Bosch Security Systems

Relazione tecnica
Video Systems, CCTV

BOSCH
Tecnologia per la vita

Documento tecnico:

Guida rapida Intelligent Video Analysis

Redatto da: Bosch Security Systems s.p.a.
Versione: 1.0
Data: Marzo 2016

BOSCH
Tecnologia per la vita

Sommario

1. Premessa

Il presente documento è stato redatto per illustrare brevemente il funzionamento di Intelligent Video Analysis, l'analisi video intelligente built-in di Bosch, che per comodità sarà identificata come IVA; si noti bene che le telecamere, gli encoder ed i decoder Bosch saranno identificati come dispositivi video.
Tutte le definizioni e prestazioni si intendono riferite alla versione 6.10 di IVA.
Per ogni ulteriore informazione e approfondimento si rimanda al manuale IVA Task Script Language 1.6.

2. Il linguaggio script delle attività IVA

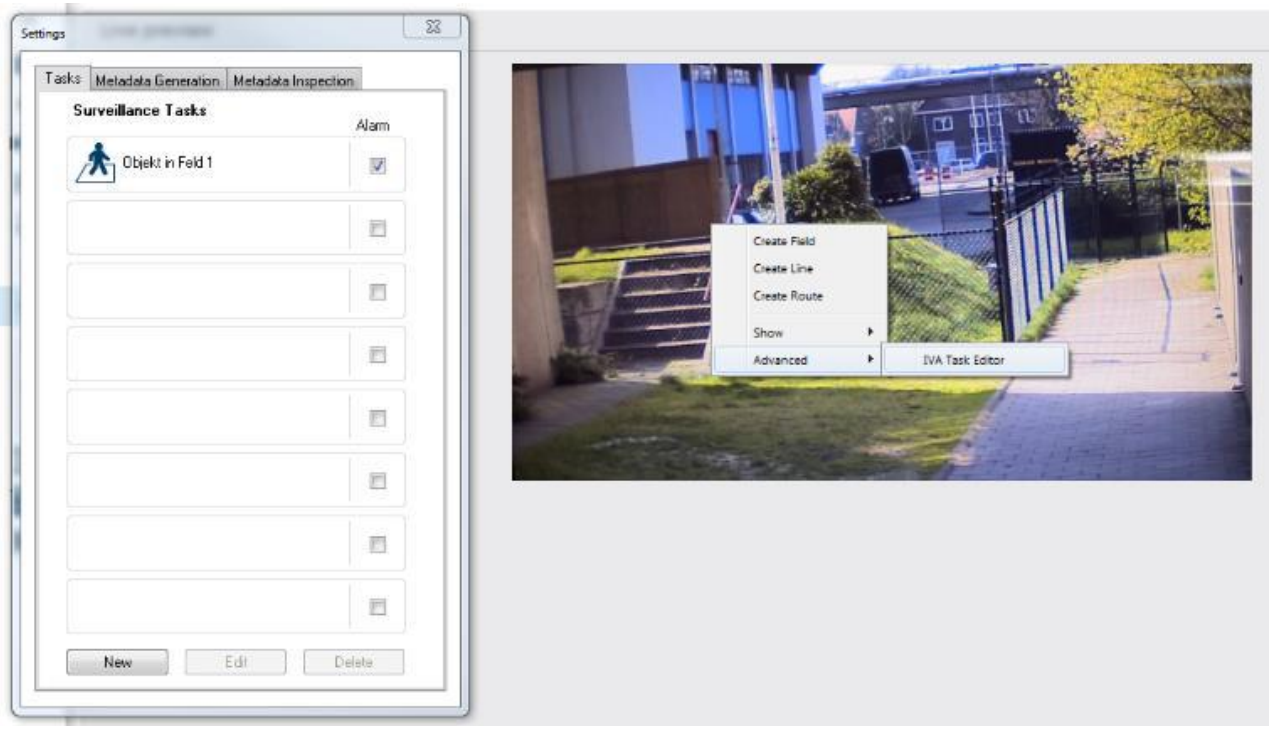
Comprendere il linguaggio script di Intelligent Video Analysis diventa di fondamentale importanza alla luce della possibilità di combinare condizioni ed eventi per crearne di nuovi ad hoc per tutte le necessità. Tramite poche nozioni di base e mirate si potrà padroneggiare la stesura di uno script definito dall'utente, che renderà Intelligent Video Analysis uno strumento ancora più potente ed utile.

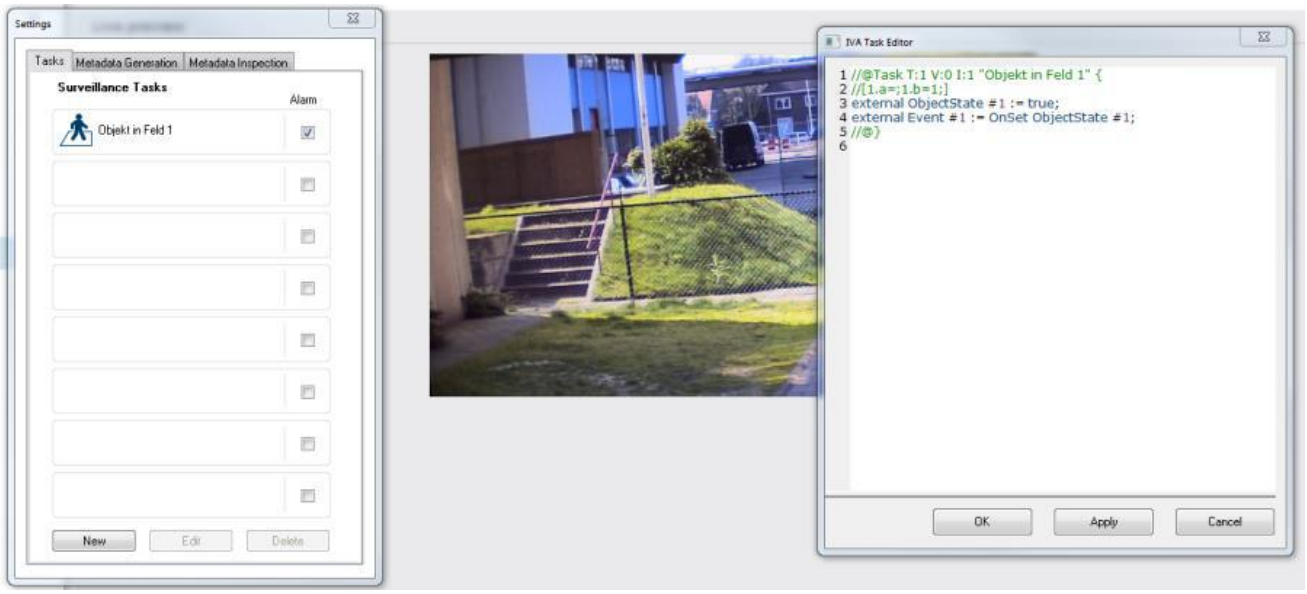
Gli script delle attività IVA sono utilizzati implicitamente ogni volta che un'attività IVA è configurata tramite l'interfaccia grafica utente. Tuttavia è possibile configurare manualmente gli script IVA, pratica suggerita quando le attività predefinite non siano sufficienti. Gli script possono essere copiati dall'editor ed incollati in esso per il backup e la sostituzione delle attività configurate.

Si consiglia di utilizzare il linguaggio script delle funzioni IVA quando siano necessarie più di otto attività di allarme: sono configurabili 16 attività di allarme esterne tramite script di attività IVA. E' utile anche per ottimizzare linee e campi, per combinare linee in ottica di utilizzare un contatore o quando siano necessarie logiche di eventi predefiniti.

Si consiglia di utilizzare la seguente metodologia per avvicinarsi all'utilizzo del linguaggio script delle attività IVA. Per prima cosa si definiscano tutte le attività tramite il task wizard per quanto possibile. Si acceda poi all'editor degli script di attività IVA: lì si possono trovare le attività generate automaticamente con tutti i dettagli. Si effettuino i propri cambiamenti ed infine si modifichino le attività predefinite in attività definite via script in modo che un utilizzo accidentale del task wizard non sovrascriva i propri cambiamenti.

Per accedere al linguaggio script delle attività IVA è sufficiente entrare nella configurazione IVA in Bosch Configuration Manager ed aprire la pagina delle attività. Si clicchi con il tasto destro del mouse sul video e si selezioni Advanced -> IVA Task Editor. Apparirà un'altra finestra con lo script delle attività IVA attuali:





3. Attività

Un'attività "task" è formata nella sua forma più semplice da una primitiva, da uno stato dell'oggetto o un'interazione con la primitiva dell'attività e da eventuali filtri da applicare alle proprietà dell'oggetto.

Attività IVA
Rilevazione di oggetti
Attraversamento linee
Oggetto in campo
Ingresso in campo
Uscita dal campo
Seguire percorsi
Indugiare in modo sospetto
Oggetto rimosso
Oggetto inattivo
Modifica delle condizioni
Ricerca di somiglianza
Conteggio
Conteggio delle persone con BEV
Rilevamento in presenza di folle
Manomissione

Primitive di attività IVA
Attività di flusso IVA
Flusso nella scena
Flusso opposto nella scena
Rilevamento in presenza di folle
Manomissione
FlowDetector
CrowdDensityEstimator
MotionDetector

Filtri
Area dell'oggetto
Proporzioni
Velocità
Direzione
Colore
Classe dell'oggetto

Alcuni semplici esempi di attività:

```
Line #1 := { Point(26,  65) Point(82,  122)
                                    DebounceTime(0.50)
Direction(1) };
            external Event#1  :={CrossedLine#1};
```

```
Loitering #1 := {
        Radius (5)
        Time (10)
    };
    external ObjectState #1  := IsLoitering #1;
```

```
Field #1 := {
      Point (10, 10)  Point (10, 50)
      Point (50, 50)  Point (50,10)
};
external Event #1 := {
        EnteredField #1 before LeftField #1
        where first.oid == second.oid
};
```

4. Eventi e stati

Gli **stati** descrivono relazioni temporali (per esempio spaziali), proprietà degli oggetti e tentativi di manomissione.

- Possono essere definiti in totale fino a 16 stati *esterni*
- Possono essere definiti in totale fino a 32 stati

Gli **eventi** permettono di esprimere relazioni temporali tra gli oggetti.

- Possono essere mostrati sull'interfaccia grafica utente fino ad 8 eventi *esterni*
- Possono essere definiti in totale fino a 16 eventi *esterni*
- Possono essere definiti in totale fino a 32 eventi

Gli eventi e gli stati possono essere interni od esterni. Solo gli eventi esterni e gli stati sono mostrati in output dall'IVA; gli eventi e gli stati dichiarati esterni, che sono output dell'Alarm Task Engine, sono definiti come regole.
Gli stati possono essere legati alle proprietà degli oggetti, possono essere stati di manomissione (gli algoritmi di IVA possono rilevare infatti tentativi di manomissione delle telecamere) o stati definiti dagli utenti.
Gli stati definiti dagli utenti sono sempre booleani, ossia assumono i valori `false` e `true`. Uno stato definito dall'utente può non avere attributi oppure avere l'ID dell'oggetto come attributo. Il primo stato è chiamato `SimpleState`, il secondo invece `ObjectState`.
Gli eventi possono essere specifici degli oggetti, eventi di cambiamento di stato, eventi predefiniti o definiti dagli utenti.
La funzione principale dell'algoritmo di IVA è la rilevazione degli oggetti e il loro tracciamento. Oltre alla posizione e alle altre proprietà degli oggetti rilevati in tempo reale, l'algoritmo notifica gli eventi di base degli oggetti. Quando l'algoritmo rileva un nuovo oggetto, un evento `Appeared` è azionato. In modo corrispondente un evento `Disappeared` è inviato non appena un oggetto è perso. `Idled` e `Removed` sono casi speciali di oggetti che scompaiono.
Per rilevare per esempio cambiamenti nell'aspetto di un oggetto, sono introdotte le parole chiave `OnChange`, `OnSet` e `OnClear`. Possono essere combinate con uno stato definito dall'utente e azionare un evento se lo stato modifica il suo valore, se lo stato diventa `true`, o se lo stato diventa `false`.
L'utente può definire nuovi eventi utilizzando relazioni temporali tra gli stessi. Oltre a relazioni temporali, possono essere formulate condizioni addizionali per limitare gli eventi in modo ulteriore.

Eventi degli oggetti
CrossedLine
EnteredField
LeftField
FollowedRoute
Appeared
Disappeared
Idled
Removed

Stati degli oggetti
InsideField
ObjectsInField
IsLoitering
SimilarToColor
DetectedFlow
EstimatedCrowdDensity
HasDirection
HasVelocity
HasAspectRatio
HasObjectSize
HasColor

Proprietà degli oggetti
Direction
Velocity
AspectRatio
ObjectSize

Stati di manomissione
SignalTooNoisy
SignalTooDark
SignalTooBright
SignalLoss
GlobalChange
RefImageCheckFailed

5. Condizioni e loro combinazioni

Per quanto concerne gli stati, possono essere combinate molteplici condizioni. La sintassi delle condizioni coordinate è la seguente:

```
<condition>  := <condition> and <condition>

<condition>  := <condition> && <condition>
```

In modo similare le condizioni disgiuntive sono scritte così:

```
<condition>  := <condition> or <condition>

<condition>  := <condition> || <condition>
```

La negazione delle condizioni è la seguente:

```
<condition>  := not <condition>

<condition>  := !<condition>
```

Le operazioni più interessanti sugli eventi sono le relazioni temporali. La parola chiave `before` combina due eventi nel modo seguente. Se il secondo evento è azionato e il primo evento è stato azionato precedentemente, un altro evento con gli stessi attributi del secondo è fatto scattare. Nella forma più semplice la sintassi è la seguente:

```
<event>  := { <event> before <event> }
```

Aggiungendo la parola chiave `not` , è inoltre possibile controllare se il primo evento non sia accaduto prima del secondo evento nell'intervallo di tempo specificato (l'intervallo di tempo è ancora una volta opzionale):

```
<event>  := { <event> not before (<from>,<to>) <event> }
```

La parola chiave `or` può essere utilizzata per azionare un evento se o l'evento A o l'evento B sia accaduto.

```
<event>  := <event> or <event>
```

Spesso è necessario limitare gli eventi con i loro attributi e le proprietà degli oggetti coinvolti. Nel linguaggio script delle attività IVA, si può fare questo tramite la proposizione `where`.

```
<event>  := { <event> where <condition> }
```

L'Alarm Task Engine si azionerà solo se la `<condition>` è soddisfatta nel momento in cui `<event>` è accaduto.

6. Esempi pratici

In questo capitolo sono proposti e spiegati degli esempi pratici di script IVA con le funzioni più utilizzate; si consiglia di approfondire le argomentazioni documentandosi mediante gli appositi manuali.

Nota bene: negli esempi seguenti i commenti agli script saranno evidenziati in verde e seguiranno il simbolo // (doppio slash), in modo da essere ignorati dall'Alarm Task Engine.

6.1 Attraversamento linee

```
// Definizione delle primitive delle attività

Line #1 := { Point(103,  30) Point(159,  77) };
Line #2 := { Point(26,  65) Point(82,  122)
             DebounceTime(0.50)  Direction(1) };

// Definizione di una attività di allarme mostrata nella GUI
//@Task T:2 V:0 I:1 "Attraversamento linea 1" {
//[1.a=s1:1;1.b=1;1.c=32;1.d=31;4.a=i:1;]
external Event#1  :={CrossedLine#1};
//@}

//Definizione di un'attività definita ad hoc mostrata nella GUI
//@Task T:0 V:0 I:2 "Attraversamento linea 2" {
external Event#2  :={CrossedLine#2};
//@}

//Definizione di un'attività allarme non mostrata nella GUI
external Event#3  :={CrossedLine#2};
```

L'esempio mostrato definisce due linee tramite due punti di delimitazione e, nel caso della seconda, anche tramite i parametri opzionali `DebounceTIme` e `Direction`. Il primo parametro è utile nell'eliminare gli errori di posizionamento nel tracking degli oggetti o gli allarmi multipli di oggetti che si muovono lunga la linea; il secondo invece permette di scegliere se un oggetto che attraversi la linea azioni l'evento (direzione 0) o solo gli oggetti che attraversano da destra a sinistra (direzione 1) o nella direzione opposta (direzione 2) siano rilevanti.
I tre eventi esterni definiti sono semplici eventi di attraversamento linea. Si noti che l'evento 3 non viene mostrato nell'interfaccia guidata utente poiché non presenta la dizione //@ Task T:x V:y I:z.

Per definire la task wizard si utilizzino le seguenti indicazioni:

```
//@ Task T:x V:y I:z
```

T:x indica il # dell'attività (Oggetto in campo, attraversamento linee… guardare l'icona per quella corretta!).
T:0 indica una task creata ad hoc. Utilizzarlo per i propri script per evitare che i task wizard la sovrascrivano.

V:0 è il numero della versione, attualmente sempre 0; I:z è lo slot nella pagina delle attività.
I:1 indica lo slot occupato nella lista delle attività sull'interfaccia utente. Affinché esso diventi rosso in caso di allarmi, l'evento/stato esterno definito nell'attività deve avere lo stesso numero dell'attività stessa.
[1.a=s1:1;...] indica i valori del task wizard.

6.2 Attraversamento linea con filtri

```
//Definizione delle primitive dell'attività
Line #1  :=  { Point(103,  30) Point(159,  77)};

//@Task T:2 V:0 I:1 "Attraversamento linea 1"  {
//[1.a=s1:1;1.b=1;1.c=32;1.d=31;3.o=(b1:1,
b2:0.1,b3:500,  c1:1,c2:0.02,c3:34.00,d1:1,
d2:0.d3:27.78,e1:1,e2:315,  e3:45,f1:1,f2:135,
f3:225);4.a=(c:2b19,i:1,pr:2);5.a=(c:1,p:1);
6.a=(d1:8,d2:33,r:2,u:1);]
ColorHistogram  #1:= { HSV(60,38,100,25)
        Similarity(75)  Outlier(75) };
external Event#1:={CrossedLine#1
        where ObjectSize within(0.1,500)
        and AspectRatio within(0.02,34.00)
        and Velocity within(0,27.78)
        and (Direction within(315,405)  or  Direction within(135,225))
        and SimilarToColor #1
        };
//@}
```

Sono definite una primitiva linea ed una `ColorHistogram`, che permette di confrontare i colori degli oggetti con i colori definiti dall'utente. I colori di base sono definiti nello spazio di colore HSV; `Similarity` specifica quanto debba essere simile un istogramma di colore per essere considerato come corrispondente e `Outlier` permette corrispondenze parziali tra i colori definiti dall'utente e l'istogramma cromatico dell'oggetto.

L'evento esterno definito è un evento di attraversamento linea con filtri applicati all'oggetto tramite la parola `where`: se l'oggetto che attraversa la linea non rispetta tutti i filtri indicati l'evento non è azionato. Tramite l'utilizzo di `and` si combinano i filtri, che devono essere rispettati contemporaneamente nell'istante dell'evento perché sia azionato. Si noti l'utilizzo di `or` nel caso della direzione: significa che una qualsiasi delle due indicate può rendere vera la condizione tra parentesi.

I filtri riguardano le dimensioni dell'oggetto in [m²] e le sue proporzioni (rapporto altezza su larghezza), la velocità dell'oggetto espressa in [m/s], la direzione in cui si sta muovendo espressa in gradi e il suo istogramma cromatico.

6.3 Campi

```
//Definizione delle primitive dell'attività

Field  #1 := { Point(56,  24) Point(106,  24)
               Point(106,  74)};
Field  #2 := { Point(56,  24) Point(106,  24)
                Point(106,  74) Point(56, 74)
                DebounceTime(0.50)
                ObjectSet(BoundingBox)
                SetRelation(Covering)};


//@Task T:1 V:0 I:1 "Oggetto in campo 1"  {
//[1.a=1;1.b=1;a=1:1;]
external ObjectState  #1 := InsideField  #1;
//@}
```

E' definita una primitiva campo tramite i punti di vertice che lo delimitano. I parametri opzionali sono il tempo di antirimbalzo, `ObjectSet` e `BoudingBox`. Gli ultimi due parametri specificano in particolare quali parte dell'oggetto debbano essere considerate per determinare se un oggetto sia dentro o fuori dal campo. In tal modo `<objectset>` può essere configurato a `BaryCenter` o `BoundingBox`, con il primo come valore di default. `<relation>` può assumere i valori `Intersection` e `Covering`, con il primo come valore di default. Per esempio, se `BoundingBox` e `Covering` sono selezionate, il bounding box dell'oggetto deve essere completamente all'interno del poligono specificato per essere considerato all'interno. Se `<objectset>` è fissato a `BaryCenter`, entrambe le opzioni di `<relation>` sfociano nello stesso comportamento, dato che l'oggetto fissato consiste solo di un unico punto.
L'evento esterno è azionato se l'oggetto si trova all'interno del campo.

6.4 Percorsi

```
//Definizione delle primitive dell'attività

Route  #1 :=  { Point(34,  111) Distance(5)
                Point(92,  125) Distance(5)
                Point(140,  104) Distance(5)
                Point(143,  72) Distance(5)
                MinPercentage(80) MaxGap(10) };

//@Task T:6 V:0 I:1 "Seguire percorso 1"{
//[1.a=id:1;1.b=1;3.a=i:1;]
external Event  #1 := {FollowedRoute  #1};
```

```
//@}
```

E' definita in primis una primitiva percorso tramite la sequenza di punti che lo costituisce, `Distance`, `MinPercentage` e `MaxGap`. `Distance` indica una tolleranza spaziale tale da trasformare una linea in una striscia. Se è stata solcata più della `MinPercentage` dell'intera striscia e il salto più grande tra due parti solcate (compresi i salti all'inizio e alla fine della striscia) è inferiore a `MaxGap`, allora l'evento `FollowedRoute #1` è azionato.

6.5 Combinare linee per il conteggio

```
//Definizione delle primitive dell'attività
Line  #1 := { Point(0, 90) Point(60, 90)
              DebounceTime(0.10)  Direction(1) };
Line #2 := { Point(60, 90) Point(90,60)
             DebounceTime(0.10) Direction(1) };
Line #3 := { Point(90, 60) Point(90, 0)
             DebounceTime(0.10) Direction(1) };

//@Task T:0 V:0 I:1 "Contatore 1"  {
Event#1 :=CrossedLine#1 or CrossedLine#2
          or CrossedLine#3;
external Counter#1  := { Event#1  Text("Corner Count1:")
                         TopLeft(4,4) Mode(Wraparound)
                         within(0,999999999)  };
//@}
```

Sono definite tre linee spezzate, una adiacente all'altra. E' definito poi un evento di attraversamento linee, tale, grazie alla parola `or`, da essere azionato qualsiasi delle linee sia attraversata. E' poi definito un evento esterno contatore azionato dall'attraversamento linee, che mostra sul video in alto a sinistra la scritta "Corner Count1:" ed il valore corrente. La modalità selezionata nell'esempio indica che, quando si raggiunge il numero massimo, il conteggio riparte da 0.
In questo caso si può notare l'efficacia della definizione di un'attività via script rispetto al wizard delle attività. Dato che si ha interesse a contare gli oggetti che attraversano in una certa direzione un insieme di linee spezzate, unite a formare una sorta di confine continuo, tramite il wizard delle attività avrei difficoltà a creare tre linee che non lascino alcun spazio tra di esse. Tramite lo script invece tutto questo risulta immediato, basta creare le linee utilizzando come punto di inizio di una linea le coordinate del punto terminale della linea precedente.

6.6 Attraversamento linee a velocità diverse

```
//Definizione delle primitive dell'attività
Line #1  := {Point(50,0)  Point(50,144)};
```

```
Line #2  := {Point(120,0) Point(120,144)};

//Definizione degli eventi di attraversamento linea
Event#11  :={CrossedLine#1  where
              Velocity  within(13.89, 19.44) };
Event#12  :={CrossedLine#2  where
              Velocity  within(2.778,5.556)  };

//Combinazione di eventi di attraversamento linea
//@Task T:0 V:0 I:1 "Controllo velocità di due linee" {
external Event#1  :={Event#11 before Event#12
                        where first.oid==second.oid};
//@}
```

Oltre alla consueta definizione delle primitive linee vista negli esempi precedenti, sono definiti due eventi interni di attraversamento linea con filtri di velocità grazie alle parole `where`, che indica una condizione, e `within`, che permette di identificare un intervallo. Si noti che la velocità è espressa in metri al secondo, pertanto il primo intervallo è compreso tra 50-70 [km/h] e il secondo tra 10-30 [km/h]. E' definito poi un evento esterno come combinazione temporale tra i due eventi precedenti: nello specifico l'evento 11 deve accadere prima dell'evento 12 e per azionare l'evento esterno deve essere il medesimo oggetto rilevato ad attraversare entrambe le linee alle suddette velocità. Si noti l'utilizzo della dizione "`first.oid==second.oid`", che permette di accedere alla lista degli attributi dei due eventi e di ottenere che sia lo stesso oggetto a far scattare gli eventi. In sostanza l'esempio rileva oggetti che decelerano all'interno di uno spazio definito dalle due linee.

6.7 Fermarsi in un'area dopo attraversamento linea

```
//Definizione delle primitive dell'attività
Line #1 := { Point(130, 0) Point(130, 144) };
Field #2 := { Point(50, 0) Point(115, 0)
              Point(115, 144) Point(50, 144)};

//@Task T:0 V:0 I:1 "Loitering dopo attraversamento linea" {
Event#11 :={CrossedLine  #1};
Loitering #13 := {Radius (15) Time (10) };
ObjectState #14 := InsideField #2 and IsLoitering #13;
external Event #1 := {Event #11 before OnSet ObjectState #14
                        where first.oid==second.oid};
//@}
```

Notiamo le consuete definizioni di primitive linee e campo ed in più la definizione della primitiva `Loitering` tramite la tolleranza spaziale in [m] ed il tempo minimo da considerare per far azionare lo stato `IsLoitering`. Lo stato è un `ObjectState` perchè l'oggetto è rilevato e gli è assegnato un `Oid`. L'unico evento che aziona un allarme è l'evento `external` numero 1, che prevede un attraversamento linea e subito dopo l'essere in campo e fermarsi in maniera sospetta. Come nell'esempio precedente tramite l'attributo `oid` si può fare in modo che sia il medesimo oggetto ad azionare l'evento di attraversamento linea e lo stato dell'oggetto.

6.8 Allarme quando un oggetto tocca un campo

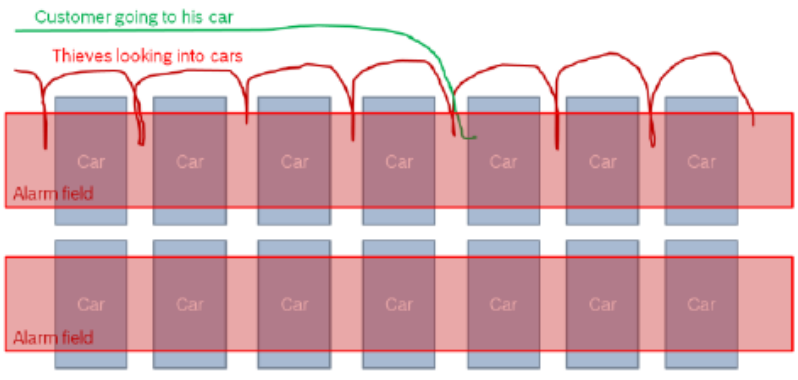
```
//Definizione delle primitive dell'attività
Field  #1 := { Point(100, 0) Point(175, 0)
               Point(175, 144) Point(100, 144)
               ObjectSet(BoundingBox)};

//@Task T:0 V:0 I:1 "Oggetto tocca il campo" {
ObjectState  #1 := InsideField  #1;
external Event #1 := OnSet ObjectState #1;
//@}

//In alternativa, gli oggetti devono arrivare da fuori il campo
//@Task T:0 V:0 I:2 "Oggetto tocca il campo 2"  {
external Event #2 := EnteredField #1;
//@}
```

Nella definizione della primitiva campo si può notare il parametro `ObjectSet` impostato al valore `BoundingBox`, che serve a far scattare l'allarme non tramite il centro di gravità dell'oggetto ma al tocco dello stesso col campo, ossia tramite l'intersezione tra il bounding box dell'oggetto e il limite del campo. Sono poi definiti due eventi, uno in cui l'oggetto deve trovarsi all'interno del campo e l'altro in cui l'oggetto deve arrivare fuori da esso. E' una differenza sottile ma importante, soprattutto in funzione di come è stato definito il campo e se c'è abbastanza spazio tra esso e il margine dell'immagine video perchè sia rilevato l'evento di ingresso nel campo. Poichè l'allarme scatta quando il bounding box di un oggetto interseca un campo, lo script dell'esempio precedente può essere utilizzato in applicazioni in cui è necessario proteggere quadri ed oggetti esposti, barriere da non scavalcare, zone sterili vicino a posti affollatti e ogniqualvolta vi chiediate perchè una persona che allunga una mano all'interno di un campo di allarme non lo faccia scattare.

6.9 Allarme quando un oggetto entra in un'area almeno 4 volte

```
//Definizione delle primitive dell'attività
Resolution  := { Min(-1, -1) Max(1, 1) };
Field  #1 := { Point(- 0.244, -0.922) Point(0.325, -0.944)
               Point(0.350, 0.944) Point(-0.231, 0.944)
               DebounceTime(0.50) };
//@Task T:0 V:0 I:1 "Rilevamento ladri" {
Event  #11  :={EnteredField  #1 before EnteredField  #1
                                  where first.oid==second.oid};
Event  #12  :={Event #11 before EnteredField  #1
                                  where first.oid==second.oid};
external Event#1  :={Event #12 before EnteredField  #1
                                  where first.oid==second.oid};
//@}
```

Si noti l'utilizzo della parola `Resolution` che serve a normalizzare le coordinate all'interno dello script. La coordinata `Min` indica il punto in alto a sinistra sul video e la coordinata `Max` il punto in basso a destra. In tal modo si crea un intervallo di valori conosciuto in cui devono rientrare tutti i punti nelle definizioni delle primitive. Si ricorda che la normalizzazione dello script deve essere riportata subito in alto come prima riga dello stesso, e vale per tutte le primitive definite successivamente. Sono definiti poi in sequenza una serie di eventi concatenati di ingresso nel campo in cui l'oggetto che aziona l'evento deve essere il medesimo per tutti. Una possibile applicazione per questo script potrebbe essere quella di proteggere i parcheggi da persone intenzionate a rubare automobili.



6.10 Allarme quando almeno due oggetti sono all'interno di un'area

```
//Definizione di primitive dell'attività
```

```
Resolution := { Min(-1, -1) Max(1,1)
Field  #1 := { Point(-0.656, -0.700) Point(0.350, -0.767)
               Point(0.469, 0.644) Point(-0.694, 0.644)
               DebounceTime(0.10)  };

//@Task T:0 V:0 I:1 “Allarme per almeno due oggetti“ {
external ObjectState  #1:= ObjectsInField#1  within (2,*);
//@}
```

Notiamo ancora una volta l’utilizzo delle coordinate normalizzate ad inizio script. Questo semplice script aziona un allarme se sono presenti nel campo definito almeno due oggetti. Si può osservare il simbolo asterisco * all’interno dell’intervallo `within`: il simbolo indica un’intervallo superiore illimitato, in tal modo si può realizzare la condizione “almeno due oggetti” nel campo.

6.11 Contare il numero di oggetti in un’area

```
//Definizione delle primitive dell’attività
Resolution := { Min(-1, -1) Max(1,1) };
Field   #1 := { Point(-0.656, -0.700) Point(0.350, -0.767)
                Point(0.469, 0.644) Point(-0.694, 0.644)
                DebounceTime(0.10) };

//@Task T:0 V:0 I:1 “Conteggio oggetti in campo“ {
external Counter#1 := { ObjectsInField#1 Text(“Counter:”)
                        TopLeft(-0.975,-0.844) };
//@
```

E’ utilizzata ancora una volta la normalizzazione delle coordinate dello script. E’ definito un contatore che si basa su `ObjectsinField`, che restituisce il numero di oggetti rilevati in quell’istante all’interno del campo definito. Il numero di oggetti è mostrato in alto a sinistra dopo la scritta “Counter:”.

6.12 Allarme se una fila è vuota e l’altra è composta da almeno 3 persone

```
//Definizione delle primitive dell’attività
```

```
Resolution:= { Min(-1, -1) Max(1, 1) };
Field #1:= { Point(-0.150, -0.700) Point(0.250, -0.700)
             Point(0.250, 0.600) Point(-0.150, 0.600)
             DebounceTime(0.10)  };
Field #2:= { Point(-0.600, -0.700) Point(-0.200, -0.700)
             Point(-0.200, 0.600) Point(-0.600, 0.600)
             DebounceTime(0.10) };

//@Task T:0 V:0 I:1 “Solo una fila” {
ObjectState  #1:= ObjectsInField#1  within (3,*);
external Event  #1:= {OnSet ObjectState  #1
                      where ObjectsInField#2 within (0,0)};
//@}

//Per vedere i risultati del controllo sull’assenza di oggetti, utilizzare la seguente:
//@Task T:0 V:0 I:2 “Nessun oggetto” {
external SimpleState  #2:= ObjectsInField#2  within (0,0);
//@}
```

Le coordinate dei punti sono espresse grazie all’uso di `Resolution`. Sono definiti due campi rettangolari che rappresentano le file. E’ definito poi uno stato dell’oggetto tale che siano presenti nel campo almeno tre oggetti (notare l’utilizzo dell’asterisco per indicare un intervallo superiore illimitato). L’evento esterno combina tale stato dell’oggetto con la contemporanea assenza di oggetti rilevati (nella nostro intento persone) nel campo due tramite la condizione `where`. Se si vuole semplicemente controllare l’assenza di oggetti nel campo 2, si può utilizzare l’ultima espressione. Si noti bene che è utilizzato un `SimpleState` anziché un `ObjectState` poiché non è coinvolto nessun oggetto.