Esercitazione di Fondamenti di Robotica

"*Controllo punto-punto dell'uniciclo (2)*"

Autunno 2018

Si consideri un sistema robotico costituito da un veicolo di tipo uniciclo e da un braccio planare ad un giunto, impiegati nella catena logistica e di montaggio di un'azienda automatizzata. Il veicolo deve muoversi lungo il percorso specificato dai waypoint P_i dell'esercitazione precedente, ma questa volta è vincolato a muoversi lungo corridoi poco più grandi della sua dimensione e i cui vertici sono specificati proprio dai waypoint stessi. Il braccio robotico è posizionato in P_3, è in grado di muoversi sul piano orizzontale e deve spostare continuamente i prodotti depositati dal veicolo di 180 gradi (cf. la figura).

Si richiede di progettare un pianificatore e degli opportuni controllori per il veicolo e un controllore per il braccio robotico. Si realizzi una simulazione in Matlab/Simulink dello scenario sopra descritto.

A tal fine, si svolgano i seguenti passi:

- Fissato il generico waypoint P_i, determinare, mediante il metodo diretto di Lyapunov, una prima legge di retroazione che, applicata attraverso i due ingressi, v e ω, consenta al veicolo di orientarsi nella direzione del waypoint rimanendo sul posto;
- Ancora mediante il metodo diretto di Lyapunov, determinare una seconda legge di retroazione che consenta al veicolo di muoversi verso il waypoint, senza urtare contro le pareti del corridoio che collega P_{i-1} a P_i;
- Formalizzare il funzionamento di un pianificatore che ha il compito di far in modo che il veicolo esegua prima la *manovra* di orientamento e poi quella di traslazione verso il waypoint P_i;
- Realizzare uno schema in Simulink che permetta di verificare il corretto funzionamento del pianificatore di manovra con i due controllori, entrambi ottenuti attraverso blocchi distinti;
- Modificare il pianificatore precedente in modo che sia anche in grado di aggiornare (istantaneamente) il waypoint desiderato, non appena il veicolo ha completato la sequenza delle due manovre; verificare il funzionamento in Matlab/Simulink;
- Modellizzare il braccio robotico e progettarne un controllore in grado di agire sulla coppia di ingresso, al fine di ottenere il movimento desiderato;

(continua)

Nota: si richiede di commentare in modo chiaro ogni passaggio importante.

Soluzione

▶ Con riferimento al sistema di coordinate polari suggerite nella precedente esercitazione, il problema dell'orientamento del veicolo può essere tradotto in quello di rendere asintoticamente stabile la variabile β. A tal fine, scegliendo la funzione candidata di Lyapunòv

$$V = \frac{1}{2}\beta^2 \,,$$

la cui derivata direzionale è

$$\dot{V} = \beta c_\beta \, v^* - \beta\omega \,,$$

si evince che una scelta degli ingressi v^* e ω tali da ottenere la funzione semi-definita negativa $\dot{V} = -k_\beta \beta^2$ è

$$\begin{pmatrix} v \\ \omega \end{pmatrix} = \begin{pmatrix} \rho\, v^* \\ \omega \end{pmatrix} = \begin{pmatrix} 0 \\ k_\beta \beta \end{pmatrix} .$$

▶ Per ottenere lo scopo è sufficiente scegliere

$$V = \frac{1}{2}\left(\rho^2 + \beta^2\right) ,$$

la cui derivata direzionale è

$$\dot{V} = -\left(\rho^2 \, c_\beta + \beta \, s_\beta\right) v^* - \beta\omega \,,$$

che diventa

$$\dot{V} = -k_v \left(\rho^2 \, c_\beta + \beta \, s_\beta\right)^2 - k_\beta \beta^2 \,,$$

scegliendo

$$\begin{pmatrix} v \\ \omega \end{pmatrix} = \begin{pmatrix} \rho\, v^* \\ \omega \end{pmatrix} = \begin{pmatrix} k_v \, \rho \left(\rho^2 \, c_\beta + \beta \, s_\beta\right) \\ k_\beta \beta \end{pmatrix} .$$

▶ Omesso

▶ Omesso

▶ Lo schema Simulink prevede quanto segue:

```
function Input  = uniorientctrl(State, Ref)
%#codegen

x = State(1);
y = State(2);
psi = State(3);

xd = Ref(1);
yd = Ref(2);

k_beta = 1;

beta = atan2((y-yd),(x-xd)) - psi + pi;

v = 0;
omega = k_beta*beta;

Input = [v; omega];

end
```

```
function Input  = unip2pctrl(State, Ref)
%#codegen

x = State(1);
y = State(2);
psi = State(3);

xd = Ref(1);
yd = Ref(2);

k_v = 0.05;

rho = sqrt((x-xd)^2+(y-yd)^2);
%phi = atan2((y-yd),(x-xd));
beta = atan2((y-yd),(x-xd)) - psi + pi;

v = rho*k_v*(rho^2*cos(beta) + beta*sin(beta));
omega = beta;

Input = [v; omega];

end
```

```
function Input  = selector(u1, u2, maneuver)
%#codegen

if maneuver == 1
    Input = u1;
elseif maneuver == 2
    Input = u2;
else
    Input = zeros(2,1);
end

end
```

```
function [Ref, PlannerState] = uniplanner(PlannerState, State, RefMatrix)
%#codegen

WP_MAX = 6; % Number of waypoints
TOLERANCE = 1;

wpIdx = PlannerState(1);
maneuver = PlannerState(2);

x = State(1);
y = State(2);
psi = State(3);

xd = RefMatrix(wpIdx, 1);
yd = RefMatrix(wpIdx, 2);

beta = atan2((y-yd),(x-xd)) - psi + pi;
```

```
Ref = RefMatrix(wpIdx,:)';

if maneuver == 2 % currently moving towards next waypoint
    error = State(1:2)-Ref;
    if abs(error(1)) + abs(error(2)) < TOLERANCE && wpIdx < WP_MAX
        wpIdx = wpIdx + 1;
        maneuver = 1; % Reorient before moving to next waypoint
    end
elseif maneuver == 1
    if abs(beta) < 0.1
        maneuver = 2; % Now ready to move to next waypoint
    end
end

PlannerState(1) = wpIdx;
PlannerState(2) = maneuver;

end

function holder(block)
      setup(block);
%endfunction

function setup(block)
       % Register number of ports
       block.NumInputPorts  = 1;
       block.NumOutputPorts = 1;

       % Setup port properties to be inherited or dynamic
       block.SetPreCompInpPortInfoToDynamic;
       block.SetPreCompOutPortInfoToDynamic;

       % Override input port properties
       block.InputPort(1).Dimensions  = 2;
       block.InputPort(1).DatatypeID  = 0;  % double
       block.InputPort(1).Complexity  = 'Real';
       block.InputPort(1).DirectFeedthrough = false;

       % Override output port properties
       block.OutputPort(1).Dimensions  = 2;
       block.OutputPort(1).DatatypeID  = 0; % double
       block.OutputPort(1).Complexity  = 'Real';

       % Register parameters
       block.NumDialogPrms     = 0;

       block.SampleTimes = [-1 0];

       block.SimStateCompliance = 'DefaultSimState';

       block.RegBlockMethod('PostPropagationSetup',    @DoPostPropSetup);
```

```
        block.RegBlockMethod('InitializeConditions', @InitializeConditions);
        block.RegBlockMethod('Start', @Start);
        block.RegBlockMethod('Outputs', @Outputs);     % Required
        block.RegBlockMethod('Update', @Update);
        block.RegBlockMethod('Derivatives', @Derivatives);
        block.RegBlockMethod('Terminate', @Terminate); % Required
%end setup

function DoPostPropSetup(block)
        block.NumDworks = 1;

        block.Dwork(1).Name            = 'x1';
        block.Dwork(1).Dimensions      = 2;
        block.Dwork(1).DatatypeID      = 0;      % double
        block.Dwork(1).Complexity      = 'Real'; % real
        block.Dwork(1).UsedAsDiscState = true;

function InitializeConditions(block)
%end InitializeConditions

function Start(block)
        block.Dwork(1).Data = ones(2,1);
%end Start

function Outputs(block)
        block.OutputPort(1).Data = block.Dwork(1).Data;% + block.InputPort(1).Data;
%end Outputs

function Update(block)
        block.Dwork(1).Data = block.InputPort(1).Data;
%end Update

function Derivatives(block)
%end Derivatives

function Terminate(block)
%end Terminate
```

Lo schema Simulink è riportato in Fig. 2.

► (*continua*)

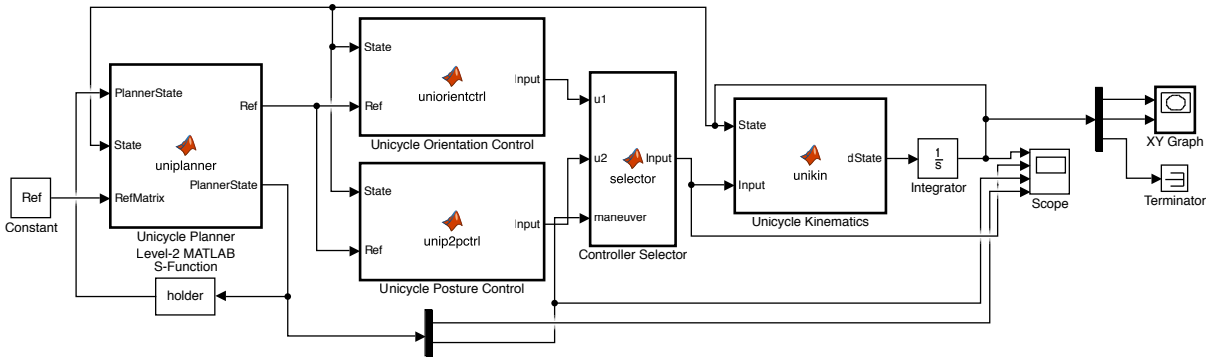

Figura 1: *Schema Simulink che consente di simulare il comportamento del veicolo controllato e supervisionato dal pianificatore.*

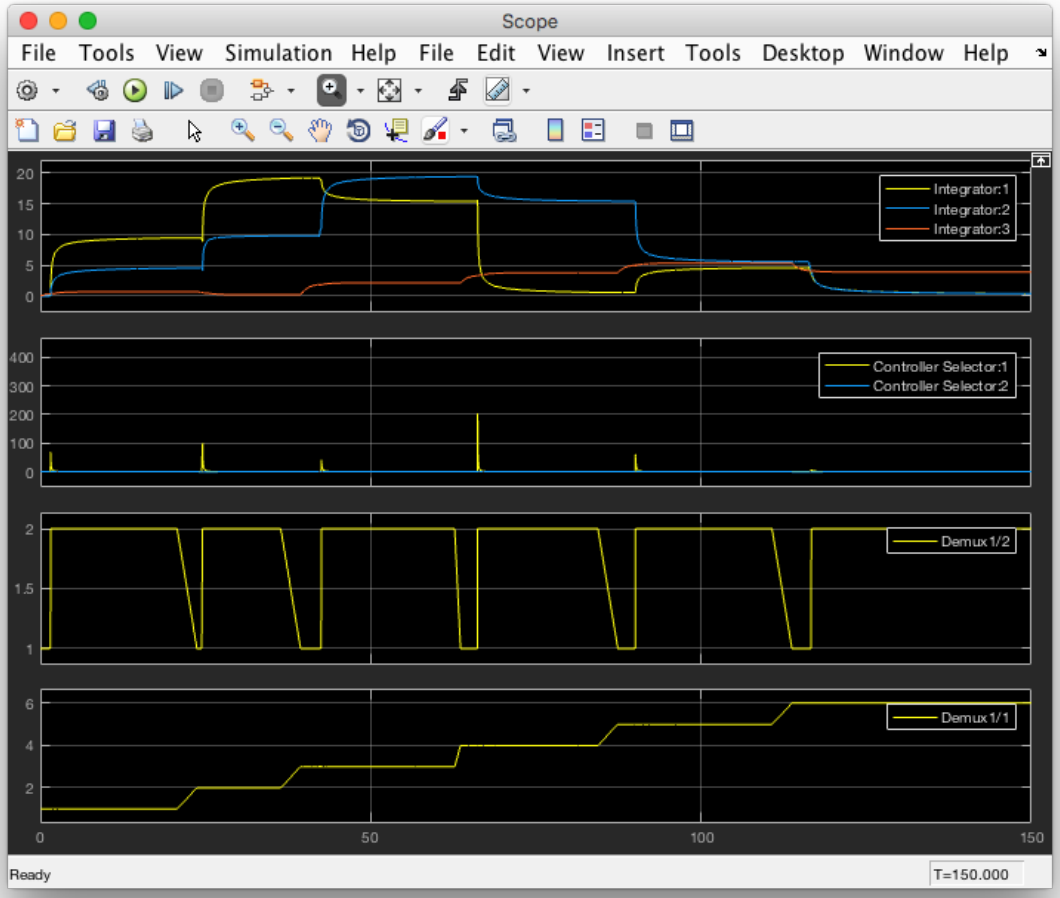

Figura 2: *Simulazione.*