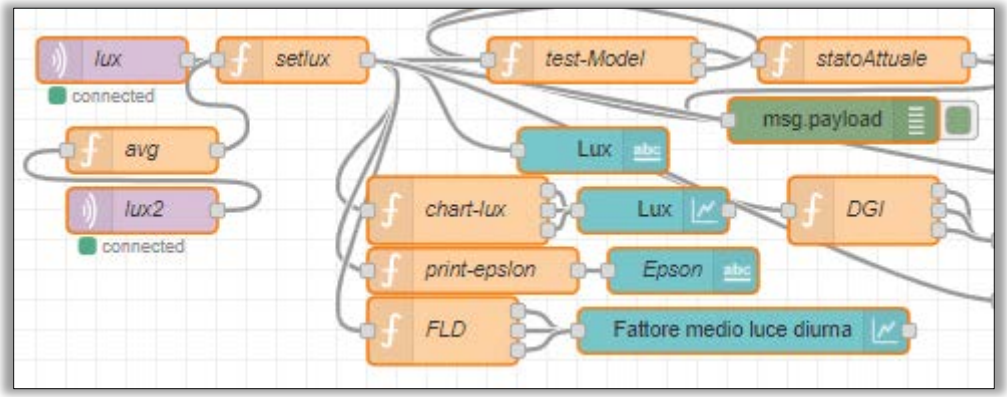
In ingresso al blocco Confort Model, primo nodo del flusso di controllo principale, intervengono due tipi di segnali o di feedback da parte del sistema di controllo:

1. Il reclamo che arriva dai sensori.
2. Il reclamo che arriva dall'utente.

Nel paragrafo precedente abbiamo visto come vengono presentati, attraverso l'interfaccia HMI, i tipi di reclamo e come questi vengono gestiti attraverso il flusso di gestione dei reclami. Un reclamo utente innesca necessariemente un aggiornamneto dei limiti che definiscono il modello di confort ed di conseguenza anche una nuova fase di apprendimento dell'intelligenza che deve ridefinire il suo comportamento in funzione dei nuovi limiti. Diverso è invece il comportamento per la gestione dei segnali provenienti dai sensori di luminosità. In linea generale il confort model interviene per verificare se il valore di lux letto dai sensori ricada all'interno dei limiti di accettabilità, se ciò non avviene si produrrà un evento che induce il decisore a cercare un'azione che riporti il sistema in uno stato di confrot. Questo meccanismo non richiede una continua fase di addestramento dell'intelligenza. Una volta stabilito un albero delle decisioni, ricavato da una fase iniziale di Learning, il controllore deve procedere utilizzando la sua base di conoscenza nella scelta delle azioni da intraprendere.

Il periodo di apprendimento dell'intelligenza viene gestito attraverso il parametro "epsoln" già visto nella descrizione dell'algoritmo di Q-Learning. Questo parametro descrive in che percentuale il decisore sceglie le azioni, ovvero se prelevarle dall'insieme delle azioni ammissibili per il dato stato, oppure prelevare l'azione dalla base di conoscenza. In altre parole descrive la politica di esplorazione dell'intelligenza. Questo valore è stato concepito in modo che sia massimo nel periodo in cui l'intelligenza ha necessità di apprendere e decresca con il tempo in modo da privilegiare la conoscenza acquisita invece che l'esplorazione di nuove soluzioni. Per questo motivo nella funzione che realizza il nodo "update_model" troviamo l'istruzione "**flow.set('epsoln',0.5)**". Questo comando di reset non lo troviamo invece del nodo di acquisizione dei segnali di luminosità, per i motivi che sono stati pocanzi specificati. Il flusso per la generazione dello stato attuale del sistema viene descritto di seguito:





Prima del nodo confort model (rinominato "test-model") , troviamo il nodo "setlux", che funge da nodo filtro per i dati provenienti dai sensori. In questo esempio in ingresso al flusso troviamo tre nodi usati in fase di test per iniettare nel flusso tre valori di lux provenienti da una sola sorgente. Il nodo setlux è un nodo che funge da filtro per i dati che arrivano, al suo ingresso possono confluire più nodi di acquisizione dati interfacciati attraverso opportuni nodi socket mqtt. Se i sensori sono più di uno, il nodo "setlux" dovrà implementare un meccanismo di aggregazione dei segnali di ingresso. In ogni caso deve garantire meccanismi di gestione di anomalie e di validazione dei segnali, intervenendo ad esempio se quando rileva un'assenza di segnale trascorso un determinato lasso di tempo o scartando delle misure che hanno una elevata deviazione standard o scarto quadratico medio rispetto ai valori attesi.

In fase di test, in ingresso al sistema inseriamo tre nodi di input (nodi in blu nello schema precedente) che iniettano nel flusso tre valori di lux che rappresentano rispettivamente: lo stato di abbagliamento, lo stato di confort e lo stato di buio. Il nodo "setlux" in questo caso andrà a salvare nella variabile d'ambiente LUX il valore di luminosità rilevato che verrà successivamente utilizzato in vari punti del programma. Nel caso in cui in ingresso abbiamo più sensori di luminosità, esso provvederà a ricavare una media dei valori letti.

Nell'implementazione più semplice del modello di confort, che si basa semplicemente sul confronto del valore di lux ricavato dal nodo precedente con i valori di soglia, la funzione che implementa questo modello eseguirà un semplice confronto e restituirà l'esito con un messaggio di testo del tipo "confort", "buio", "abbagliante". Anche in questo caso, la funzione prevede due uscite di cui una in auto anello. L'uscita che porta su l'auto anello serve qualora i valori di soglia non sono stati ancora definiti o letti dal file.

```
var ml = flow.get('ML');
var mh = flow.get('MH');
var lx = parseInt(flow.get('lux'));
if (ml!==undefined && ml!==null){
    if(ml<lx<mh)
        msg.payload= 'confort';
    else if(lx<=ml)
        msg.payload= 'buio';
    else if(lx>=mh)
        msg.payload='abbagliante';
    return [msg,null];
    }else {
        msg.payload='setlimits';
        return [null,msg];
    }
```

Da notare che in questa implementazione i valori di soglia non sono compresi nella zona di confort, questa scelta è determinata dalla logica con cui è stato realizzato l'algoritmo di gestione dei reclami utente. In quest'ultimo, il valore di soglia viene aggiornato con il valore attuale di LUX. Come conseguenza di questo setup si vuole produrre un evento che dia inizio ad una fase di Learning del sistema. Pertanto, definendo come zona di confort**: ml<=lx<=mh**, un evento che proviene dal nodo HMI non produrrà o un messaggio di disconfort e conseguentemente non si procederebbe alla fase di Learning. Ecco spiegata la necessità di dovere definire la zona di confort come: **ml<lx<mh.**

In questa prima fase di elaborazione dell'informazione di feedback, siamo giunti ad ***una classificazione*** di un solo parametro che definisce la rappresentazione di stato del sistema. Gli altri elementi che concorrono a definizione dello stato ricordiamo essere:

- numero di luci accese;
- altezza di apertura della tapparella;
- inclinazione delle doghe della tapparella.

Queste informazioni non vengono acquisite nel flusso principale di controllo del sistema ma vengono gestiti da flussi secondari. L'architettura del sistema, come già specificato in precedenza è di tipo event driven, pertanto devono essere implementati meccanismi che vadano a monitorare eventuali variazioni dell'ambiente per poi renderli disponibili alle altre parti del sistema per l'elaborazione.

Così come visto per il nodo "setlux" i nodi di acquisizione dati devono implementare maccanismi capaci di rilevare anomalie di funzionamento e di filtro sui dati ricevuti. In questo primo prototipo non ci preoccupiamo di questi aspetti e li rimandiamo ad una successiva implementazione.

Supponiamo in questa fase di elaborazione che tutti i dati siano disponibili e validati. Per poter ricostruire quindi l'intera informazione sullo stato, per come è stato definito dal modello MDP, occorre estrarre le informazioni dei dispositivi leggendoli dalle variabili d'ambiente. Variabili che vengono inizializzate e mantenute costantemente aggiornate dai flussi di acquisizione

precedentemente introdotti. Il nodo funzione che provvede alla definizione dello stato è descritto in dettaglio dal seguente codice:

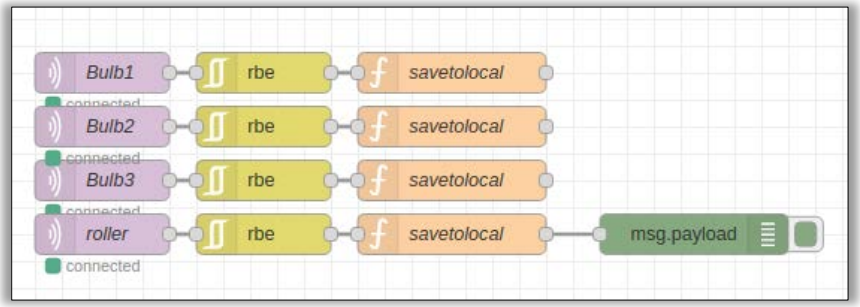
```
/*---------stato foramato da:--------------
2-50-30&c
numero luci accese,
altezza-inclinazione tapparella
reclamo */
var confort=msg.payload;
var cont=0;
if(flow.get('b1')===true)
    cont++;
if(flow.get('b2')===true)
    cont++;
if(flow.get('b3')===true)
    cont++;
// stato tapparella
var roller = parseInt(flow.get('roller'));
var posizione=flow.get('posizione');
//generiamo il valore dello stato
var stato={"count":cont,"pos":posizione.pos,"angolo":posizione.angolo,"confort":confort};
flow.set('statoat',stato);//itilizzata nel nodo finale
msg.payload=stato;
if (flow.get('stato')===undefined){
    flow.set('stato',stato);
}
msg.topic = "stato";
return msg;
```

Nel ricostruire l'informazione di rappresentazione dello stato, come ad es: "2-50-30-c", ricaviamo il primo parametro conteggiando il valore delle variabili d'ambiente "b1","b2" e "b3", il secondo ricavandolo dalla variabile "roller" ed il terzo dalla variabile "posizione".

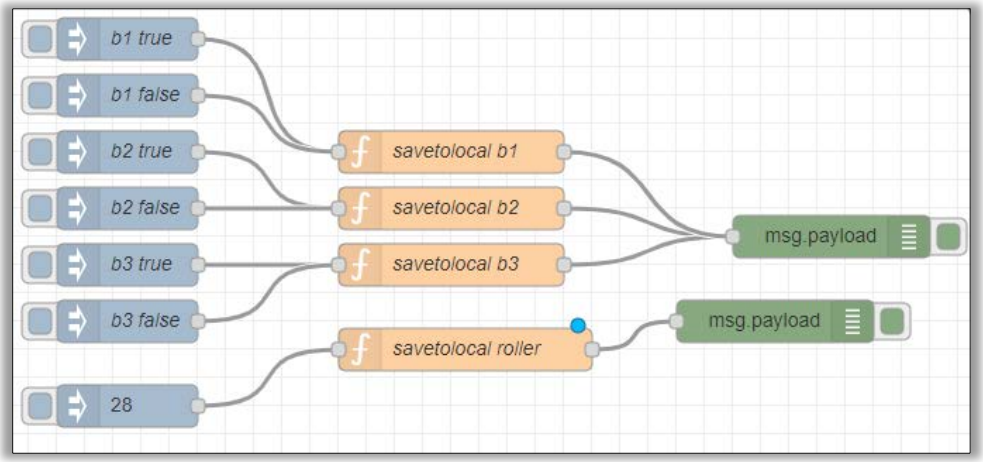
All'interno di questo nodo, o in questa fase di elaborazione, è possibile procedere all'archiviazione ed all'aggiornamento dello stato attuale. Questa informazione viene inserita all'interno della variabile d'ambiente chiamata "statoat". Variabile che verrà successivamente confrontata con lo stato precedente per determinare il reward del sistema. L'informazione dello stato precedente viene archiviata nella variabile "stato" che viene inizializzata in questo nodo con in valore dello stato attuale.

Blocco Confort Model: Flussi di acquisizione dati

I flussi di acquisizione dei dati sono costituiti essenzialmente da due semplici nodi, una porta socket mqtt ed un nodo funzione. Il nodo funzione provvede alla validazione delle misure ed alla loro archiviazione all'interno di variabili d'ambiente. I valori archiviati all'interno delle variabili d'ambiente sono successivamente utilizzate dal flusso principale del programma per le successive elaborazioni.



In fase di test il nodo mqtt socket si sostituisce con nodi che iniettano nel flusso i valori voluti, così come viene riportato nell'esempio seguente:



Se supponiamo per il momento di non implementare meccanismi di rilevazione di errori o filtri sui dati ricevuti, possiamo semplificare lo sviluppo dei nodi funzione "savetolocal bx" in quanto eseguono un semplice funzione di set su una variabile d'ambiente. Una trattazione particolare va invece fatta per il nodo funzione "savetolocal roller". Questo nodo ha l'obiettivo di catturare sia le azioni che vengono impartite dallo stesso controllore che quelle che vengono impartite in modo manuale direttamente dall'utente. Le informazioni che il nodo dovrà ricavare sono: L'altezza della tapparella, l'inclinazione delle doghe e la direzione di movimento. Quest'ultima informazione non è necessaria all'algoritmo di controllo per stabilirne lo stato, ma è fondamentale sia nel nodo "savetolocal roller" che nel nodo "attuatore" per riuscire ad elaborare l'inclinazione delle doghe altrimenti non rilevabili.

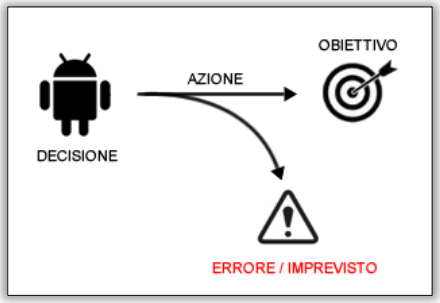
Per ridurre la numerosità degli stati del sistema e rendere l'algoritmo più efficiente si è deciso in fase di progettazione di stabilire un numero definito di posizioni della tapparella. L'elaborazione di questo nodo quindi deve tener conto di questi vincoli progettuali e tradurli in dati per il modello.

L'algoritmo che determina direzione, posizione ed angolo delle tapparelle viene riportato di seguito:

```
roller=Number(msg.payload);
var posprec= flow.get('roller');
flow.set('roller',roller);
var direzione='giu';
if(posprec!==undefined)
  if(posprec<roller){
    flow.set('dir','su');
    direzione='su'
   }else if(posprec> roller)
    flow.set('dir','giu');
var posi=null;
var ang=null;
if(roller===0){posi=0;ang=0;}
else if(roller<=2){posi=0;ang=30;}
else if(roller===3){posi=0;ang=60;}
else if(roller===4){posi=0;ang=90;}
else if(roller<=25){posi=25; if (direzione=='giu') ang=0; else ang=90;}
else if(roller<=27){posi=25;ang=30;}
else if(roller<=28){posi=25;ang=60;}
else if(roller<=29){posi=25;ang=90;}
else if(roller<=50){posi=50; if (direzione=='giu') ang=0; else ang=90;}
else if(roller<=52){posi=50;ang=30;}
else if(roller<=53){posi=50;ang=60;}
else if(roller<=54){posi=50;ang=90;}
else if(roller<=75){posi=75; if (direzione=='giu') ang=0; else ang=90;}
else if(roller<=77){posi=75;ang=30;}
else if(roller<=78){posi=75;ang=60;}
else if(roller<=79){posi=75;ang=90;}
else {posi=100;ang=90;}
var posizione={"pos":posi,"angolo":ang};
flow.set('posizione',posizione);
msg.payload={"pos":posi,"angolo":ang,"direzione":direzione,"posprec":posprec,"posattuale":roller };
//msg.payload=roller;
return msg;
```

Funzionamento stocastico e funzionamento deterministico

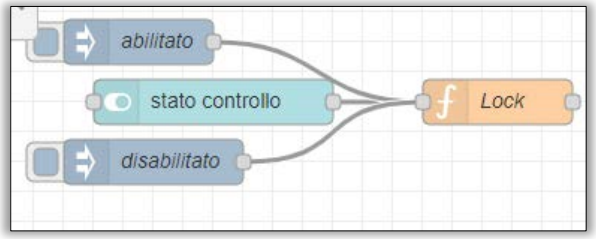
Ricordiamo che l'ambiente in cui opera il nostro decisore è un ambiente dinamico e non deterministico, ovvero si opera in condizioni di incertezza. Quando l'agente razionale prende una decisione, non è detto che gli attuatori robotici eseguano l'azione richiesta dall'intelligenza artificiale, il sistema potrebbe riprodurre per errore un'azione diversa, oppure l'attuazione viene interrotta per una qualsiasi causa non prevista.



La scelta dell'azione migliore da intraprendere in ogni stato, valutando anche i rischi di cadere in errore, vengono catturati dall'algoritmo di controllo durante la fase di learning del sistema ed utilizzate per scelte successive, pertanto se durante una fase di learning il sistema esegue un'azione errata, il reward negativo che ne risulterà, produrrà un valore atteso inferiore rispetto ad altre azioni.

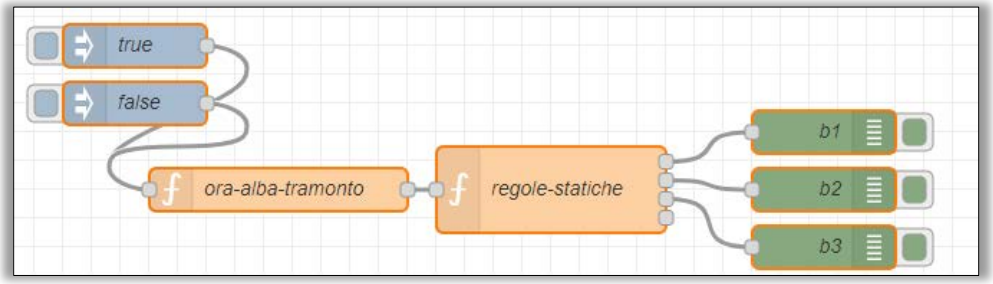
Il modello che verrà ricavato dall'algoritmo sarà frutto di un processo decisionale di Markov non deterministico.

Ci sono tuttavia dei comportamenti del sistema che non vogliamo ricadano in comportamenti stocastici, che non richiedono una fase di apprendimento o che comunque non vogliamo lasciare alla casualità. Un funzionamento deterministico del sistema ad esempio si ha quando si imporre un modello con scelta forzata. Nel nostro caso si impongono due modalità d'uso del sistema: sistema in controllo automatico o sistema in controllo manuale. Scelta demandata all'utilizzatore che interviene sul sistema attraverso l'interfaccia HMI tramite l'interruttore "stato di controllo". Per poter implementare questo meccanismo di funzionamento si fa ricorso al seguente flusso:



Costituito dal nodo "stato controllo" rappresentato da un interruttore presente nella HMI. Interruttore che consente all'operatore di abilitare o disabilitare il controllo automatico.

Un altro esempio di comportamento deterministico è rappresentato dal seguente flusso:



In ingresso al flusso in fase di produzione viene inserito un nodo mqtt socket che acquisisce i dati di un sensore di presenza. In fase di test questo nodo è sostituito con due nodi che iniettano nel flusso un evento di presenza o assenza. Il flusso ha lo scopo di generare il seguente comportamento da parte del sistema:

1. Se all'interno dell'ambiente non è presente nessun occupante e vengono rilevate le luci aperte allora il sistema provvederà a spegnerle;
2. Se all'interno dell'ambiente è presente un operatore e l'ora corrente è maggiore dell'ora di tramonto, allora accendi le luci;

I nodi presenti nel flusso sono due: il primo calcola l'ora di alba e l'ora di tramonto in funzione delle coordinate di geo localizzazione del dispositivo (latitudine e longitudine) ed il secondo abilita o disabilita il controllo automatico in funzione della presenza di un occupante e dell'ora attuale. Infine procede all'attuazione diretta sui dispositivi controllati. Nell'esempio si è deciso di intervenire solo sulle luci, ma è possibile estendere l'attuazione sulla tapparella, sul condizionatore o sugli altri dispositivi presenti nell'ambiente.

Blocco Q_Learning Controller

Infine, in questo paragrafo andremo ad analizzare lo sviluppo dell'ultimo blocco del sistema di controllo, ovvero il Controller.

Come anticipato nei paragrafi precedenti, questo blocco funzionale può essere scisso in due processi separati e paralleli:

- **il processo di aggiornamento della base di conoscenza (Learning)**
- **il processo di attuazione e/o di esplorazione**

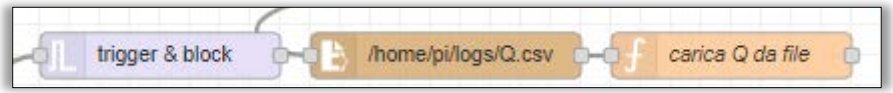
Per poter operare un algoritmo di Q_Learning ha le necessità di disporre di alcune informazioni basilari impartite dal progettista. Anzitutto necessita dell'insieme degli stati e delle azioni di cui il controllore può disporre, nonché il valore iniziale della matrice di conoscenza Q. L'insieme di queste informazioni vengono passate all'algoritmo attraverso un unico file, il file "Q.csv".

All'interno del file è contenuta una tabella Excel costruita come segue:

- sulla prima riga viene riportato l'insieme delle azioni,
- la prima colonna contiene l'insieme degli stati
- le altre celle della tabella contengono il valore atteso delle coppie (stato-azione).

Inizialmente il contenuto delle celle della tabella viene posto a zero. Il seguito questi valori vengono aggiornati e resi persistenti in memoria su file. La scelta dell'uso di una tabella Excel per memorizzare la matrice di conoscenza, rende più semplice l'interpretazione dei dati da parte dell'analista programmatore che può così seguire le fasi di apprendimento e di aggiornamento dell'algoritmo step by step.

Le informazioni contenute sul file devono essere caricare in memoria locale per poter essere rese disponibili immediatamente all'algoritmo di Q_Learning. Di seguito viene illustrato il flusso di inizializzazione, descritto da tre nodi:



Il nodo "trigger&block" in ingresso al flusso rende eseguibile il flusso per una sola volta, mentre il nodo funzione "/home/pi/logs/Q.csv" legge il file csv contenente la matrice di conoscenza, generando una stringa formattata che verrà successivamente analizzata dal nodo successivo.

Il nodo "carica Q da file" riceve la striga che descrive il contenuto del file csv letto ed istanzia le seguenti variabili d'ambiente:

- la matrice Q

- l’insieme degli stati (prima colonna del file csv)
- l’insieme delle azioni (prima riga del file csv)

I valori così letti verranno utilizzati nelle fasi successive dell’algoritmo.mIn dettagli il nodo “carica Q da file” viene descritto dal seguente codice:

```
var lines = msg.payload.split("\n");
var messages =[];
var items =[];
var state =[];
var azioni =[];
for(var i=1;i<205;i++){
    items = lines[i].split(",");
    var line =[];
    state.push(items[0]);
    for(var j=1;j<13;j++){
        line.push(parseFloat(items[j]));
    }
    messages.push(line);
}
azioni =lines[0].split(",");
flow.set('actions',azioni.slice(1, azioni.length));
flow.set('states',state);
flow.set('q',messages);
msg.payload = {"actions":azioni.slice(1, azioni.length),"states":state,"q":messages};
return msg;
```

Controller: Processo di aggiornamento

L’algoritmo Q-Learning ha la peculiarità di non inglobare la policy all’interno della regola di aggiornamento della base della conoscenza come invece ha l’algoritmo SARSA. L’algoritmo nell’aggiornare la sua base di conoscenza adotta la seguente strategia: per ogni stato che viene visitato durante l’esplorazione di una soluzione, esso va ad aggiornare la coppia stato-azione che produce **il massimo valore** atteso indipendentemente dall’azione che poi invece viene scelta per esplorare lo spazio degli stati. Per la scelta dell’azione si parla di politica di tipo Ɛ-greedy per dire che una certa percentuale di azioni che vengono intraprese vengono scelte in base alla conoscenza acquisita precedentemente, ed una parte di esse vengono scelte in modo random. Come già detto questa proprietà dell’algoritmo ci consente scindere in due processi la fase di aggiornamento e la fase di esplorazione.

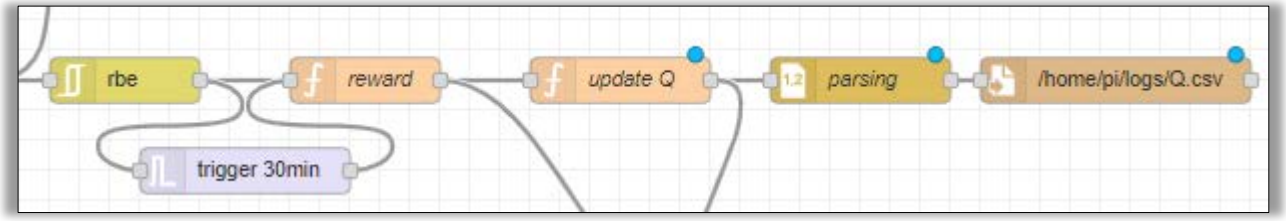
In questa architettura di sistema di tipo event driven non viene stabilita una temporizzazione o una suddivisione del tempo in step temporali. L’algoritmo è di tipo Time Difference TD, pertanto le sue elaborazioni e le sue scelte comunque si basano sull’osservazione dell’ambiente tra due step temporali. In questo caso i due step temporali oggetto di osservazione e valutazione non hanno un valore prestabilito. Questo aspetto ci aiuta in quando l’algoritmo interviene solo a seguito di una variazione di stato. Per poter intercettare una variazione di stato dell’ambiente oggetto di osservazione, in ingresso a questo flusso troviamo il nodo “rbe”. Questo nodo genera un evento/messaggio in uscita

solo se rileva una variazione tra il messaggio ricevuto precedentemente e quello attuale. I messaggi che arrivano in ingresso a questo flusso sono il risultato dell'elaborazione del flusso di generazione dello stato. Il flusso di generazione dello stato produce un messaggio per ogni lettura che gli arriva dai sensori posti al suo ingresso.

A seguire troveremo il nodo "reward", questo nodo ha la funzione di elaborare una ricompensa per il decisore in funzione dell'azione che ha prodotto la transizione avvenuta tra stato attuale e stato precedente.

Tra il nodo "rbe" ed il nodo "reward" troviamo il nodo "trigger 30min". Questo nodo avvia un timer quando riceve in ingresso un messaggio. Allo scadere il timer produce in uscita un nuovo messaggio.

Impostando il timer a 10-30 minuti, questo timer raggiungerà il tempo di spiro **soltanto se lo stato in ingresso corrisponde ad uno stato di confort**. Nel caso in cui si presenta uno stato di disconfort, il timer non avrà tempo di spirare perché interviene ancor prima il controllore per riportare il sistema in una condizione di confort.



Lo scopo di questo timer è quello di consentire una **valutazione di tipo "energetico"** dell'azione che è stata intrapresa per passare nello stato di confort. Perché è necessaria questa valutazione? Ebbene, l'ipotesi che si fa e che possano esistere più combinazioni di settaggio dei dispositivi di attuazione che conducono allo stesso obbiettivo, ovvero avere un certo grado di illuminamento.

Un esempio per tutti è l'accensione di tutte le luci piuttosto che l'apertura della tapparella. A parità di lux, ci possono essere dei momenti in cui conviene tenere aperte la luci piuttosto che la tapparella perché il calore che entra dalla finestra è talmente alto da produrre l'attivazione del condizionatore con un conseguente aumento dell'energia consumata.

Per catturare questo comportamento è necessario quindi intervenire anche in una situazione di confort per soppesare le azioni anche sulla base dell'impatto energetico che producono.

La strategia di utilizzo del timer ci consente di avere sempre la stessa unità di misura per ogni azione. Sappiamo che l'energia consumata è pari alla potenza assorbita per unità di tempo, misurando quindi l'energia prodotta dai vari dispositivi presenti nell'ambiente e dividendola per il tempo di timer è possibile ricavare un indice di confronto normalizzato.

In fase di test la componente energetica può essere trascurata e basare la logica di scelta considerando solo il confort luminoso.

```
var lock = global.get('controllo');
if(lock==='abilitato'){
    var statoat= msg.payload;
    var statoprec = flow.get('stato');
    var reward=0;
    if(statoat!=="1"){
        if(statoat.confort!=='confort'){
            reward = -0.8;
            //procediamo con la scelta di una azione
        }
        else{
            reward = null;
        }
    //flow.set('st',statoat);
    }
    else{
        var lastime = flow.get('lastime');
        var timestamp= Date.now() / 1000;
        reward=0;
        //proseguiamo con la scelta di un azione
    }
    msg.topic='reward';
    if (reward!==null){
        if(statoat==="1")
            msg.payload={"rw":reward,"st":flow.get('statoat')};
        else
            msg.payload={"rw":reward,"st":statoat};
    return msg;
    }
}
```

In questo nodo andiamo a verificare il valore della variabile d'ambiente "controllo" che definisce se il sistema è in controllo automatico o manuale. Nel caso in cui il controllo è disabilitato il nodo non genera nessun messaggio in uscita. Se il controllo automatico è abilitato si va ad indagare se il messaggio in ingresso è proveniente dal timer o dal flusso principale. Nel caso in cui il messaggio è pari ad "1", ci troviamo nella condizione di timer spirato ovvero in zona confort. Per il momento generiamo in uscita un reward=0.

Nel caso in cui il messaggio in ingresso non provenga dal timer, si va a verificare se lo stato attuale è uno stato di confort. Se siamo giunti in uno stato di confort non dovremmo fare nulla e quindi non restituiamo nessun messaggio in uscita. Diversamente una condizione di disconfort genera un reward negativo che poniamo uguale ad -0,8.

Avendo le informazioni sul reward, sullo stato attuale e sullo stato precedente è possibile ora procedere all'aggiornamento della matrice di conoscenza seguendo la seguente regola:

$$Q(s_{t-1},a_{t-1}) \leftarrow Q(s_{t-1},a_{t-1}) + \alpha[r_t + \gamma \max_a Q(s_t,a) - Q(s_{t-1},a_{t-1})]$$

Si va ad aggiornare la coppia stato azione dello step precedente. Il codice viene riportato di seguito:

```
Q(sprec,aprec)=(1-alfa)Q(sprec,aprec)+alfa(R(sprec,aprec)+gamma*Max[Q(satt,a)])
*/
var rw= Number(msg.payload.rw);
var newvalue= (1-alfa)*qm[h1][ap]+alfa*(rw+gamma*max(qm[h]));
qm[h1][ap]=newvalue.toFixed(2);
// aggiorno la matrice Q in memoria interna---------------
flow.set('q',qm);
// genera matrice q per salvataggio su file
var q =[];
q.push(primariga);
for(var i=0;i<stati.length;i++){
    var riga = qm[i].toString();
    riga = stati[i]+','+riga;
    var elem = riga.split(",");
    q.push(elem);
}
```

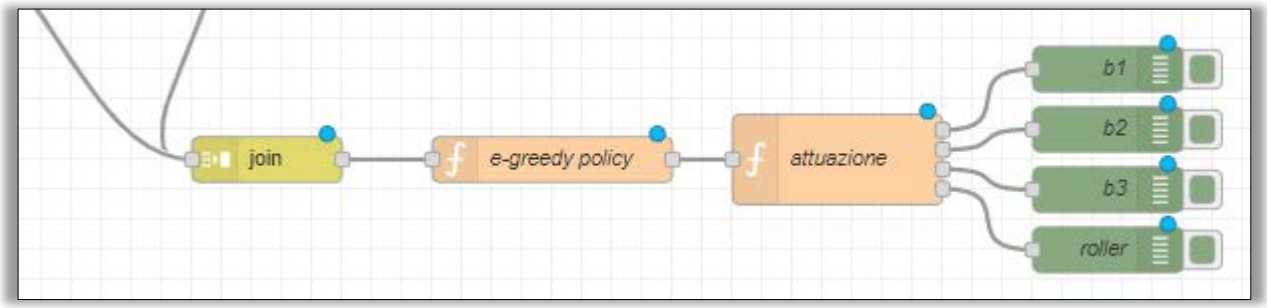
Anche in questo caso si pone il problema di dover gestire l'inizializzazione della variabile che memorizza l'azione al passo precedente. In questo caso si è scelto di impostare la prima volta l'azione "null-alza". Successivamente questa variabile verrà aggiornata nell'ultimo nodo che esegue l'attuazione sui dispositivi.

```
var alfa =1;// si tiene conto solo del reward attuale e non dei valori pregressi
var gamma =0.8;
var stati=flow.get('states');
var qm =flow.get('q').slice();
var r1=flow.get('actions');
var r0=['stato'];
var primariga = r0.concat(r1);
var a = flow.get('a');// azione stato precedente aggiornata nel nodo finale - flusso sequenziale
if(a===undefined ){
    flow.set('a','null-alza');//non accendere luci, alza tapparella.
    a = 0;
}//estrai indice azione precedente
var ap = primariga.indexOf(a);
// estrai riga stato attuale
var h = index(msg.payload.st);
//estrai riga stato precedente
var h1 = index(flow.get('stato'));
```

Controller*:* Processo di attuazione e/o di esplorazione

Nel paragrafo precedente abbiamo visto nel dettaglio come evolve il flusso di aggiornamento della matrice della conoscenza Q. Flusso che si conclude con l'aggiornamneto della matrice in memoria locale e successivamente su file di sistema, mantenendo la struttura della tabella CVS come impostata dallo sviluppatore. E' da sottolineare che si va ad aggiornare la coppia stato azione del paso precedente considerando come parametro il reward attualmente generato ed il valore massimo presente nella riga Q relativa allo stato attuale. L'azione corrispondente a questo valore massimo non è necessariamente l'azione che si andrà a scegliere per proseguire con l'elaborazione. Infatti la scelta della prossima azione segue come già detto la politica Ɛ-greedy, in modo da bilanciare in modo appropriata la fase di esplorazione verso l'apprendimento.

In ingresso a questo flusso troviamo un nodo di Join. Questo nodo prima di inviare un messaggio in uscita attende l'arrivo di due messaggi da due nodi differenti. In questo caso è utilizzato per acquisire informazioni dal nodo reward e da nodo update Q.



Riprendendo il codice descritto nei paragrafi precedente:

1. *Imposta il parametro epslon0, definito dal progettista.*
2. *Calcola il parametro epsoln come epsoln0 - (days/20)*
3. *Calcola l'insieme delle azioni ammissibili **A** dallo stato attuale*
3. *Calcola un numero random **x** compreso tra 0 ed 1*
4. *Se x < epsoln*

   Esplorazione: Seleziona l'azione a in modo random dall'insieme delle azioni ammissibili A.

   altrimenti

   Conoscenza: Seleziona l'azione a dalla matrice Q.

Di seguito andremo a vedere come questo si traduce in codice javascript all'interno del nodo funzione utilizzato.
Occorre come prima cosa elaborare l'insieme delle azioni ammissibili a partire dallo stato attuale, per fare ciò occorre partire dalle singole informazioni contenute nello stato e costruire un insieme di azioni che non sono assolutamente possibili da attuare, come ad esempio spegnare le luci quando lo stato ci dice che nessuna luce è accesa. Infine usando le nozioni di insiemistica, l'insieme delle azioni ammissibili saranno semplicemente tutte quelle che non sono comprese nell'insieme delle azioni inammissibili.

```
function legal_action(stato){
    var azioni = flow.get('actions');
    var count = stato.count;
    var pos = stato.pos;
    var angolo = stato.angolo;
    var confort = stato.confort;
    var nonammissibili= new Set();
    if(stato.count===0)
        nonammissibili.add('spegni-alza').add('spegni-abbassa').add('spegni-inclinasu').add('spegni-inclinagiu');
    else if (stato.count===3)
        nonammissibili.add('accendi-alza').add('accendi-abbassa').add('accendi-inclinasu').add('accendi-inclinagiu');
    if(stato.pos===0)
        nonammissibili.add('null-abbassa').add('spegni-abbassa').add('accendi-abbassa');
    else if (stato.pos===100)
        nonammissibili.add('null-inclinagiu').add('spegni-inclinagiu').add('accendi-inclinagiu').add('null-inclinasu').add('spegni-inclinasu')
        .add('accendi-inclinasu').add('null-alza').add('spegni-alza').add('accendi-alza');
    if(stato.angolo===0)
        nonammissibili.add('null-inclinagiu').add('spegni-inclinagiu').add('accendi-inclinagiu');
    else if (stato.angolo===90)
        nonammissibili.add('null-inclinasu').add('spegni-inclinasu').add('accendi-inclinasu');
    if(stato.confort==='bulo')
        nonammissibili.add('null-abbassa').add('spegni-abbassa').add('accendi-abbassa').add('null-inclinagiu').add('spegni-inclinagiu').add('accendi-inclinagiu');
    else if (stato.confort==='abbagliante')
        nonammissibili.add('spegni-alza').add('accendi-alza').add('null-alza');

    var ammissibili=[];
    for(var i=0;i<azioni.length;i++){
        if(!nonammissibili.has(azioni[i]))
            ammissibili.push(azioni[i]);
    }
    return ammissibili;
```

L’algoritmo di selezione dell’azione Ɛ-greedy invece è il seguente:

```
// epsoln va ad descescere al trascorrere dei giorni (n°iterazioni) fino ad annullarsi
// riparte col valore mazzimo a seguito di un reclamo
var q = flow.get('q');
var epsoln=0.5 - (day/20); //tasso di esplorazione 50% presi random

if (q!==undefined ){
    var items = flow.get('actions');
    var actions = legal_action(msg.payload.reward.st);
    var action;
    if(Math.random()<epsoln){
        //scegli un'azione in modo random tra lo spazio delle azioni ammissibili
        action = Math.floor(Math.random() * actions.length);
    }else{
        //scegli l'azione migliore dalla matrice Q
        var indicestatoattuale = index(msg.payload.reward.st);
        action = maxind(q[indicestatoattuale]);
    }
    msg.payload= items[action];
    return msg;
}
```

Infine abbiamo l’ultimo nodo del sistema che si occupa dell’attuazione sui singoli dispositivi. In questa configurazione abbiamo messo una uscita per ugni dispositivo controllato. Naturalmente i nodi per debug presenti in figura verranno sostituiti con nodi mqtt di interfacciamento verso i dispositivi.

Il nodo riceve in ingresso un comando del tipo “accendi-alza” che dovrà opportunamente interpretare e tradurre in comandi per i dispositivi.

La prima parte del comando è riferita alla luci ed ha tre tipi di valori : “null”, “accendi”,”spegni”.

Il comando non è riferito ad un singolo corpo luminoso, pertanto occorrerà verificare quale tra i corpi luminosi risulta attivo per poter poi procedere con l’accensione o lo spegnimento

```
// gestione accensione luci
var cmluci=null;
switch(comand[0]) {
    case 'null': break;
    case 'spegni':
        if(flow.get('b1')===true)
            cmluci='b1off';
        else if(flow.get('b2')===true)
            cmluci='b2off';
        else
            cmluci='b3off';
    break;
    case 'accendi':
        if(flow.get('b1')===false)
            cmluci='b1on';
        else if(flow.get('b2')===false)
            cmluci='b2on';
        else
            cmluci='b3on';
    break;
}
```

Un po' più complicata è la gestione del controllo della tapparella. In quest’ultima occorre partire dalla condizione preesistente della tapparella e risalire alla nuova posizione considerando il fatto che l’inclinazione delle doghe è ottenibile soltanto attraverso la regolazione dell’altezza. Un esempio di gestione è il seguente:

```
else if(p.pos===25 && p.angolo===0)//{posi=25; if (direzione=='giu') ang=0; else ang=90;}
    switch(comand[1]) {
        case 'alza': altezza =50;flow.set('dir','su');break;
        case 'abbassa':altezza =0;flow.set('dir','giu');break;
        case 'inclinasu': if (direzione==='giu') altezza=p.pos+2; break;
        case 'inclinagiu':if (direzione==='su') altezza=p.pos-2; break;
    }
else if(p.pos===25 && p.angolo===30)//{posi=25;ang=30;}
    switch(comand[1]) {
        case 'alza': altezza =50;flow.set('dir','su');break;
        case 'abbassa':altezza =0;flow.set('dir','giu');break;
        case 'inclinasu': altezza=28; break;//60°
        case 'inclinagiu':altezza=25; break;//0°
    }
```

Dove possiamo notare che oltre alle informazioni della posizione precedente, occorre tenere traccia anche della direzione di movimento della tapparella che ne condiziona l'inclinazione delle doghe.

```
// aggiorna azione precedente con l'azione attuale
flow.set('a',msg.payload);// utilizzata in aggiorna Q
// aggiorna lo stato precedente con lo stato attuale
var st = flow.get('statoat');
flow.set('stato',st);
// eseguiamo i comandi sulle varie uscite
var msg2={"payload":altezza, "topic":''};
var msg1={"payload":'', "topic":''};
if(cmluci!==null)
    switch(cmluci){
        case 'b1off':msg1.payload = 'off'; return [msg1,null,null,msg2];
        case 'b2off':msg1.payload = 'off'; return [null,msg1,null,msg2];
        case 'b3off':msg1.payload = 'off'; return [null,null,msg1,msg2];
        case 'b1on':msg1.payload = 'on'; return [msg1,null,null,msg2];
        case 'b2on':msg1.payload = 'on'; return [null,msg1,null,msg2];
        case 'b3on':msg1.payload = 'on'; return [null,null,msg1,msg2];
    }
else return [null,null,null,msg2];
```

Prima di procedere con l'inoltro dei comandi in uscita ai dispositivi, devono essere aggiornate le due variabili: "stato precedente" ed "azione precedente" necessarie al prossimo step di apprendimento al flusso di aggiornamento della matrice Q.

Analisi dei risultati

Una prima analisi del comportamento del sistema è stata effettuata considerando il feedback energetico proveniente da un impianto di illuminazione costituito da tre lampade a led controllabili e da un solo sensore di luminosità posto in un punto adeguato della stanza. Tale sensore invia il dato dopo un minuto se rileva una variazione significativa del valore misurato (+-5%) altrimenti va in uno stato di quiescenza ed invia il dato dopo 5 minuti. I dati sono stati raccolti inserendo nel flusso un nodo di "Log" che ha la funzione di archiviare le informazioni in ingresso in un file di testo aggiungendovi un timestamp. A tale nodo sono state fatte confluire le uscite del nodo di ingresso (lux), del nodo che determina lo stato, poi il rewad, l'insieme delle azioni ammissibili con la relativa scelta dell'algoritmo. Il sistema è stato lasciato evolvere liberamente per cinque giorni alla fine dei quali si è fatta una prima valutazione. In questa prima implementazione è stato realizzato un primo sistema di gestione dei reclami molto semplificato. A seguito di un solo reclamo il sistema aggiorna il valore di soglia precedentemente impostato, pertanto non vengono effettuati test rispetto questa funzionalità. Successivamente è stato inserito il blocco di elaborazione per gestire il "**miglioramento paretiano**".

Tutto ciò premesso, per comprendere dall'analisi dei dati raccolti se l'algoritmo rispetta gli obbiettivi prefissati, occorre fare delle precisazioni. Il sistema è stato concepito come sistema di controllo ad anello chiuso per la regolazione della luminosità.

La dinamica che segue è quindi la dinamica tipica di un sistema di controllo, ovvero:

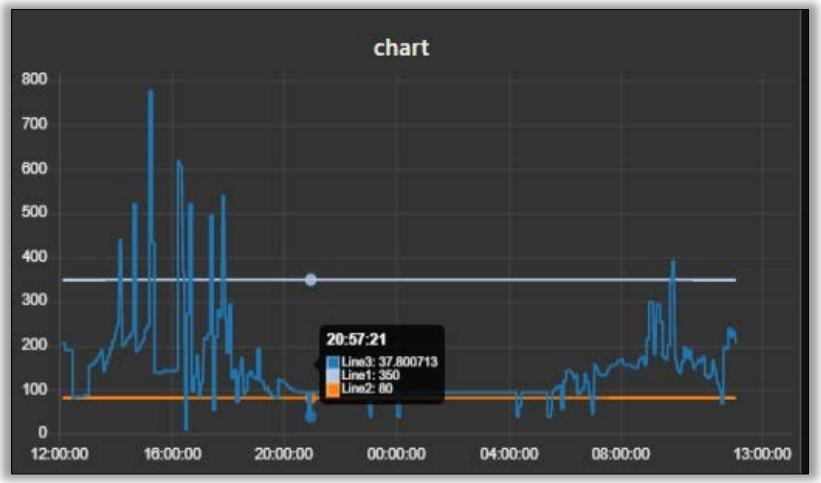
1. Misura il parametro di controllo
2. Confronta il valore letto con il setpoint e calcolo dell'errore
3. Se il valore del parametro di controllo supera i valori di setpoit (errore maggiore di zero) esegui un'azione per riportare il sistema in controllo.

Tipicamente non esiste un solo punto che consente al sistema di tornare in una condizione di controllo, ma esistono varie opzioni. Diversamente quindi da un ciclo di controllo classico, il controllore in questo caso vaglierà tutte le possibili azioni che portano il sistema in uno stato di controllo e sceglierà quella che produce il massimo risparmio energetico.

Questa dinamica, come si può constatare richiede un processo di decisione "one-step". Diversamente da altri processi decisionali in cui è richiesto al decisore di stabilire qual è la sequenza ottima di azioni che consentono l'evoluzione del sistema da uno stato iniziale verso uno stato terminale. In questo caso non essendoci uno stato terminale, poiché il raggiungimento di uno stato di "confort" non produce la terminazione del ciclo, il decisore sarà chiamato a decidere soltanto quale azione ci consente di uscire da una situazione di disconfort ottenendo il maggior risparmio energetico.

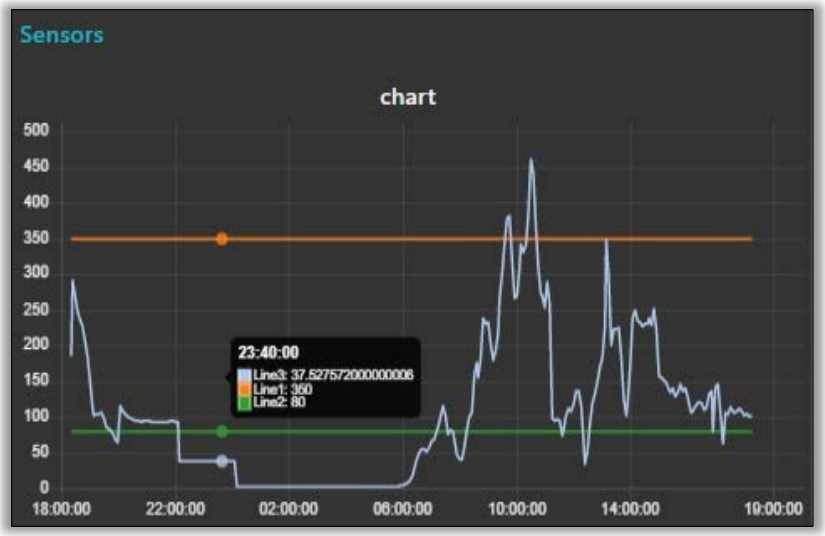
Alla luce di queste considerazioni vediamo di seguito come evolve il sistema e le criticità riscontrate.

Nel grafico successivo viene riportato il comportamento del sistema nei primi due giorni di addestramento. Imponendo una policy che alterna l'apprendimento verso l'esplorazione con pari probabilità ε=0.5 nel primo giorno, per poi passare a scelte sempre più improntate sui valori appresi con ε decrescente, possiamo notare come nel secondo giorno di apprendimento il sistema abbia discrimitato ed escluso quelle azioni che hanno determinato uno sforamento dei limiti di controllo nel primo giorno di funzionamento.



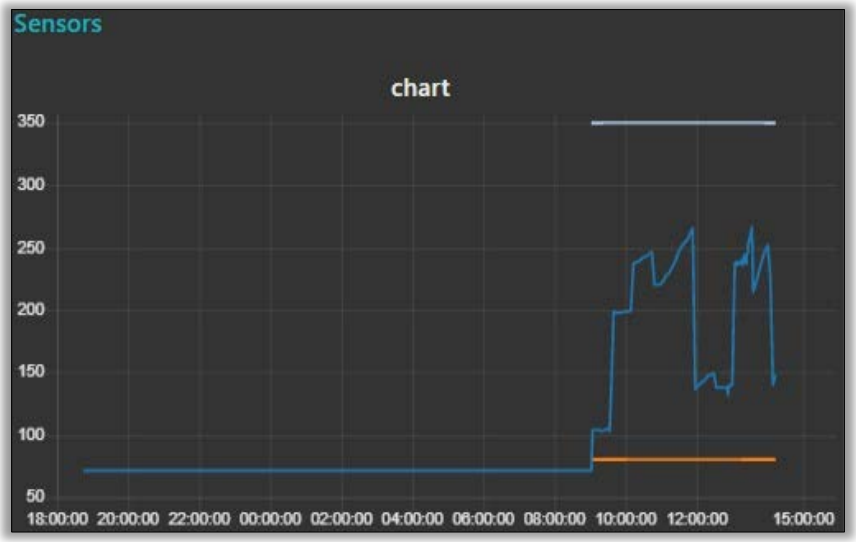
Nel grafico successivo viene evidenziato come il sistema risponde alle variazioni repentine di luminosità tra le 12 e le 13 che producono abbagliamento. In questo caso riesce ad abbassare

completamente la tapparella mantenendo le luci aperte. Vediamo inoltre come ogni mezzora tende a cercare una soluzione diversa che generi un guadagno. Infine alle 14 riesce ad alzare la tapparella senza incorrere in abbagliamento diretto ed a spegnere le luci.



Dai dati raccolti nella fase di addestramento si è evidenziato un comportamento del sistema in linea con le aspettative. Imponendo il learning rate ed il discount factor ad un valore pari ad uno nella Value Function, ovvero imponendo un indice di apprendimento massimo abbiamo ottenuto l'effetto desiderato. Questo introduce una reattività immediata del sistema ai cambiamenti, fondamentale per sistemi on-line. Di contro si elimina l'effetto memoria con la scelta precedentemente compiuta dal decisore e la propagazione dei reward nella catena di decisione che come già detto nel nostro caso è lunga 1, massimo due step.

Dopo alcuni giorni di apprendimento si è deciso di sospendere la fase di esplorazione modificando la policy, ovvero ponendo il parametro $\varepsilon=0$. Il risultato ottenuto è stato il seguente:



Da questo grafico si evincono due informazioni:

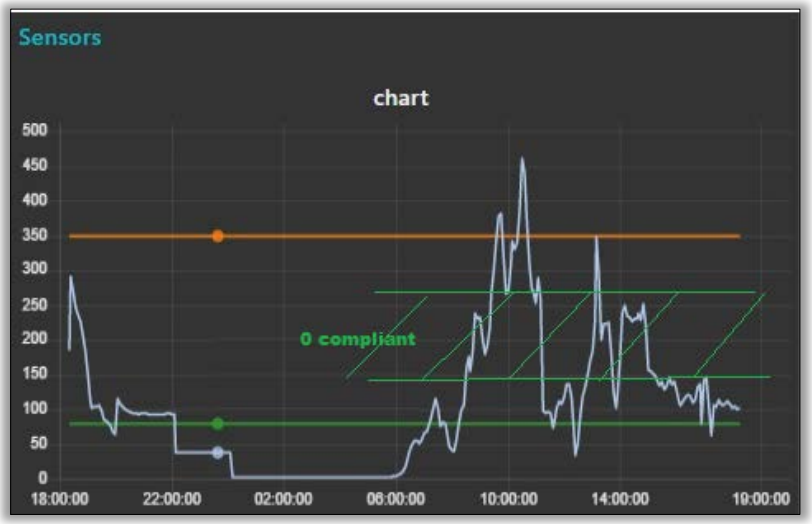
1. Il sistema evolve per step temporali di 30 min. Gli eventi che scatenano l'intervento del controllore sono due: o il rilevamento di uno stato di disconfort o l'intervento del timer utilizzato in fase di apprendimento per generare la stima del guadagno energetico.
2. Il sistema evolve per stati di confort successivi mantenendosi il più vicino possibile al valore medio di 200 lux, **discostandosi da quella che è la dinamica di un sistema di controllo classico**, che come abbiamo evidenziato interviene solo al superamento di una soglia. Inoltre notiamo che non esistono punti oltre la fascia 80-140 ed 260-350 lux.

Relativamente al punto 1, si è visto che un intervento continuo (con cadenza di mezz'ora in questo caso) del sistema potrebbe ingenerare disturbo acustico e visivo ed essere poco tollerato dall'utente.

Una disattivazione del timer, una volta che la matrice è addestrata, permetterebbe un funzionamento di tipo "classico" del sistema di controllo, ovvero quanto il sistema passa in uno stato di confort vi rimane finchè non interviene una variazione significativa della variabile di controllo che lo porti in disconfort.

Relativamente al punto 2, dall'analisi effettuata sugli stati percorsi dal sistema per mantenere il livello di lux nell'intorno del valore di 200, si è evidenziata una ***situazione inaspettata***, ovvero il sistema mantiene accese durante tutto il giorno una o due luci per bilanciare il gap tra luce interna ed esterna. La scelta a primo acchito è sembrata errata in quando l'accensione di luci comporta uno spreco energetico. Ancor di più inaspettato perchè in questa prima realizzazione l'unico bilancio energetico viene fornito proprio dalle luci, manca di fatto la misura dell'impatto energetico introdotto dal condizionamento.

Una spiegazione a queste scelte viene fornita analizzando più in dettaglio il seguente grafico:



Incrociando i dati provenienti dal grafico ed i reclami che raggiungono il controllore possiamo denotare due zone:

1. Zona "0 compliant", parte centrale del grafico compresa tra i 140 ed i 260 lux
2. Zona "compliant", parte della regione di confort compresa tra 80 -140 lux ed 260 – 350 lux

La zona “0 compliant” viene generata dall’algoritmo nella sua evoluzione “casuale” quanto questo evolve da uno stato di confort ad un altro stato di confort. In questo caso l’unico reward che viene recepito dal sistema è quello di risparmio energetico. Questo generalmente avviene se si ci discosta di poco dal valore medio della zona di confort. Condizione al momento possibile attraverso l’accensione o lo spegnendo delle luci che hanno un impatto meno incisivo sulla luminosità rispetto alla tapparella.
All’interno di questa zona, l’algoritmo dopo l’apprendimento, “**trova**” in qualche modo un percorso che lo fa evolvere per stati successivi di confort, evitando lo sforamento dei limiti. Comportamento come già evidenziato che si discosta dal funzionamento tipico di un sistema di controllo.

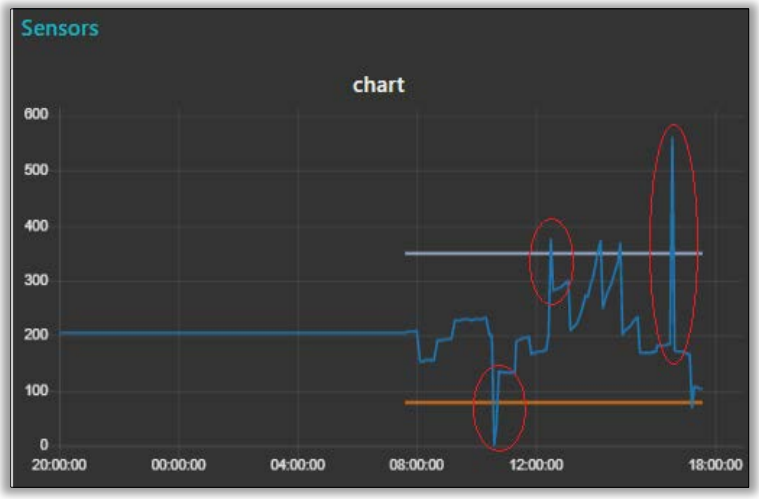
La zona “compliant” viene generata da tutte quelle coppie stato-azione che creano discostamenti troppo repentini del sistema e pertanto sono più portate ad avere reward di disconfort negativi. Questa ultima zona viene “**tagliata**” dal controllore **perché il valore di reward associato** a tali stati ha un peso pari ad **80**, che rispetto ai 20 del reward energetico produce questa zona d’ombra.

In conclusione possiamo dire che con i parametri di settaggio forniti al sistema, si riesce a raggiungere una condizione di funzionamento “ottimale” dal punto di vista del confort visivo in pochi giorni di apprendimento. Margini di miglioramento sono ottenibili cercando una strategia di gestione del comportamento in zona confort, con l’introduzione di un timer variabile che eviti interventi continui del decisore ed inserendo una strategia ottimale predittiva del superamento della soglia.

Una seconda anali approntata sul sistema è stata fatta sulle reazioni alle sollecitazioni esterne. Se l’operatore interviene sul sistema per modificarne lo stato, come reagisce il controllore?

Per rispondere a questa domanda sono stati compiuti due tipi di test:

1. Variazioni dell’angolo della tapparella ed accensione luci in modo da non generare un aumento eccessivo dei lux. In questo caso l’algoritmo non interviene a variare l’azione dell’utente perché si permane in uno stato di confort. Soltanto dopo 30 minuti il controllore cambierà stato. Non è ancora chiaro se il sistema “lascia” lo stato attuale verso uno energicamente più favorevole.
2. Si interviene con azioni che ci portano in una condizione di disconfort. Il grafico sottostante mostra l’esito dell’esperimento.



Nella maggior parte dei casi, vedi ad esempio il primo e l'ultimo punto mostrato nel grafico, il sistema si riporta nella zona "0 compliant". In un solo caso il sistema rimane nella zona "compliant" ma comunque riesce in due step a rientrare nella fascia "0 compliant".

A causa dei tempi di risposta del sensore di luminosità, sul grafico sono visibili degli spikes e non dei singoli punti. Il tempo di risposta del sensore è dell'ordine dei 5 minuti, pertanto l'esito dell'intervento del controllore è visibile sul grafico perché interviene solo su ricezione del feedback del sensore, impiegando un tempo che può sembrare eccessivo. L'inserimento di altri sensori di luminosità nell'ambiente ci ha consentito di migliorare i tempi di risposta rendendo il sistema più reattivo, come si evince dal seguente grafico:

Confort visivo ed indici di confort

L'illuminazione di un ambiente deve svolgere fondamentalmente tre funzioni:

- sicurezza: deve consentire lo svolgimento dell'attività ed il movimento in condizioni di sicurezza
- prestazione: deve consentire lo svolgimento del compito visivo in condizioni ottimali
- comfort: deve garantire un ambiente interno confortevole

La luce migliore è quella naturale, perché riproduce i colori fedelmente, giova al benessere psicofisico delle persone ed è gratuita. Inoltre, un'adeguata illuminazione diurna ed un'integrazione ottimale di luce naturale/artificiale possono contribuire in maniera significativa al risparmio energetico negli edifici. Una disponibilità di luce naturale particolarmente elevata in un ambiente chiuso può creare alcuni svantaggi per le condizioni climatiche, un elevato irraggiamento comporta un incremento notevole dei carichi frigoriferi e quindi un conseguente incremento del fabbisogno energetico in fase di climatizzazione.

Il confort visivo, in linea generale, dipende da:

- livello di illuminamento
- livello di abbagliamento

Per garantire una buona qualità di luminosità all'interno delle abitazioni, la dimensione delle finestre dovrebbe corrispondere al 10% – 12% di quella del pavimento. In particolare, la normativa nazionale stabilisce un fattore medio di luce diurna non inferiore al 2% e una superficie apribile non inferiore a 1/8 della superficie calpestabile. Entrambi i fattori determinano il rapporto aero-illuminante che deve assicurare due condizioni: luce e ricambio d'aria sufficiente all'interno dei locali.

Il comfort visivo dipende non solo dal livello dell'illuminamento interno, ma include anche il rapporto visivo che si ha tra l'interno verso l'esterno. Pertanto le finestre non solo devono procurare la luce necessaria per svolgere un'attività, ma anche consentire la vista dell'esterno. Indipendentemente dal rispetto dei rapporti aero-illuminanti di legge, il fattore più indicativo del livello di illuminazione naturale di un ambiente interno è il **fattore di luce diurna**: esso esprime il rapporto fra l'illuminamento in un punto all'interno dell'edificio e quello che si avrebbe in assenza dell'edificio stesso.

Per poter inserire dei dispositivi di ombreggiamento al fine di contribuire al risparmio del fabbisogno energetico e nel contempo garantire dei buoni livelli di confort visivo, occorre innanzitutto verificare se l'ambiente abbia buone caratteristiche strutturali che ci consentano di intervenire con dispositivi di ombreggiamento.

Nel caso specifico del laboratorio sito nell'ICAR-CNR di Rende è stato precedentemente calcolato il fattore di luce diurna e verificata la possibilità di installazione di un sistema di ombreggiamento automatizzato.

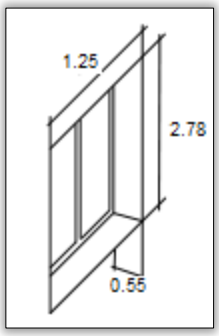
Seguendo i metodi e le tabelle di riferimento della normativa nazione si è pervenuti ai risultai che vengono descritti di seguito.

Il primo indice che viene calcolato è il **fattore medio di luce diurna**, ricavato come:

$$F_{mld} = \frac{\sum_i A_i \tau_i \varepsilon_i \psi_i}{S(1 - r_m)}$$

in cui:

- S è la somma delle superfici interne illuminate. Per una stanza di forma rettangolare di dimensioni
  in pianta 4,00 m x 5 m e altezza 2,70 m, S = 88,6 m^2
- Ai è l'area della finestra i-esima.



Ai = 2.78*1.25 = 3.48m^2

- τi è il coefficiente di trasmissione luminosa del vetro.

Tabella I. Valori indicativi del coefficienti di trasmissione per incidenza normale nel visibile di alcuni sistemi trasparenti.

Sistema trasparenti	τ_v
vetro float singolo chiaro 4-6 mm	0,80-0,90
vetro float singolo assorbente	0,70-0,80
vetro singolo retinato	0,85
vetro float singolo colorato in massa a seconda del colore	0,30-0,60
vetro float singolo riflettente	0,35-0,60
vetro float singolo bassoemissivo	0,50-0,75
doppio vetro 6-12-6 – lastre float chiare	0,65-0,75
doppio vetro 6-12-6 – lastre float chiare con ricoprimento bassoemissivo	0,60
policarbonato chiaro	0,80-0,90
lastre traslucide in materiale plastico	0,10-0,80

Nel nostro caso abbiamo un doppio vetro con ricoprimento bassoemissivo τi = 0,60

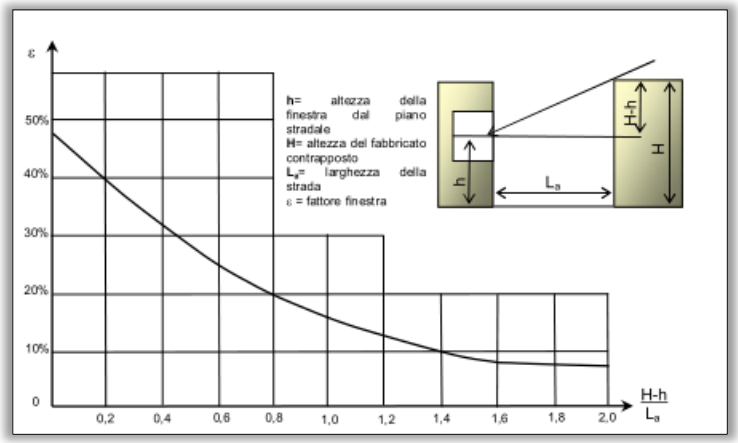
- rm è il coefficiente di riflessione medio nel visibile delle superfici che costituiscono l'involucro dell'ambiente considerato;

Tabella II. Valori indicativi dei coefficienti di riflessione, r, per differenti colori delle superfici.

colore	**r**	**colore**	**r**
bianco	0.90 ÷ 0.75	blu scuro	0.10 ÷ 0.05
avorio	0.85 ÷ 0.80	verde scuro	0.10 ÷ 0.05
crema	0.80 ÷ 0.70	marrone	0.15 ÷ 0.05
giallo chiaro	0.70 ÷ 0.60	rosso scuro	0.10 ÷ 0.05
rosa	0.60 ÷ 0.45	grigio chiaro	0.40 ÷ 0.15
arancio	0.60 ÷ 0.40	grigio scuro	0.15 ÷ 0.05
verde chiaro	0.50 ÷ 0.40	nero	0.04 ÷ 0.01
azzurro chiaro	0.45 ÷ 0.40		

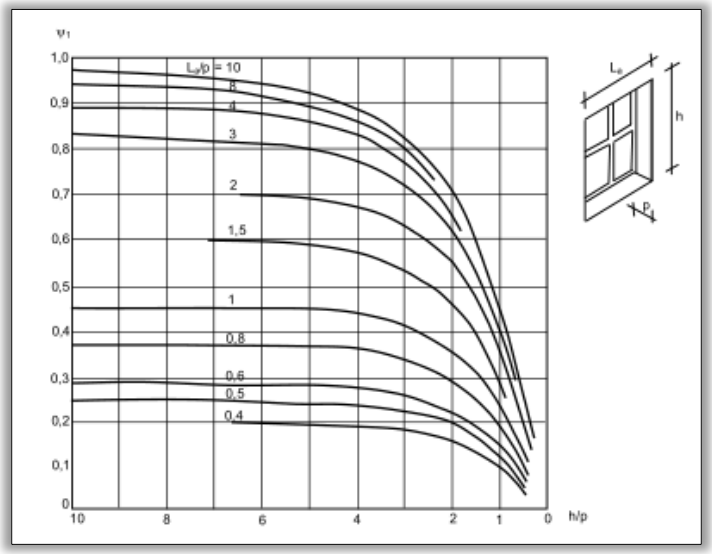
Le pareti del laboratorio riportano una superficie bianca con rm = 0.75-0.9

- εi , è il fattore finestra cioè il rapporto tra l'illuminamento sul baricentro della finestra e l'illuminamento su una superficie orizzontale liberamente esposta alla volta celeste. Di fronte alla finestra ad una distanza di 4 m si eleva il piano stradale con ostruzioni di piante e macchine. Possiamo considerare un fattore H-h/La = 0.25. Dove H = 3m è l'altezza dell'ostruzione rispetto al piano di riferimento; h = 2m è l'altezza del centro della finestra rispetto al piano di riferimento; La=4m è la distanza dell'ostruzione dalla finestra.



Tracciando la verticale sull'asse delle ascisse, il punto corrispondente al rapporto ottenuto fino ad intersecare la curva nel punto corrispondente, sull'asse delle ordinate, si ottiene il valore del fattore finestra: εi = 40%

- ψi , è un fattore che tiene conto dell'ombreggiamento indotto sulla finestra dall'imbotte. Ottenuti: h f / P = 1,25/0,55 = 2.27 e l f / P = 2.78/0,55 = 5 si trova attraverso il grafico seguente il valore del coefficiente di riduzione: ψ1 = 0,86.



Fmld = (A1* τ1* ε1* ψ1)/S(1-rm) =(3.48*0.6*0.4*0.86)/88.6 *(1-0.8) = 3.6%

I risultati ottenuti da dal calcolo strutturale sopra esposto ha portato a **determinare un fattore di luce diurno medio di 3.6%** che colloca l'ambiente nella fascia tra il 2% ed il 4% considerata "buona".

F < 0,3%	insufficiente
0.3% < F < 2%	discreto
2% < F < 4%	buono
4% < F	ottimo

In ottemperanza della circolare ministeriale N°3151, al punto 1.3.02, i livelli minimi che si devono garantire all'interno di uno spazio adibito ad ufficio non devono essere inferiori al 1%:

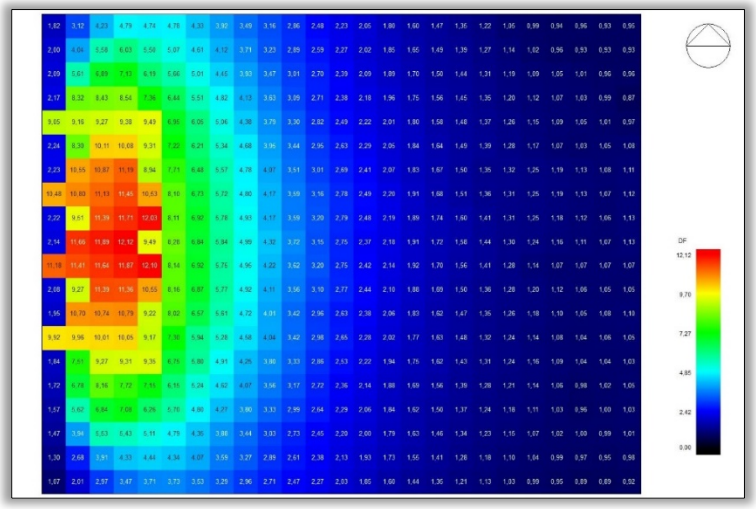
tipo di destinazione d'uso	F_{mid}
ambienti di degenza	0,03
palestre, refettori	0,02
uffici, spazi di distribuzione, scale	0,01

Ciò ci consente di poter variare l'apertura e l'inclinazione della tapparella fino ad un valore calcolato come:

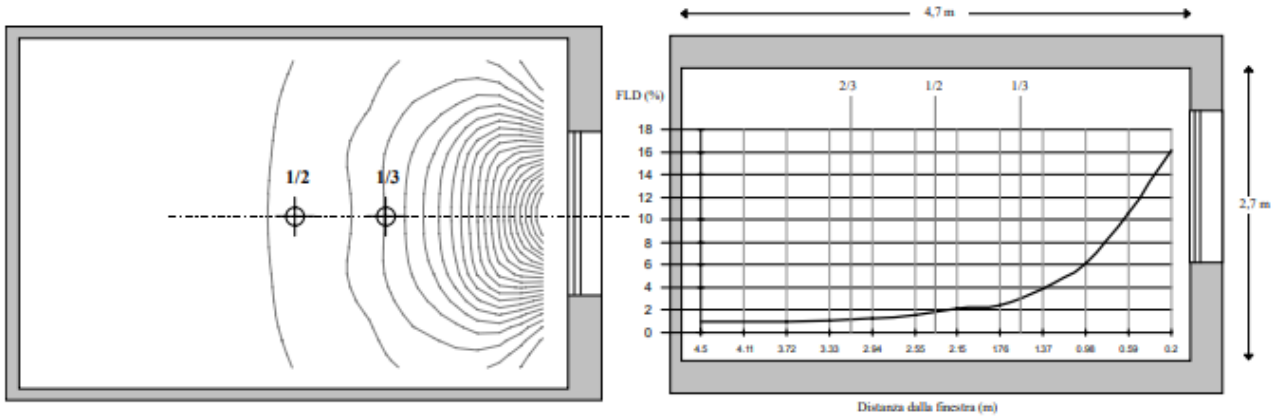
$$0.01=(x*2.78*0.6*0.4*0.86)/88.6 *(1-0.8)$$

$$X= 0.01* [88.6 *(1-0.8) / (2.78*0.6*0.4*0.86)] = 0.3m$$

corrispondente al **25%** del valore dell'altezza totale.
Naturalmente stiamo parlano di valori medi calcolati, i valori puntuali calcolati con uno specifico software di Illuminotecnica mostrerebbero un andamento dei valori di illuminamento come riportati di seguito:



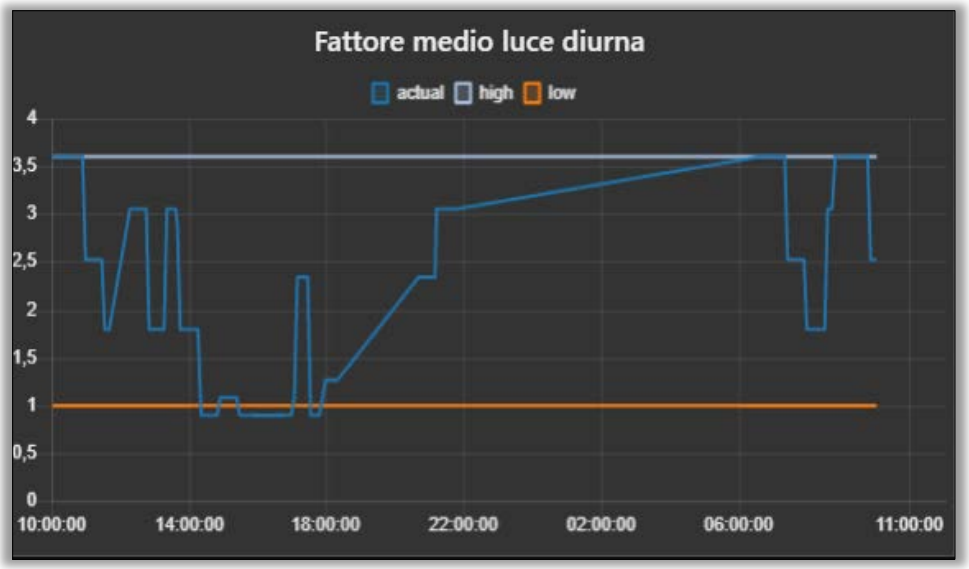
Man mano che ci allontaniamo dalla superficie vetrata il livello di illuminamento degrada in maniera esponenziale:



Sulla base dei calcoli effettuati, relativamente all'ambiente considerato per il test, siamo riusciti ad elaborare un grafico che ci dia indicazione della variazione del fattore di luce diurna medio in funzione dell'attuazione eseguita.
Considerando che a finestra completamente aperta corrisponde un Fmld = 3,6% ed a una apertura della finestra del 25% corrisponde un Fmld = 1%, in proporzione abbiamo ricavato il valore di Fmld per le varie posizioni della tapparella:

```
1
2  var p = flow.get('posizione');// posizione={"pos":posi,"angolo":ang};
3  var fml = 0.0;
4  if(p.pos===0 && p.angolo===0)//{posi=0;ang=0;} altezza 0
5      fml = 0.0;
6  else if(p.pos===0 && p.angolo===30)//{posi=0;ang=30;} altezza 2
7      fml = 0.36;//10%
8  else if(p.pos===0 && p.angolo===60)//{posi=0;ang=60;} altezza 3
9      fml = 0.54;//15%
10 else if(p.pos===0 && p.angolo===90)//{posi=0;ang=90;} altezza 4
11     fml = 0.72;//20%
12 else if(p.pos===25 && p.angolo===0)//{posi=25; ang=0;} dir=giu
13     fml = 0.9;// 25%
14 else if(p.pos===25 && p.angolo===30)//{posi=25;ang=30;} altezza 27
15     fml = 1.08;//30%
16 else if(p.pos===25 && p.angolo===60)//{posi=25;ango=60;}
17     fml = 1.26;//35%
18 else if(p.pos===25 && p.angolo===90)//{posi=25;ang=90;} altezza 29 dir su
19     fml = 1.44;// 40%
20 else if(p.pos===50 && p.angolo===0)//{posi=50;ang=0} dir giu
21     fml = 1.8;// 50%
22 else if(p.pos===50 && p.angolo===30)//{posi=50;ango=30;} altezza 52 dir su
23     fml = 1.98;// 55%
24 else if(p.pos===50 && p.angolo===60)//{posi=50;ango=60;} altezza 53 dir su
25     fml = 2.16;// 60%
26 else if(p.pos===50 && p.angolo===90)//{posi=50;ango=90;}
27     fml = 2.34;// 65%
28 else if(p.pos===75 && p.angolo===0)//{posi=75;ang=0;} altezza 75 dir giu abbassa
29     fml = 2.52;// 70%
30 else if(p.pos===75 && p.angolo===30)//{posi=75;ango=30;} altezza 77
31     fml = 2.7;// 75%
32 else if(p.pos===75 && p.angolo===60)//{posi=75;ango=60;} altezza 78
33     fml = 2.88;// 80%
34 else if(p.pos===75 && p.angolo===90)//{posi=75;ango=90;} altezza 79
35     fml = 3.06;// 85%
36 else //{posi=100;ango=90;} altezza 100
37     fml = 3.6;// 100%
```

L’andamento giornaliero che ne è venuto fuori è il seguente:



In questo caso tra le 14 e le 18 a seguito dei raggi diretti che colpiscono la superficie vetrata, la tapparella risulta chiusa sotto il 25% producendo un indice Fmld = 0,9%.

Il secondo indice che viene calcolato è **l’indice di abbagliamento da luce naturale** (daylight glare index).

L’abbagliamento da luce naturale dipende dai seguenti fattori:

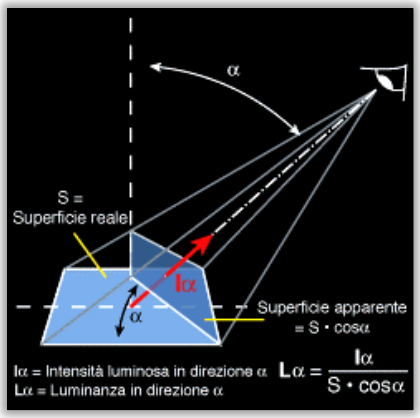
- luminanza della porzione di cielo inquadrata dalla superficie vetrata
- posizione e dimensione della superficie vetrata
- contrasto di luminanza tra le superfici interne
- presenza di superfici riflettenti

Per definire i valori limite negli ambienti di lavoro si utilizza l’indice DGI (daylight glare index) definito dalla UNI 10840:2000.

abbagliamento intollerabile	DGI>28
abbagliamento quasi intollerabile	DGI=28
abbagliamento fastidioso	DGI=26
abbagliamento quasi fastidioso	DGI=24
abbagliamento appena accettabile	DGI=22
abbagliamento accettabile	DGI=20
abbagliamento percepibile	DGI=18
abbagliamento appena percepibile	DGI<16

Si definisce intensità luminosa ***I*** il rapporto tra il flusso luminoso infinitesimo emesso dalla sorgente in una data direzione e l’angolo solido elementare. L’unità di misura è la candela (cd).

Un flusso luminoso di 1 candela produce un illuminamento di 1 lux ad 1 metro di distanza e 0.25 lux a 2 metri di distanza (1/4), allo stesso modo 1 lumen distribuito su un'area di 1 m2. (legge dell'inverso del quadrato: E = I/d2)
La luminanza L in un punto di una superficie, in una certa direzione, è il rapporto tra l'intensità luminosa emessa in quella direzione e la superficie emittente proiettata su un piano perpendicolare alla direzione stessa. L'unità di misura è candela per metro quadrato (cd/m^2).

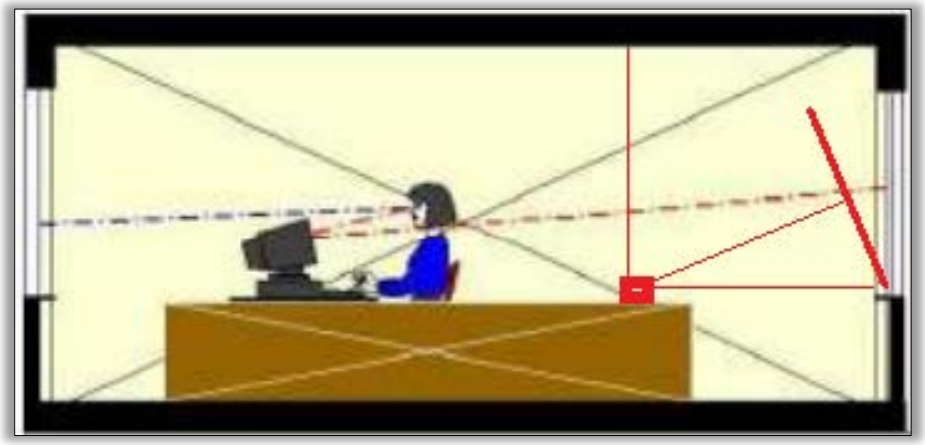


$$= \frac{dI}{dA \cdot \cos\ \alpha}$$

Se un luxmetro rileva 500lx da una fonte luminosa posta a due metri di distanza, essa produrrà il corrispondente di 125 candele:

500 lx (2m distanza) -> 500/4 = 125 cd

Considerando un sensore di luminosità posto sul piano di lavoro ad una distanza prestabilita dalla superfice vetrata, è possibile calcolare la proiezione la superficie emittente sul piano perpendicolare alla direzione stessa pari a cosα A



Dove A = 2.78 + 1,25 = 3.48 m^2

Per il calcolo di cosα consideriamo il sensore posto su un lato della finestra ad una altezza di 1,20m ad una distanza di 2m. La proiezione lungo la perpendicolare sarà di:

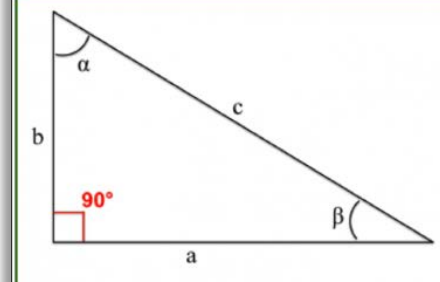
$$Cos\alpha = a/c = 2/2.36 = 0,85$$

La proiezione longitudinale sarà:

$$Cos\beta = a/c = 2/3.34 = 0,60$$

I due proiezioni producono una riduzione dell'area di A'=3.48*0.85*0.6=1.77 m^2

Triangolo rettangolo Formule trigonometriche



$a = c\ sin(\alpha)$	$a = b\ tan(\alpha)$
$a = c\ cos(\beta)$	$a = b\ cot(\beta)$
$b = c\ sin(\beta)$	$b = a\ tan(\beta)$
$b = c\ cos(\alpha)$	$b = a\ cot(\alpha)$

$$L\alpha\ (500lx) = 125/A' = 125/1.77 = 70\ cd/m^2$$

Attraverso queste approssimazioni abbiamo legato l'illuminamento, misurabile attraverso un luxometro, e la luminanza prodotta dalla superficie vetrata paragonandola ad un corpo luminoso.

Quindi abbiamo trovato la seguente corrispondenza: a 500 lx misurati dal sensore posto alla distanza di 2m dalla finestra si ottiene un valore di luminanza prodotta dalla finestra pari a 70 cd/m^2 .

Ottenuta la luminanza della finestra come misura indiretta dell'illuminamento ora non ci resta che ricavare gli altri parametri che definiscono l'indice DGI .

L'abbagliamento dovuto alla luce naturale può essere verificato facendo riferimento all'indice di abbagliamento DGI

$$DGI = 10 \log \sum_{i=1}^{n} G_i$$

$$G_i = 0{,}48 \frac{L_s^{1,6} \ \Omega^{0,8}}{L_b + 0{,}07 \ \omega^{0,5} L_w}$$

dove:

DGI	è l'indice di abbagliamento;
G_i	è la costante di abbagliamento calcolata per ciascuna porzione di sorgente, primaria e secondaria, vista attraverso la finestra (cielo, ostruzioni, terreno);
L_s	è la luminanza della sorgente (primaria o secondaria) in cd m^{-2};
Ω	è l'angolo solido sotteso dalla sorgente (primaria o secondaria) corretto in relazione alla direzione di osservazione in steradianti (sr);
L_b	è la luminanza media delle superfici interne dell'ambiente, che rientrano nel campo visivo dell'occupante in cd m^{-2};
ω	è l'angolo solido totale sotteso dalla finestra in steradianti (sr);
L_w	è la luminanza media della finestra, ponderata rispetto alle aree relative di cielo, ostruzione e terreno in cd m^{-2}.

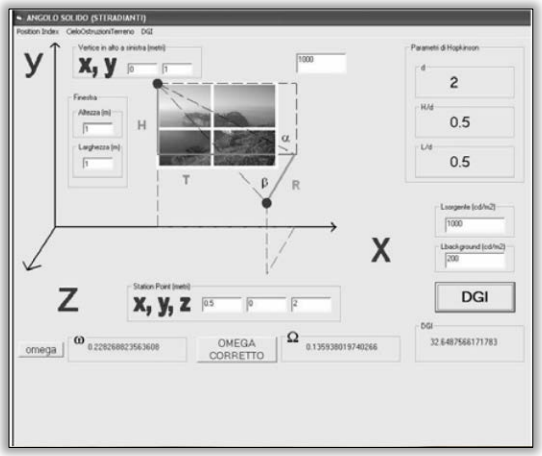
Recenti studi sperimentali dimostrano che l'abbagliamento dovuto ad una singola finestra dipende essenzialmente dalla luminanza della sorgente, evidenziando come l'abbagliamento possa essere considerato praticamente costante per tutti gli ambienti interni con finestre di dimensioni maggiori del 2% della superficie del pavimento e quindi esclusivamente variabile in funzione della luminanza della sorgente e del fattore medio di riflessione dell'ambiente interno.

Per il calcolo dell'angolo solido sotteso tra il punto di misura e la sorgente ci avvaliamo della seguente relazione:

- Nel caso di un cono di apertura α, la misura dell'angolo solido Ω rispetto al vertice è uguale a:

$$\Omega = 2\pi \left(1 - \cos \frac{\alpha}{2}\right).$$

Questa ci dice che se il punto di misura si trova ad una distanza di 2 metri da una finestra di dimensione 1x1, il raggio sotteso nel punto di misura sarà di 24° a cui corrisponde un angolo solido $\Omega = 2\pi(1-\cos(12°))=0.136$ sr



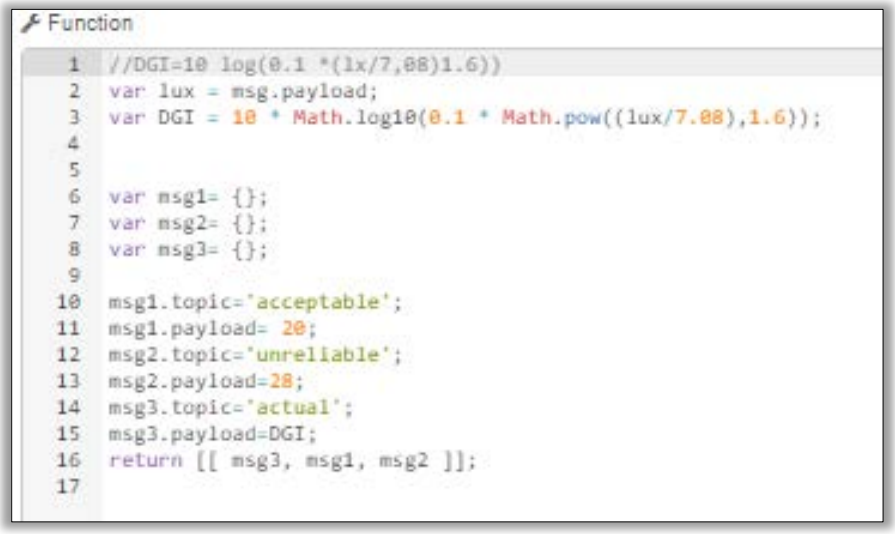
In modo analogo è possibile determinare l'angolo solido sotteso attraverso i software disponibili online.
Nel nostro caso l'angolo solido sotteso dalla sorgente è pari ad $\Omega = 0{,}143$.

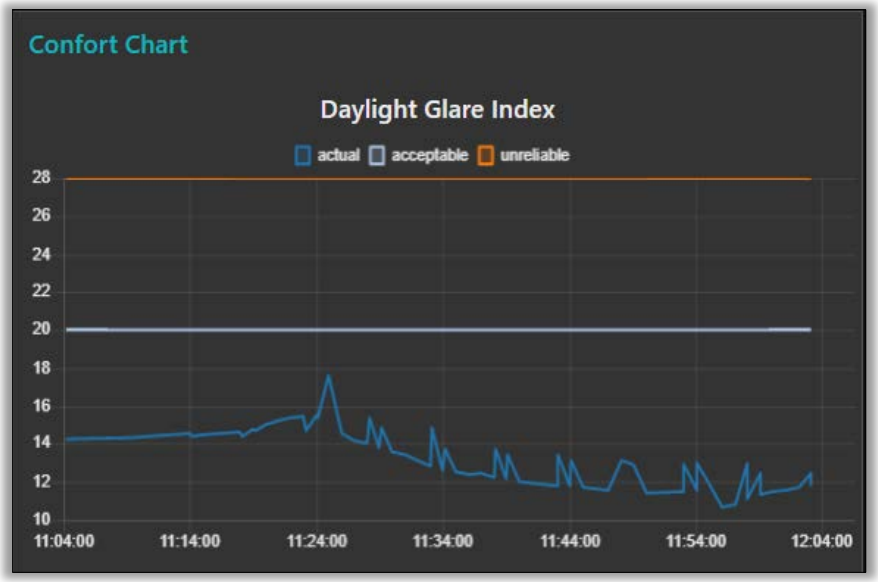
Considerando gli altri contributi di illuminamento trascurabili, avremo che a 500lx misurati dal nostro luxometro posto ad una distanza di 2 metri si otterrà un DGI pari ad:

$$DGI = 10 \log (0.48 \times (70^{1.6} \times 0.143^{0.8})) = 19.58$$

Per ottenere dei valori di abbagliamento percettibili il valore dei lux misurati deve superare i 700 lx che produce un DGI=21,81.

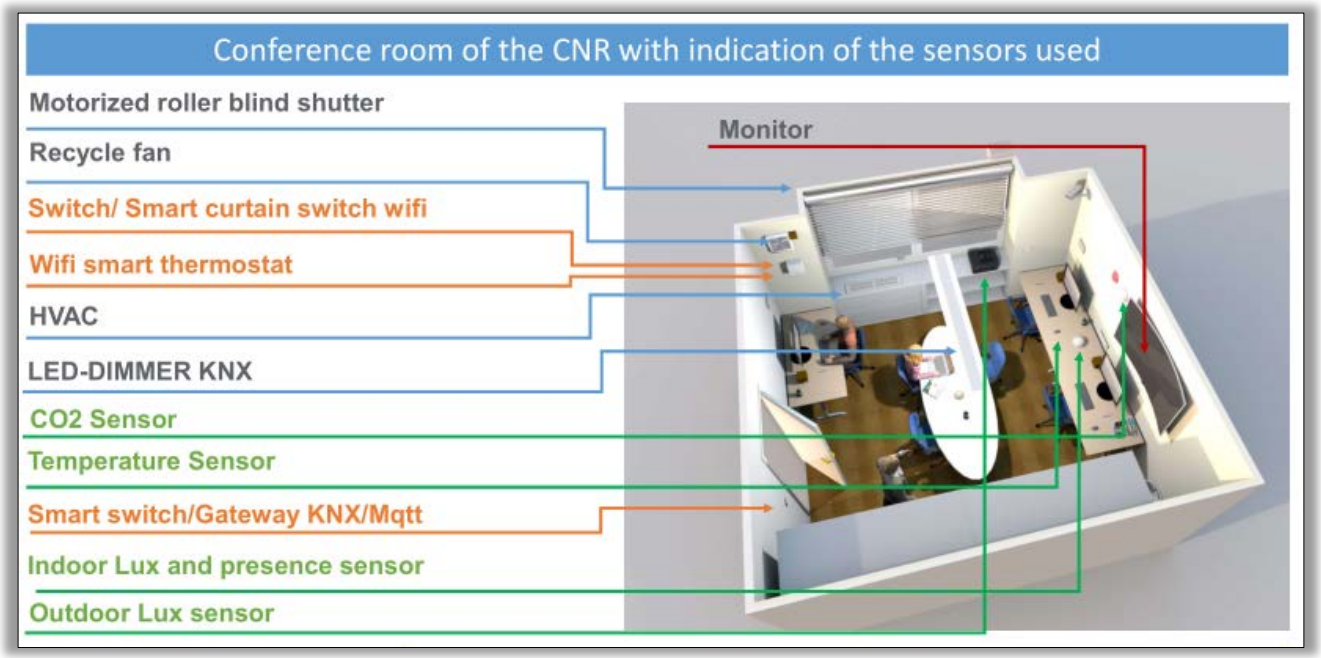
Per il calcolo dell'indice di abbagliamento ci siamo quindi avvalsi della relazione precedentemente elaborata:





Hardware ed architettura del sistema

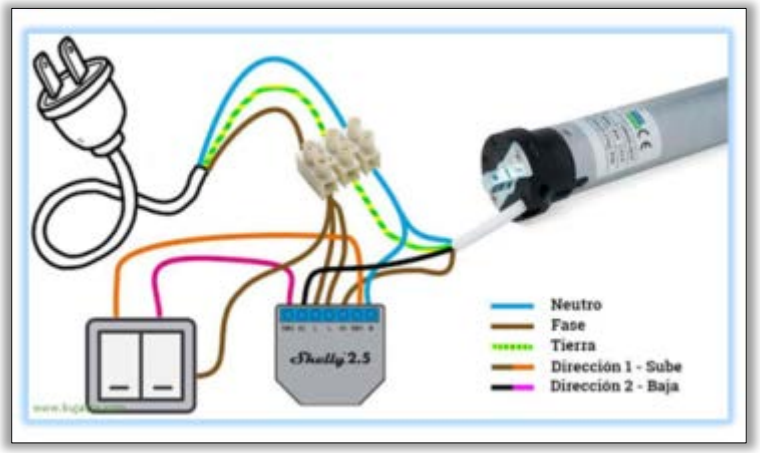
L’ambiente di test utilizzato per realizzare il sistema è raffigurato di seguito:



Gli attuatori utilizzati sono:

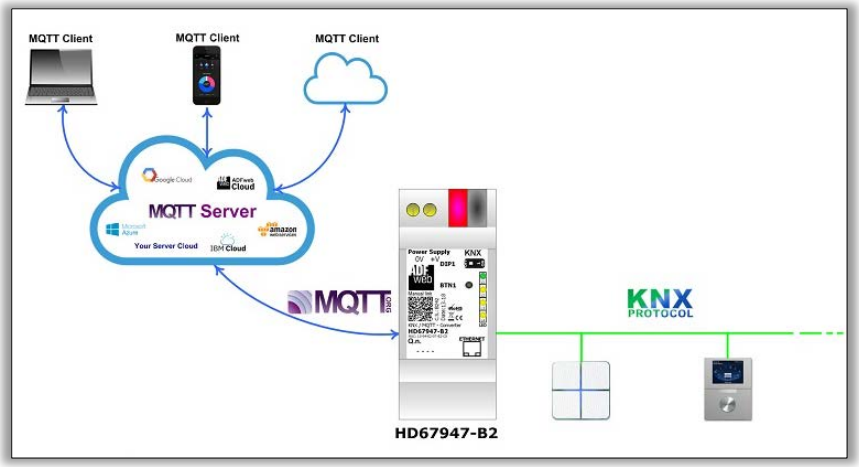
- Tapparella motorizzata

- Schelly 2.5 per il controllo wify della tapparella



- Set di lampadine philips hue per il controllo della luce artificiale, nella prima versione

- IoT Gateway KNX to MQTT per il controllo dei corpi luminosi preinstallati:



- Termostato Beca Serie 1000 Due tubi per climatizzazione Fan Coil di retroilluminazione bianca con connessione WiFi

- Ventola di aspirazione Xpelair con dispositivo Shelly switch per il controllo:

- Sensore di presenza Heiman HS1MS

- Sensore di CO^2 Heiman HS1CA-E