UNIVERSITÀ DI BRESCIA
FACOLTÀ DI INGEGNERIA
Dipartimento di Ingegneria dell'Informazione

Laboratorio di Robotica Avanzata
Advanced Robotics Laboratory

Corso di Robot industriali e di servizio
(Prof. Riccardo Cassinis)

Realizzazione di un programma dimostrativo su robot Kawasaki RS03N

Elaborato di esame di:

Manuel Pagnoni,
Alessandro Zanotti

Consegnato il:

30 Agosto 2012

Sommario

Il lavoro scelto dai candidati consiste nella realizzazione di alcune figure geometriche semplici utilizzando il manipolatore Kawasaki presente nel laboratorio ARL. Tramite la programmazione di quest'ultimo, utilizzando un linguaggio di programmazione proprietario del robot in questione (AS), il manipolatore, con l'ausilio dell'apposito tavolo di lavoro, si occupa della raccolta di alcune sfere di legno e del posizionamento delle stesse secondo uno schema geometrico sul piano di lavoro. Il lavoro svolto è dunque un esempio dimostrativo pratico di come un manipolatore industriale, attraverso la programmazione e il suo funzionamento, prende dei pezzi e li posiziona in un ordine logico (Pick and Place).

1. Introduzione

L'obbiettivo di questo documento è la descrizione e la realizzazione di un programma dimostrativo del robot Kawasaki RS03N situato nel laboratorio di robotica avanzata ARL della Facoltà di Ingegneria dell'Università degli studi di Brescia.

Tale programma sarà in grado di istruire il robot in modo che compia azioni e movimenti in grado di prendere delle sferette in legno da una posizione predeterminata e riporle in un punto preciso del piano di lavoro in modo da "disegnare" semplici figure geometriche per uno scopo didattico-dimostrativo.

Il piano di lavoro è costituito da una matrice di 7 x 7 posizioni, composta da tappi di bottiglia di plastica rovesciati ed equispaziati lungo i due assi che formano il piano XY, sul quale verranno posizionate in seguito le palline.

Questo tipo di operazione è chiamata "Pick and Place" in linguaggio tecnico ed è ampiamente diffusa in ambito industriale soprattutto in attività quali il montaggio di schede elettroniche o l'assemblaggio di parti meccaniche di un pezzo.

2. Breve descrizione del robot Kawasaki RS03N

Il robot Kawasaki RS03N è un manipolatore a 6 gradi di libertà rotazionali e viene più comunemente chiamato robot articolato o antropomorfo in quanto rispecchia il funzionamento del braccio umano.

Kawasaki fornisce il robot assieme ad altri elementi necessari per il suo utilizzo. Il sistema completo è costituito da:

Fig. 1 - Il manipolatore

- Manipolatore RS03N
- Controller del robot
- Teach pendant

La figura a lato (Fig. 1) mostra il manipolatore RS03N. Grazie al suo design compatto unito all'elevata velocità dei movimenti dei giunti l'RS03N è il manipolatore ideale per una larga gamma di applicazioni. Nell'immagine è possibile anche individuare i vari giunti e le relative parti meccaniche che li collegano (link).

Il manipolatore ha una massa di 20 Kg e può essere vincolato a terra, soffitto o pareti a seconda delle esigenze dell'utilizzatore. Possiede un carico massimo di 3 Kg, una velocità massima di 6000 mm/s ed un valore di precisione in ripetibilità di 0.02 mm.

La figura 2 mostra la proiezione ortogonale dei principali pezzi del manipolatore. Sempre nella stessa figura è possibile osservare il campo di lavoro del robot ossia il luogo dei punti dello spazio raggiungibili dall'**end effector**.

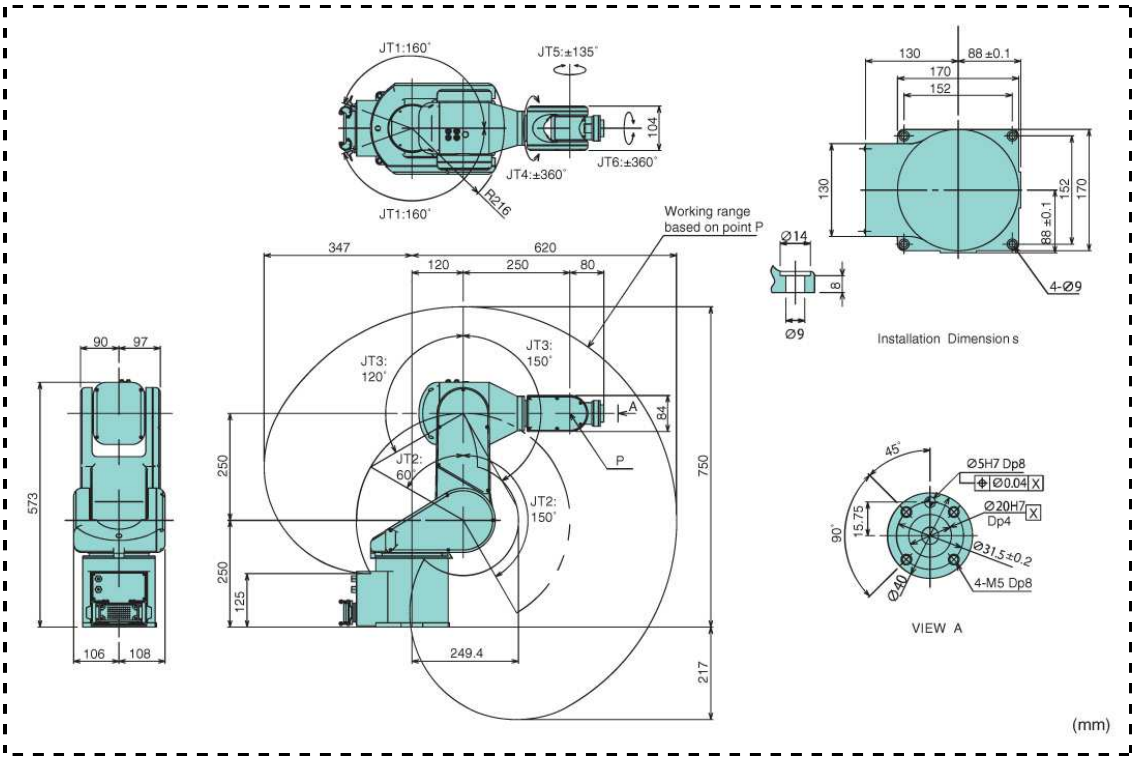

Fig. 2 - Le proiezioni del robot

Dall'analisi delle principali caratteristiche si è giunti alla conclusione che il robot è più che idoneo alla nostra applicazione.

La figura 3 mostra il controller del manipolatore industriale. All'interno del case giace la componentistica che si occupa della comunicazione con il robot ed è proprio da lì che partono tutti i cavi su cui viaggiano tutti segnali dal/al robot.

Fig. 3 - Il controller

La figura 4 illustra il **Teach Pendant** ovvero l'organo principale di comunicazione con il robot, infatti esso rappresenta l'interfaccia uomo-macchina del sistema. Tramite questo dispositivo è possibile muovere direttamente il manipolatore sfruttando la programmazione "teach" e memorizzare i vari punti in modo da poterli utilizzare nel programma. Una delle più comode utilità del teach pendant è la scelta dei movimenti secondo vari criteri:

1. Movimento del robot per giunti
2. Movimento del robot lungo gli assi XYZ.

 Fig. 4 - Il teach pendant

2.1. Il software KRterm

Kawasaki mette a disposizione un software per comunicare con il robot chiamato KRterm. Questo software permette di scrivere programmi in un linguaggio proprietario chiamato AS. Per poter utilizzare il software KRterm è necessaria una licenza che, al momento, è presente su un portatile del laboratorio ARL. Dopo aver aperto l'eseguibile, il programma mostra la schermata principale mostrata nella figura sottostante:

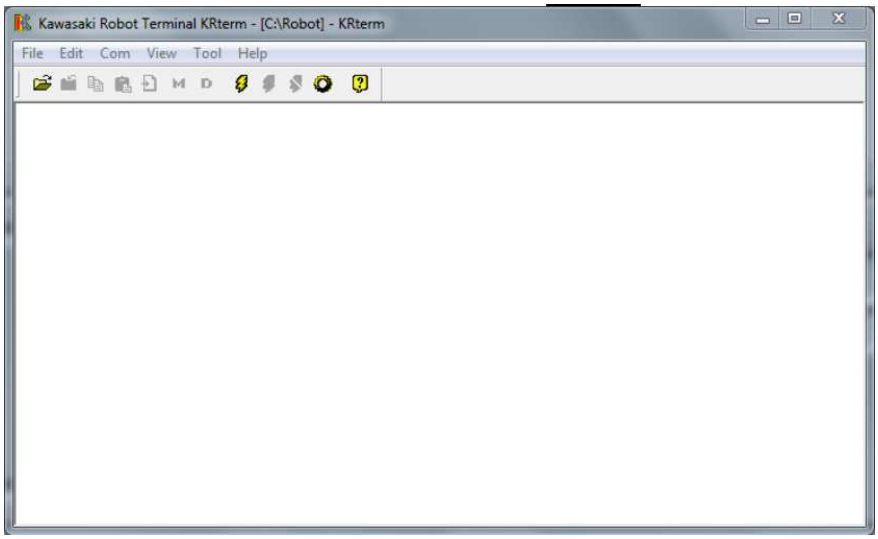

Fig. 5 - Schermata principale di KRterm

A questo punto è necessario effettuare la comunicazione col robot utilizzando il seguente indirizzo ip:

192.168.0.2:23, oppure selezionare la voce **Com** nella schermata principale e selezionare 192.168.0.2:23kawasaki.

A questo punto è necessario effettuare un login digitando **as** e premendo **invio**.

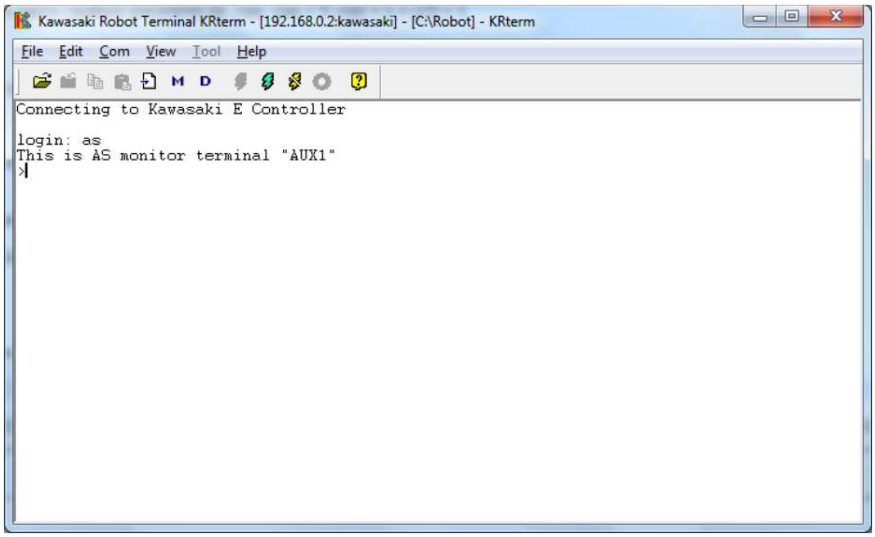

Fig. 6 - Login effettuato su KRterm

Da questo punto in poi è possibile iniziare la scrittura del programma.

3. La soluzione adottata

Il piano di lavoro è composto da una matrice di punti disponibili, costituiti da una serie di tappi di bottiglia rivolti al contrario in grado di contenere le sferette di legno disponibili in laboratorio, utilizzate per il lavoro di pick & place.

La disposizione dei punti disponibili è quella mostrata in figura 7:

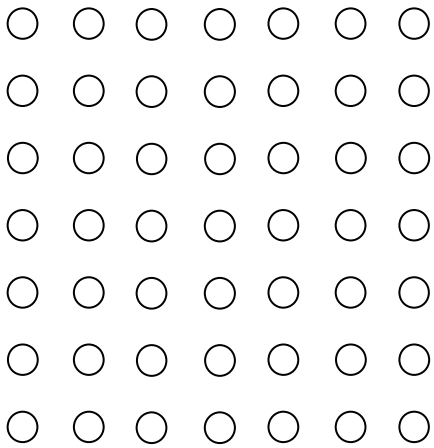

Fig. 7 - Area di lavoro del robot

L'area di lavoro è quindi una matrice di punti 7x7. Le palline di legno a disposizione sono 10, pertanto abbiamo deciso di dedicare le due file di punti più vicine al robot per allocare le sfere come posizione di partenza di tutti i programmi che verranno svolti. Le posizioni di partenza scelte per le palline sono mostrate in figura 8:

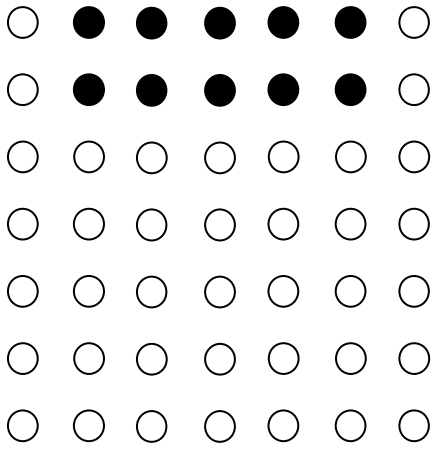

Fig. 8 - Posizioni di partenza delle sferette

Questa è la posizione iniziale in cui sistemare le sferette quando si desidera eseguire i programmi contenuti in questa relazione, pertanto è compito dell'utilizzatore assicurarsi la corretta posizione di partenza.

Un'operazione fondamentale è quella del salvataggio delle posizioni d'interesse del robot: infatti vi è la possibilità di posizionare il braccio del robot in una certa posizione, con l'utilizzo del teach pendant (in

modalità teach), e salvarla con un nome a piacere; I punti che abbiamo salvato per la nostra area di lavoro sono mostrati nella figura 9, con l'indicazione del nome di ognuno:

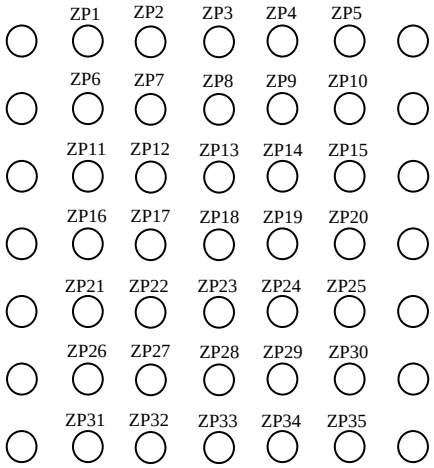

Fig. 9 - Nomi dei punti utilizzati sul piano di lavoro

3.1. Il salvataggio dei punti

Una volta definito il piano di lavoro è stato necessario salvare i punti di interesse in memoria. Il processo di salvataggio è una procedura che non fa altro che memorizzare le coordinate dei singoli punti in formato **joint.** Questo significa che ad ogni giunto viene associata una coordinata relativa alla posizione che questi deve assumere al fine di portare l'end effector (nel nostro caso la pinza per afferrare le sferette) nella posizione desiderata.

La procedura adottata per effettuare il salvataggio dei punti è la seguente:

1. Portarsi in modalità `teach` tramite la levetta sul controller del robot
2. Portare l'end effector nella posizione desiderata, utilizzando il teach pendant, in modo che la pinza afferri senza problemi la sferetta in questione
3. Digitare il comando `HERE #nomeposa`

A questo punto ad ogni posizione salvata in memoria saranno associate le 6 coordinate XYZOAT.

Le prime 3 coordinate XYZ costituiscono la terna cartesiana dello spazio eucludeo mentre OAT sono rispettivamente le coordinate che descrivono la rotazione attorno all'asse XYZ. In questo modo è possibile definire la posizione e l'orientamento dell'end effector.

Quest'operazione evita di doversi scrivere a mano (e dover ricopiare ogni volta) i valori delle singole coordinate e, oltre a migliorare la velocità di scrittura del codice, ne migliora visibilmente la leggibilità. I tappi sul piano di lavoro del robot sono equidistanti, quindi si sarebbe potuto procedere nell'individuare un certo punto e ricavare tutti gli altri variando i valori dei singoli assi in base alla distanza tra un punto e l'altro, ma abbiamo preferito posizionare il braccio sopra ogni singolo punto e salvarne ognuno singolarmente, in quanto i tappi sono sì equidistanti, ma fra ognuno vi è sempre un piccolo errore di qualche millimetro e prelevare ogni singolo punto migliora decisamente la precisione; inoltre prendendo ogni punto singolarmente permette di "mescolare" interpolazione per assi e per giunti, in modo da avere una totale libertà sul posizionamento del braccio del robot, e abbiamo verificato che alcune volte, se i punti sono stati ricavati per via matematica, il robot quando si avvicina ad una certa posizione, soprattutto quelle più lontane rispetto al punto in cui è fissato al tavolo, tende ad uscire dalla propria area di lavoro, restituendo un messaggio d'errore e ovviamente interrompendo l'esecuzione. Quindi la scelta che abbiamo adottato è stata quella di salvare ogni punto singolarmente, operazione che in termini di tempo è risultata veramente minima ed ha migliorato sensibilmente la precisione del nostro lavoro. Si è

pensato di utilizzare le iniziali dei cognomi dei candidati (ZP) per nominare i vari punti della matrice del piano di lavoro in modo tale da evitare nomi comuni e quindi evitare la sovrascrittura delle pose già presenti in memoria. La procedura adottata per effettuare il salvataggio dei punti è stata ripetuta per ogni punto utilizzato del piano di lavoro.

3.2 I programmi

Una volta proceduto al salvataggio dei punti, abbiamo iniziato a creare dei programmi per il robot con il software KRterm; ad esempio si riporta un programma per creare un rettangolo nell'area di lavoro, passando dalle posizioni segnate in blu a quelle rosse, come mostrato in figura 10:

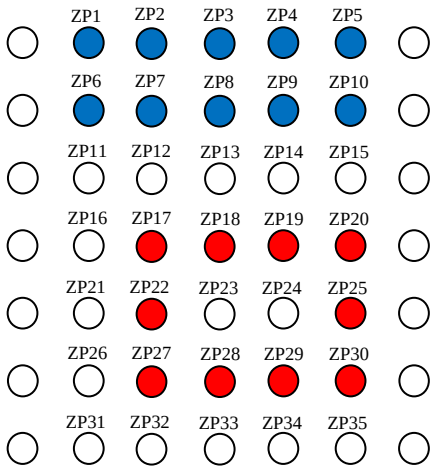

Fig. 10 - Rettangolo sul piano di lavoro

Il programma che effettua lo spostamento delle dieci palline dalla posizione iniziale a quella per formare il rettangolo è il seguente:

```
1-    OPENI
2-    SPEED 20 ALWAYS
3-    HOME
4-    JAPPRO ZP1, 100
5-    LMOVE ZP1
6-    CLOSEI
7-    JAPPRO ZP1, 100
8-    JAPPRO ZP30, 100
9-    LMOVE ZP30
10-   OPENI
11-   JAPPRO ZP30, 100
12-   JAPPRO ZP5, 100
13-   LMOVE ZP5
14-   CLOSEI
15-   JAPPRO ZP5, 100
16-   JAPPRO ZP29, 100
17-   LMOVE ZP29
18-   OPENI
19-   JAPPRO ZP29, 100
44-   JAPPRO ZP4, 100
45-   LMOVE ZP4
46-   CLOSEI
47-   JAPPRO ZP4, 100
48-   JAPPRO ZP17, 100
49-   LMOVE ZP17
50-   OPENI
51-   JAPPRO ZP17, 100
52-   JAPPRO ZP7, 100
53-   LMOVE ZP7
54-   CLOSEI
55-   JAPPRO ZP7, 100
56-   JAPPRO ZP18, 100
57-   LMOVE ZP18
58-   OPENI
59-   JAPPRO ZP18, 100
60-   JAPPRO ZP9, 100
61-   LMOVE ZP9
62-   CLOSEI
```

```
20-   JAPPRO ZP6, 100
21-   LMOVE ZP6
22-   CLOSEI
23-   JAPPRO ZP6, 100
24-   JAPPRO ZP28, 100
25-   LMOVE ZP28
26-   OPENI
27-   JAPPRO ZP28, 100
28-   JAPPRO ZP10, 100
29-   LMOVE ZP10
30-   CLOSEI
31-   JAPPRO ZP10, 100
32-   JAPPRO ZP27, 100
33-   LMOVE ZP27
34-   OPENI
35-   JAPPRO ZP27, 100
36-   JAPPRO ZP2, 100
37-   LMOVE ZP2
38-   CLOSEI
39-   JAPPRO ZP2, 100
40-   JAPPRO ZP22, 100
41-   LMOVE ZP22
42-   OPENI
43-   JAPPRO ZP22, 100
```

```
63-   JAPPRO ZP9, 100
64-   JAPPRO ZP19, 100
65-   LMOVE ZP19
66-   OPENI
67-   JAPPRO ZP19, 100
68-   JAPPRO ZP3, 100
69-   LMOVE ZP3
70-   CLOSEI
71-   JAPPRO ZP3, 100
72-   JAPPRO ZP20, 100
73-   LMOVE ZP20
74-   OPENI
75-   JAPPRO ZP20, 100
76-   JAPPRO ZP8, 100
77-   LMOVE ZP8
78-   CLOSEI
79-   JAPPRO ZP8, 100
80-   JAPPRO ZP25, 100
81-   LMOVE ZP25
82-   OPENI
83-   JAPPRO ZP25, 100
84-   HOME
```

Le palline vengono spostate con questo ordine:

1. ZP1 → ZP30
2. ZP5 → ZP29
3. ZP6 → ZP28
4. ZP10 → ZP27
5. ZP2 → ZP22
6. ZP4 → ZP17
7. ZP7 → ZP18
8. ZP9 → ZP19
9. ZP3 → ZP20
10. ZP8 → ZP25

Valutando il programma risulta evidente a prima vista come il codice sia molto ripetitivo: le istruzioni utilizzate sono sempre JAPPRO, LMOVE, OPENI e CLOSEI[1] nello stesso ordine, essendo le operazioni effettuate dal robot ripetute tra di loro (con la differenza ovviamente delle posizioni per ogni singolo movimento). Nasce l'esigenza quindi di rendere il codice più snello, leggibile e chiaro a chiunque lo voglia "decifrare"; essendo le istruzioni ripetitive tra di loro, la strada migliore da seguire è quella di creare una **subroutine** che effettui il movimento di prelievo della pallina in una certa posizione e collocarla in un'altra. Il programma che effettua questo movimento è descritto nel paragrafo successivo:

3.3 Il programma SPOSTA

```
PROGRAM SPOSTA (.&P1,.&P2,.DIST)

1- OPENI
2- JAPPRO .P1, .DIST
3- LMOVE .P1
4- CLOSEI
5- JAPPRO .P1, .DIST
6- JAPPRO .P2, .DIST
```

[1] Le istruzioni verranno spiegate nel dettaglio nel paragrafo successivo.

```
7- LMOVE .P2
8- OPENI
9- JAPPRO .P2, .DIST
```

Queste sono le operazioni che venivano ripetute di continuo per la creazione del rettangolo sul piano di lavoro:

- le istruzioni `OPENI` e `CLOSEI` servono rispettivamente ad aprire e chiudere la pinza dell'end effector: a differenza delle istruzioni `OPEN` e `CLOSE`, esse inviano immediatamente alla valvola di controllo del serraggio un segnale di apertura o chiusura della pinza se il robot è fermo, oppure viceversa lo inviano non appena il movimento termina;
- l'istruzione `JAPPRO .P1, .DIST` serve ad avvicinarsi ad un certo punto, indicato dal parametro `.P1` della subroutine, ad una certa altezza, espressa in mm dal parametro `.DIST`; `JAPPRO` utilizza un'interpolazione per giunti per arrivare alla posa desiderata, in contrapposizione al comando `LAPPRO` che prevede un'interpolazione lineare del movimento. Abbiamo valutato entrambe le istruzioni nel nostro lavoro, ma il risultato rimane pressoché invariato per le nostre specifiche;
- l'istruzione `LMOVE .P1` indica un movimento lineare verso il punto `.P1`, essendo quest'istruzione sempre seguente a `JAPPRO .P1, .DIST` in ogni situazione, il movimento si riduce ad una traslazione verso il basso, lungo l'asse z, per arrivare al punto `.P1` con un movimento lineare.

Questa subroutine risulta molto efficiente ed adatta ad ogni situazione: infatti, oltre ai parametri necessari quali il punto di partenza e di arrivo, si è deciso di lasciare libertà all'utente della scelta dell'altezza di "approach" per lo spostamento desiderato, in modo da avere un avvicinamento più o meno marcato alla posizione in base alle proprie necessità. Inoltre si sottolinea, come evidente nel codice, che i parametri delle subroutine, in fase di scrittura del programma, devono essere indicati con un punto davanti al nome della variabile.

A questo punto è possibile rivedere il programma per la creazione del rettangolo con l'utilizzo della subroutine appena creata; il codice diventa il seguente:

```
PROGRAM ZP_RETTANGOLO()

1- SPEED VEL ALWAYS
2- OPENI
3- HOME
4- CALL SPOSTA(&ZP1, &ZP30, DIS)
5- CALL SPOSTA(&ZP5, &ZP29, DIS)
6- CALL SPOSTA(&ZP6, &ZP28, DIS)
7- CALL SPOSTA(&ZP10, &ZP27, DIS)
8- CALL SPOSTA(&ZP2, &ZP22, DIS)
9- CALL SPOSTA(&ZP4, &ZP17, DIS)
10- CALL SPOSTA(&ZP7, &ZP18, DIS)
11- CALL SPOSTA(&ZP9, &ZP19, DIS)
12- CALL SPOSTA(&ZP3, &ZP20, DIS)
13- CALL SPOSTA(&ZP8, &ZP25, DIS)
14- HOME
```

Come previsto il programma risulta notevolmente più leggibile e si vedono con chiarezza quali spostamenti vengono effettuati da un punto all'altro.

L'istruzione `CALL` è un tipo di comando di chiamata ad una subroutine esterna: essa infatti richiama il codice della funzione in questione creata separatamente; l'effetto è quello di sostituire alla stringa `CALL NOMEFUNZIONE(PARAMETRI)` l'intero codice contenuto nel programma della subroutine. In questo modo è possibile ripetere N volte la subroutine in questione senza dover copiarla N volte nel codice del programma che, ovviamente, diventerebbe pesante o addirittura illeggibile.

L'istruzione `HOME` inserita sia all'inizio che alla fine del codice fa in modo che il robot si porti nella propria posizione "HOME", ovvero si estende completamente in verticale; anche noi utilizzeremo questa posizione come punto di partenza e fine di ogni singolo programma.

L'istruzione `SPEED VEL ALWAYS` serve ad impostare la velocità di esecuzione del programma; il parametro `VEL` indica la velocità percentuale rispetto alla velocità massima, pertanto se si indica 20 o 100, si avrà rispettivamente l'esecuzione al 20% o al 100% della velocità massima.

Si sottolinea come le variabili posa, all'interno della subroutine, vadano indicati con & anteposto al nome, in questo modo il programma sa che il parametro passato è per l'appunto una variabile posa. Inoltre si nota come non sono stati inseriti direttamente i valori `VEL` e `DIS`, in quanto, come successivamente sarà più chiaro, creeremo un programma "main" che ha lo scopo di eseguire tutte le istruzioni che abbiamo scritto in sequenza e imposteremo dentro a tale programma i parametri `VEL` e `DIS`, in modo che siano gli stessi per tutti i programmi.

Un'altra figura geometrica che può essere facilmente creata con le nostre palline a disposizione è il quadrato. Nella figura 11 sono mostrate le posizioni iniziali (in colore blu) e quelle finali (in rosso); ovviamente per la costruzione del quadrato sono state utilizzate nove sferette sulle dieci disponibili, pertanto una verrà lasciata nella posizione di partenza (il punto `ZP3`, in colore nero):

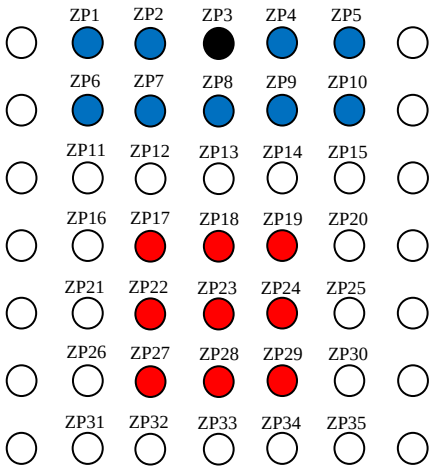

Fig. 11 - Quadrato sul piano di lavoro

Le istruzioni per la creazione di questa figura, sfruttando ovviamente la subroutine `SPOSTA`, sono riportate di seguito:

```
PROGRAM ZP_QUADRATO()


1- SPEED VEL ALWAYS
2- OPENI
3- HOME
4- CALL SPOSTA(&ZP10, &ZP19, DIS)
5- CALL SPOSTA(&ZP9, &ZP18, DIS)
6- CALL SPOSTA(&ZP8, &ZP17, DIS)
7- CALL SPOSTA(&ZP7, &ZP22, DIS)
```

```
8- CALL SPOSTA(&ZP6, &ZP27, DIS)
9- CALL SPOSTA(&ZP5, &ZP28, DIS)
10- CALL SPOSTA(&ZP1, &ZP29, DIS)
11- CALL SPOSTA(&ZP4, &ZP24, DIS)
12- CALL SPOSTA(&ZP2, &ZP23, DIS)
13- HOME
```

Dove gli spostamenti vengono intuitivamente svolti nel seguente ordine:

1. ZP10 ➔ ZP19
2. ZP9 ➔ ZP18
3. ZP8 ➔ ZP17
4. ZP7 ➔ ZP22
5. ZP6 ➔ ZP27
6. ZP5 ➔ ZP28
7. ZP1 ➔ ZP29
8. ZP4 ➔ ZP24
9. ZP2 ➔ ZP23

Le considerazioni sul codice del programma sono analoghe a quelle viste già sulle istruzioni per la creazione del rettangolo.

Un altro programma che abbiamo deciso di creare è quello per la costruzione di un triangolo isoscele sul piano di lavoro, secondo lo schema riportato in figura 12, con posizioni in blu iniziali e quelle in rosso finali; anche in questo caso sono state utilizzate nove palline, pertanto quella in posizione ZP3 non verrà utilizzata:

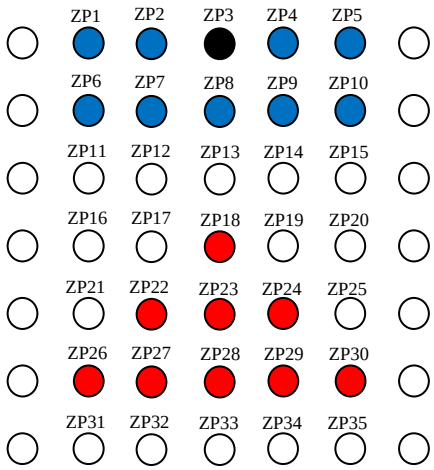

Fig. 12 - Triangolo isoscele sul piano di lavoro

Il codice AS che crea questa figura è riportato di seguito:

```
PROGRAM ZP_TRIANG_ISO()

1- SPEED VEL ALWAYS
2- OPENI
3- HOME
4- CALL SPOSTA(&ZP1, &ZP18, DIS)
5- CALL SPOSTA(&ZP6, &ZP22, DIS)
6- CALL SPOSTA(&ZP2, &ZP24, DIS)
```

```
7- CALL SPOSTA(&ZP7, &ZP26, DIS)
8- CALL SPOSTA(&ZP8, &ZP30, DIS)
9- CALL SPOSTA(&ZP4, &ZP27, DIS)
10- CALL SPOSTA(&ZP9, &ZP29, DIS)
11- CALL SPOSTA(&ZP5, &ZP28, DIS)
12- CALL SPOSTA(&ZP10, &ZP23, DIS)
13- HOME
```

Anche in questo caso si è sfruttata la subroutine che semplifica notevolmente il codice. Gli spostamenti delle palline vengono effettuati secondo l'ordine:

1. ZP1 ➔ ZP18
2. ZP6 ➔ ZP22
3. ZP2 ➔ ZP24
4. ZP7 ➔ ZP26
5. ZP8 ➔ ZP30
6. ZP4 ➔ ZP27
7. ZP9 ➔ ZP29
8. ZP5 ➔ ZP28
9. ZP10 ➔ ZP23

L'ultimo programma che abbiamo deciso di produrre è quello per la realizzazione di un trapezio rettangolo nell'area di lavoro. Nella figura 13 sono indicati i punti di partenza e quelli d'arrivo (rispettivamente blu e rossi, come di consueto):

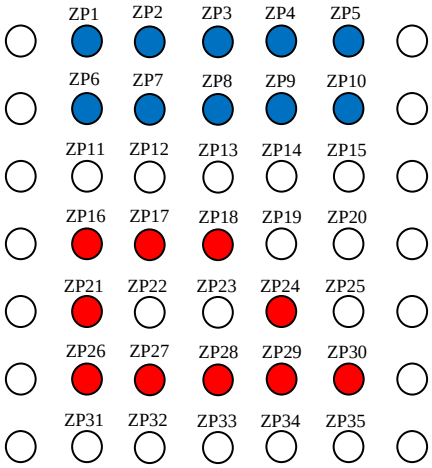

Fig. 13 - Trapezio rettangolo sul piano di lavoro

Il programma per la creazione del trapezio rettangolo è riportato di seguito:

```
PROGRAM ZP_TRAPEZIO_RET()

1- SPEED VEL ALWAYS
2- OPENI
3- HOME
4- CALL SPOSTA(&ZP1, &ZP30, DIS)
5- CALL SPOSTA(&ZP2, &ZP29, DIS)
```

```
6- CALL SPOSTA(&ZP3, &ZP28, DIS)
7- CALL SPOSTA(&ZP4, &ZP27, DIS)
8- CALL SPOSTA(&ZP5, &ZP26, DIS)
9- CALL SPOSTA(&ZP10, &ZP21, DIS)
10- CALL SPOSTA(&ZP9, &ZP16, DIS)
11- CALL SPOSTA(&ZP8, &ZP17, DIS)
12- CALL SPOSTA(&ZP7, &ZP18, DIS)
13- CALL SPOSTA(&ZP6, &ZP24, DIS)
14- HOME
```

Le sfere vengono quindi spostate nel seguente ordine:

1. ZP1 ➔ ZP30
2. ZP2 ➔ ZP29
3. ZP3 ➔ ZP28
4. ZP4 ➔ ZP27
5. ZP5 ➔ ZP26
6. ZP10 ➔ ZP21
7. ZP9 ➔ ZP16
8. ZP8 ➔ ZP17
9. ZP7 ➔ ZP18
10. ZP6 ➔ ZP24

A questo punto sono stati creati i programmi per la realizzazione delle quattro figure che ci eravamo posti di produrre; gli unici tasselli che mancano, per dare continuità ai programmi e per poter permettere di eseguirli uno di seguito all'altro, sono i codici in grado di riportare nelle posizioni iniziali le palline dopo aver creato la figura (permettendo così di eseguire il programma successivo senza dover interrompere l'esecuzione complessiva). Nel prossimo paragrafo verranno illustrati i programmi in grado di riportare indietro le sferette dopo le esecuzioni.

3.4 I programmi per il recupero delle sferette

Tutti i programmi creati sono accumunati dallo stesso punto di partenza di tutte le sferette il legno. Come già accennato in precedenza i programmi vengono svolti in sequenza, questo significa che alla fine di ogni programma tutte le sferette si troveranno in una posizione che, rappresentando la figura per il quale il programma è ideato, non coincide con la posizione di partenza per l'inizio del programma successivo. Sarebbe pertanto possibile riposizionare tutte le sferette in posizione iniziale in maniera manuale, e poi avviare l'esecuzione del programma successivo. La cosa più utile e sensata anche da un punto di vista didattico è quella di fare eseguire questo lavoro direttamente al manipolatore. Per cui, analogamente a quanto fatto per creare le figure, si possono scrivere i programmi che permettano di riportare le palline nelle posizioni iniziali dopo l'esecuzione. Si sarebbero potuti scrivere questi codici direttamente nei programmi che creano le figure (intervallati magari da un `WAIT`), ma la scelta migliore è stata di dividerli in due programmi separati, sia per mantenere una miglior leggibilità e comprensione, ma soprattutto per lasciar una maggior libertà all'utente finale (ad esempio, in questo modo, si può formare una figura e lasciarla intatta sul piano di lavoro).

Ovviamente anche questi programmi, che sono in numero uguale a quello dei "programmi figura", sfruttano il programma `SPOSTA` descritto in precedenza.

I programmi per riportare le sferette nelle loro posizioni iniziali a partire dalle figure composte sono riportati qua di seguito (con l'indicazione dei singoli passaggi effettuati):

1) Programma per il recupero dalle posizioni del rettangolo:

```
PROGRAM ZP_RIT_RETT()

1- SPEED VEL_RIT ALWAYS
2- OPENI
3- CALL SPOSTA(&ZP30, &ZP5, DIS)
```

```
4- CALL SPOSTA(&ZP29, &ZP4, DIS)
5- CALL SPOSTA(&ZP28, &ZP3, DIS)
6- CALL SPOSTA(&ZP27, &ZP2, DIS)
7- CALL SPOSTA(&ZP22, &ZP1, DIS)
8- CALL SPOSTA(&ZP17, &ZP6, DIS)
9- CALL SPOSTA(&ZP18, &ZP7, DIS)
10- CALL SPOSTA(&ZP19, &ZP8, DIS)
11- CALL SPOSTA(&ZP20, &ZP9, DIS)
12- CALL SPOSTA(&ZP25, &ZP10, DIS)
13- HOME
```

1. ZP30 ➔ ZP5
2. ZP29 ➔ ZP4
3. ZP28 ➔ ZP3
4. ZP27 ➔ ZP2
5. ZP22 ➔ ZP1
6. ZP17 ➔ ZP6
7. ZP18 ➔ ZP7
8. ZP19 ➔ ZP8
9. ZP20 ➔ ZP9
10. ZP25 ➔ ZP10

2) Programma per il recupero dalle posizioni del quadrato:

```
PROGRAM ZP_RIT_QUAD()

1- SPEED VEL_RIT ALWAYS
2- OPENI
3- CALL SPOSTA(&ZP19, &ZP1, DIS)
4- CALL SPOSTA(&ZP18, &ZP2, DIS)
5- CALL SPOSTA(&ZP17, &ZP4, DIS)
6- CALL SPOSTA(&ZP22, &ZP5, DIS)
7- CALL SPOSTA(&ZP27, &ZP6, DIS)
8- CALL SPOSTA(&ZP28, &ZP7, DIS)
9- CALL SPOSTA(&ZP29, &ZP8, DIS)
10- CALL SPOSTA(&ZP24, &ZP9, DIS)
11- CALL SPOSTA(&ZP23, &ZP10, DIS)
12- HOME
```

1. ZP19 ➔ ZP1
2. ZP18 ➔ ZP2
3. ZP17 ➔ ZP4
4. ZP22 ➔ ZP5
5. ZP27 ➔ ZP6
6. ZP28 ➔ ZP7
7. ZP29 ➔ ZP8
8. ZP24 ➔ ZP9
9. ZP23 ➔ ZP10

3) Programma per il recupero dalle posizioni del triangolo isoscele:

```
PROGRAM ZP_RIT_TRI()

1- SPEED VEL_RIT ALWAYS
2- OPENI
3- CALL SPOSTA(&ZP18, &ZP1, DIS)
4- CALL SPOSTA(&ZP22, &ZP5, DIS)
5- CALL SPOSTA(&ZP24, &ZP2, DIS)
6- CALL SPOSTA(&ZP26, &ZP4, DIS)
7- CALL SPOSTA(&ZP30, &ZP6, DIS)
8- CALL SPOSTA(&ZP27, &ZP10, DIS)
9- CALL SPOSTA(&ZP29, &ZP7, DIS)
10- CALL SPOSTA(&ZP28, &ZP9, DIS)
11- CALL SPOSTA(&ZP23, &ZP8, DIS)
12- HOME
```

1. ZP18 → ZP1
2. ZP22 → ZP5
3. ZP24 → ZP2
4. ZP26 → ZP4
5. ZP30 → ZP6
6. ZP27 → ZP10
7. ZP29 → ZP7
8. ZP28 → ZP9
9. ZP23 → ZP8

4) Programma per il recupero dalle posizioni del trapezio rettangolo:

```
PROGRAM ZP_RIT_TRAP()

1- SPEED VEL_RIT ALWAYS
2- OPENI
3- CALL SPOSTA(&ZP30, &ZP5, DIS)
4- CALL SPOSTA(&ZP29, &ZP4, DIS)
5- CALL SPOSTA(&ZP28, &ZP3, DIS)
6- CALL SPOSTA(&ZP27, &ZP2, DIS)
7- CALL SPOSTA(&ZP26, &ZP1, DIS)
8- CALL SPOSTA(&ZP21, &ZP10, DIS)
9- CALL SPOSTA(&ZP16, &ZP9, DIS)
10- CALL SPOSTA(&ZP17, &ZP8, DIS)
11- CALL SPOSTA(&ZP18, &ZP7, DIS)
12- CALL SPOSTA(&ZP24, &ZP6, DIS)
13- HOME
```

1. ZP30 → ZP5
2. ZP29 → ZP4
3. ZP28 → ZP3
4. ZP27 → ZP2
5. ZP26 → ZP1
6. ZP21 → ZP10
7. ZP16 → ZP9
8. ZP17 → ZP8
9. ZP18 → ZP7
10. ZP24 → ZP6

Come si può notare in questi programmi, vi è un altro comando per l'impostazione della velocità di esecuzione dei movimenti contenuti. Infatti si è inserito un nuovo parametro, denominato VEL_RIT, che ha lo scopo di diversificare la velocità di esecuzione dei programmi di recupero delle sferette rispetto a quelli di creazione delle figure: esso è stato inserito se si vuole ad esempio velocizzare la "pulizia" dell'area di lavoro rispetto ai programmi di posizionamento e, nel caso si decida di eseguire tutto alla stessa velocità, sarà sufficiente indicare VEL e VEL_RIT con lo stesso valore. Questo parametro verrà impostato direttamente nel programma "main" che richiamerà tutti i sottoprogrammi visti fin qui; come spiegato precedentemente, la velocità viene espressa in percentuale su quella massima, pertanto il valore più grande a cui è possibile impostarlo è 100. Il comando SPEED VEL ALWAYS assegna la velocità dei movimenti di esecuzione del programma da parte del robot, e tale velocità rimane invariata al valore VEL fino a quando esso non viene modificato da un altro comando SPEED. Visto che i programmi ritorna vengono posti direttamente dopo l'esecuzione dei rispettivi programmi di creazione delle figure, è necessario riassegnare il valore della velocità di esecuzione di ogni programma all'inizio dello stesso.

3.5 Il programma MAIN

Per richiamare tutti i programmi visti fin qui, la strada più semplice è stata quella di creare un programma "main" che, tramite il comando CALL, è in grado di eseguire sequenzialmente tutti i programmi desiderati:

```
PROGRAM ZP_MAIN()


1- DIS=100
2- VEL=20
3- VEL_RIT=40
4- SPEED VEL ALWAYS
5- OPENI
6- HOME
7- CALL ZP_QUADRATO
8- CALL ZP_RIT_QUAD
9- CALL ZP_RETTANGOLO
10- CALL ZP_RIT_RETT
11- CALL ZP_TRIANG_ISO
12- CALL ZP_RIT_TRI
13- CALL ZP_TRAPEZIO_RET
14- CALL ZP_RIT_TRAP
15- HOME
```

Oltre a dare una sequenza ai programmi dimostrativi il programma ZP_MAIN inizializza la variabile DIS: questa variabile viene utilizzata per definire una quota sopra la quale si va a portare l'end effector prima e dopo l'operazione di presa della sferetta. Il motivo per il quale si è deciso di utilizzare la variabile DIS è dovuto alla differenza tra il movimento per giunti e il movimento lineare, infatti per arrivare al di sopra di ogni sferetta si ignora la traiettoria con la quale l'end effector raggiunga la posizione, questo perché al di sopra del piano di lavoro non vi è alcun ostacolo che impedisca il libero movimento del manipolatore. Una volta raggiunta la quota DIS al di sopra della sferetta viene impostato forzatamente il movimento del manipolatore secondo una legge lineare, in questo modo il movimento per raggiungere la sferetta è perfettamente verticale: è stato scelto questo metodo per aumentare la precisione nella fase di presa delle singole sferette. Nel codice il valore DIS è stato impostato a 100, ciò significa che prima di raggiungere una pallina, il braccio del robot si porterà ad una quota di 100 mm sopra la posizione desiderata.

Come indicato precedentemente, nel programma main sono stati assegnati anche i valori VEL e VEL_RIT: in questo modo si imposta la velocità di esecuzione al 20%, per la creazione delle figure, e al

40% della velocità massima, per il riposizionamento delle palline nelle posizioni iniziali. Nella sezione successiva si illustrerà anche come è possibile modificare rapidamente questi valori.

4. Modalità operative

La prima operazione da effettuare per il corretto funzionamento è, come detto precedentemente, il posizionamento delle sferette nelle loro posizioni iniziali. La corretta disposizione è mostrata in figura (si tenga presente che i punti `ZP1` - … - `ZP5` sono i più vicini al robot):

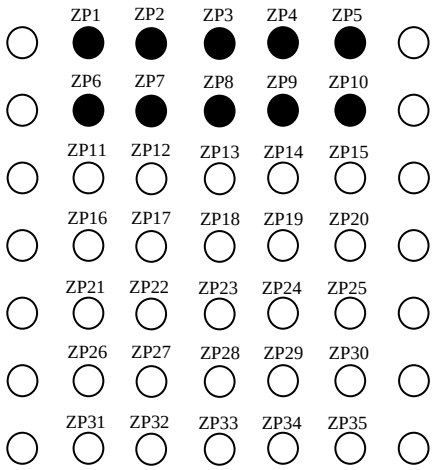

Fig. 14 - Posizioni iniziali delle sfere

Una volta sistemate le sferette nel modo indicato si può passare all'esecuzione vera e propria dei programmi. Per fare questo, dopo aver effettuato tutte le operazioni di accensioni del robot, aver impostato la modalità di esecuzione su REPEAT nel pannello operativo del robot e aver selezionato T. LOCK su OFF sul teach pendant, si passa ad utilizzare il software KRterm, installato sul notebook in dotazione per il robot Kawasaki.

Innanzitutto è necessario collegare il cavo ethernet al pc portatile, poi, una volta avviato il programma KRterm, si effettua la connessione al robot, come già spiegato nel paragrafo 2.1. Per avviare i programmi dal terminale si utilizza il comando `EXECUTE <nome_programma>`, quindi se vogliamo avviare il nostro programma main dovremo digitare `EXECUTE ZP_MAIN` e poi premere INVIO; il robot a questo punto inizierà l'esecuzione di tutti i programmi in sequenza.

Se si vogliono invece modificare le velocità di esecuzione dei programmi, è necessario editare il file `ZP_MAIN` e assegnare dei nuovi valori alle variabili `VEL` e `VEL_RIT`. Per fare questo si utilizza il comando `EDIT <nome programma>,<numero passo>`, che nella nostra specifica situazione verrà scritto come: `EDIT ZP_MAIN,2`; una volta premuto INVIO, il software chiederà di editare la riga 2 del codice (ovvero `VEL=20`), a questo punto è sufficiente scrivere `VEL=<nuova velocità>` e premere INVIO. Il software ora chiederà di editare la riga successiva, ovvero `VEL_RIT=40`, se si è interessati a modificare anche questa variabile è sufficiente scrivere `VEL_RIT=<nuova velocità ritorno>` e poi premere INVIO. Per terminare la modifica del codice bisogna digitare `E` e poi premere INVIO; a questo punto si torna sul terminale e si possono dare nuovamente comandi monitor al robot.

5. Conclusioni e sviluppi futuri

Il lavoro svolto ha permesso di creare un programma dimostrativo adatto, ad esempio, ai ragazzi delle scuole superiori che vengono in visita all'università o a tutti coloro che si avvicinano per la prima volta al laboratorio ARL della facoltà di ingegneria di Brescia; tramite questo programma si può dare una semplice dimostrazione delle funzionalità del robot Kawasaki presente in laboratorio.

Grazie alla creazione della subroutine `SPOSTA` è possibile creare un qualsiasi programma di pick & place: se si vuole continuare sulla strada della formazione di figure geometriche, se ne possono creare ancora molte (ad esempio un parallelogramma o un rombo) oppure con un po' di inventiva ci può divertire a costruire nuovi programmi in stile pick & place: una volta selezionati i punti di interesse (ed eventualmente salvati se non presenti nella memoria del robot), tramite il comando `SPOSTA` risulta immediato spostare un oggetto da un punto ad un altro.

Bibliografia

[1] Petre, D.: "Guida introduttiva all'utilizzo del robot Kawasaki RS03N", Brescia, 2012.

URL: http://www.ing.unibs.it/~arl/docs/projects/Doc_07.pdf

[2] Saccani, S: "Guida all'utilizzo del robot Kawasaki RS03N", Brescia, 2012.

URL: http://www.ing.unibs.it/~arl/docs/projects/Doc_09.pdf

[3] Kawasaki Heavy Industries: "Kawasaki robot controller serie E – Manuale operativo", prima edizione, Marzo 2010.

URL: http://www.ing.unibs.it/~arl/docs/documentation/Kawasaki.html

[4] Kawasaki Heavy Industries: "Kawasaki robot controller serie E – Manuale di riferimento del linguaggio AS", prima edizione, Marzo 2010.

URL: http://www.ing.unibs.it/~arl/docs/documentation/Kawasaki.html

Allegati

Si allegano di seguito gli screenshot, prelevati dal software KRterm, dei codici dei programmi sviluppati in questa relazione. Per visualizzare il codice di un programma in KRterm è sufficiente scrivere `list/p <nome_programma>` e premere INVIO.

```
   .PROGRAM sposta(.&p1,.&p2,.dist)
1          OPENI
2          JAPPRO .p1,.dist
3          LMOVE .p1
4          CLOSEI
5          JAPPRO .p1,.dist
6          JAPPRO .p2,.dist
7          LMOVE .p2
8          OPENI
9          JAPPRO .p2,.dist
   .END
```

```
   .PROGRAM zp_quadrato()
 1          SPEED vel ALWAYS
 2          OPENI
 3          HOME
 4          CALL sposta(&zp10,&zp19,dis)
 5          CALL sposta(&zp9,&zp18,dis)
 6          CALL sposta(&zp8,&zp17,dis)
 7          CALL sposta(&zp7.&zp22.dis)
 8          CALL sposta(&zp6,&zp27,dis)
 9          CALL sposta(&zp5,&zp28,dis)
10          CALL sposta(&zp1,&zp29,dis)
11          CALL sposta(&zp4,&zp24,dis)
12          CALL sposta(&zp2,&zp23,dis)
13          HOME
   .END
```

```
.PROGRAM zp_rit_quad()
 1          SPEED vel_rit ALWAYS
 2          OPENI
 3          CALL sposta(&zp19,&zp1,dis)
 4          CALL sposta(&zp18,&zp2,dis)
 5          CALL sposta(&zp17,&zp4,dis)
 6          CALL sposta(&zp22,&zp5,dis)
 7          CALL sposta(&zp27,&zp6,dis)
 8          CALL sposta(&zp28,&zp7,dis)
 9          CALL sposta(&zp29,&zp8,dis)
10          CALL sposta(&zp24,&zp9,dis)
11          CALL sposta(&zp23,&zp10,dis)
12          HOME
.END
```

```
.PROGRAM zp_rettangolo()
 1          SPEED vel ALWAYS
 2          OPENI
 3          HOME
 4          CALL sposta(&zp1,&zp30,dis)
 5          CALL sposta(&zp5,&zp29,dis)
 6          CALL sposta(&zp6,&zp28,dis)
 7          CALL sposta(&zp10,&zp27,dis)
 8          CALL sposta(&zp2,&zp22,dis)
 9          CALL sposta(&zp4,&zp17,dis)
10          CALL sposta(&zp7,&zp18,dis)
11          CALL sposta(&zp9,&zp19,dis)
12          CALL sposta(&zp3,&zp20,dis)
13          CALL sposta(&zp8,&zp25,dis)
14          HOME
.END
```

```
.PROGRAM zp_rit_rett()
 1          SPEED vel_rit ALWAYS
 2          OPENI
 3          CALL sposta(&zp30,&zp5,dis)
 4          CALL sposta(&zp29,&zp4,dis)
 5          CALL sposta(&zp28,&zp3,dis)
 6          CALL sposta(&zp27,&zp2,dis)
 7          CALL sposta(&zp22,&zp1,dis)
 8          CALL sposta(&zp17,&zp6,dis)
 9          CALL sposta(&zp18,&zp7,dis)
10          CALL sposta(&zp19,&zp8,dis)
11          CALL sposta(&zp20,&zp9,dis)
12          CALL sposta(&zp25,&zp10,dis)
13          HOME
.END
```

```
.PROGRAM zp_triang_iso()
 1          SPEED vel ALWAYS
 2          OPENI
 3          HOME
 4          CALL sposta(&zp1,&zp18,dis)
 5          CALL sposta(&zp6,&zp22,dis)
 6          CALL sposta(&zp2,&zp24,dis)
 7          CALL sposta(&zp7,&zp26,dis)
 8          CALL sposta(&zp8,&zp30,dis)
 9          CALL sposta(&zp4,&zp27,dis)
10          CALL sposta(&zp9,&zp29,dis)
11          CALL sposta(&zp5,&zp28,dis)
12          CALL sposta(&zp10,&zp23,dis)
13          HOME
.END
```

```
.PROGRAM zp_rit_tri()
 1          SPEED vel_rit ALWAYS
 2          OPENI
 3          CALL sposta(&zp18,&zp1,dis)
 4          CALL sposta(&zp22,&zp5,dis)
 5          CALL sposta(&zp24,&zp2,dis)
 6          CALL sposta(&zp26,&zp4,dis)
 7          CALL sposta(&zp30,&zp6,dis)
 8          CALL sposta(&zp27,&zp10,dis)
 9          CALL sposta(&zp29,&zp7,dis)
10          CALL sposta(&zp28,&zp9,dis)
11          CALL sposta(&zp23,&zp8,dis)
12          HOME
.END
```

```
.PROGRAM zp_trapezio_ret()
 1          SPEED vel ALWAYS
 2          OPENI
 3          HOME
 4          CALL sposta(&zp1,&zp30,dis)
 5          CALL sposta(&zp2,&zp29,dis)
 6          CALL sposta(&zp3,&zp28,dis)
 7          CALL sposta(&zp4,&zp27,dis)
 8          CALL sposta(&zp5,&zp26,dis)
 9          CALL sposta(&zp10,&zp21,dis)
10          CALL sposta(&zp9,&zp16,dis)
11          CALL sposta(&zp8,&zp17,dis)
12          CALL sposta(&zp7,&zp18,dis)
13          CALL sposta(&zp6,&zp24,dis)
14          HOME
.END
```

```
.PROGRAM zp_rit_trap()
 1          SPEED vel_rit ALWAYS
 2          OPENI
 3          CALL sposta(&zp30,&zp5,dis)
 4          CALL sposta(&zp29,&zp4,dis)
 5          CALL sposta(&zp28,&zp3,dis)
 6          CALL sposta(&zp27,&zp2,dis)
 7          CALL sposta(&zp26,&zp1,dis)
 8          CALL sposta(&zp21,&zp10,dis)
 9          CALL sposta(&zp16,&zp9,dis)
10          CALL sposta(&zp17,&zp8,dis)
11          CALL sposta(&zp18,&zp7,dis)
12          CALL sposta(&zp24,&zp6,dis)
13          HOME
.END
.PROGRAM zp_main()
 1          dis = 100
 2          vel = 20
 3          vel_rit = 40
 4          SPEED vel ALWAYS
 5          OPENI
 6          HOME
 7          CALL zp_quadrato
 8          CALL zp_rit_quad
 9          CALL zp_rettangolo
10          CALL zp_rit_rett
11          CALL zp_triang_iso
12          CALL zp_rit_tri
13          CALL zp_trapezio_ret
14          CALL zp_rit_trap
15          HOME
.END
```

Indice