Esercitazione di Fondamenti di Robotica

"*Controllo punto-punto dell'uniciclo*"

Autunno 2018

Si consideri un veicolo di tipo uniciclo che deve svolgere autonomamente un compito, che consiste nel raggiungere presso alcuni cosiddetti "punti di via" (waypoint) muovendosi in un ambiente privo di ostacoli. Supponendo che il veicolo si trovi inizialmente nella posizione $P_0 = (0, 0)$ e che i waypoint siano specificati dalle coordinate

$$P_0 = (0,0), P_1 = (10,5), P_2 = (20,10), P_3 = (15,20), P_4 = (0,15), P_5 = (5,5), P_6 = (0,0)\,.$$

e dato il modello cinematico del veicolo stesso

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\psi} \end{pmatrix} = \begin{pmatrix} c_\psi \\ s_\psi \\ 0 \end{pmatrix} v + \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \omega\,,$$

si progetti un controllore che, agendo sugli ingressi disponibili v e ω, consente di ottenere lo scopo. Si realizzi una simulazione in Matlab/Simulink dello scenario sopra descritto.

A tal fine, si svolgano i seguenti passi:

- Ricavare la cinematica del veicolo in coordinate polari rispetto alla posizione del generico waypoint $P_i = (X_i, Y_i)$ (*Suggerimento: si consideri come una delle variabili di stato l'angolo β tra la direzione di avanzamento del veicolo e la direzione istantanea del waypoint:*

$$\begin{pmatrix} \rho \\ \varphi \\ \beta \end{pmatrix} = \begin{pmatrix} \sqrt{(x-X_i)^2 + (y-Y_i)^2} \\ \arctan\left(\dfrac{y-Y_i}{x-X_i}\right) \\ \arctan\left(\dfrac{y-Y_i}{x-X_i}\right) + \pi - \psi \end{pmatrix},$$

  si scriva la relazione inversa che esprime le variabili originali (x, y, ψ) in funzione delle coordinate polari, ed infine si consideri come quasi velocità lineare di avanzamento la quantità $v^* = v/\rho$);

- Fissato il generico waypoint P_i, determinare, mediante il metodo diretto di Lyapunov, una legge di retroazione che, applicata attraverso i due ingressi, v e ω, rende asintoticamente stabile l'origine del sistema scritto nelle coordinate polari;

- Realizzare uno schema in Simulink che permetta di verificare il corretto funzionamento del controllore;

- Formalizzare il funzionamento di un pianificatore che ha il compito di aggiornare (istantaneamente) il waypoint desiderato, non appena il veicolo ha raggiunto quello precedente, specificando tale condizione in funzione delle variabili di stato;

- Realizzare il pianificatore in Simulink (*Suggerimento: laddove necessario, si può fare uso di variabili locali alla funzione Matlab che realizzare il pianificatore, ma con tempo di vita globale; ciò si ottiene dichiarando e definendo nella funzione la generica variabile* var *secondo lo schema:*

```
persistent var
if isempty(var)
    var = <initial_value>;
end
```

- Realizzare e simulare il sistema completo.

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

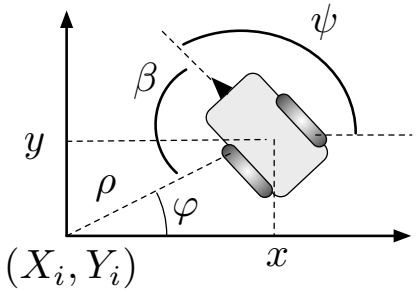

Figura 1: *Variabili polari dello stato del veicolo riferite rispetto al generico waypoint* $P_i = (X_i, Y_i)$

Soluzione

▶ Il compito di raggiungere il generico waypoint $P_i = (X_i, Y_i)$ può essere tradotto nel problema di stabilizzare l'origine del modello del veicolo, esprimendo il suo stato nelle variabili polari suggerite nel testo (cf. Fig 1). A tal scopo, la relazione inversa che permette di esprimere le variabili originali in funzione di quelle polari è la seguente:

$$\begin{pmatrix} x \\ y \\ \psi \end{pmatrix} = \begin{pmatrix} X_i + \rho\, c_\varphi \\ Y_i + \rho\, s_\varphi \\ \beta - \varphi - \pi \end{pmatrix} ,$$

che può essere facilmente ricavata geometricamente. Inoltre, derivando rispetto al tempo le variabili polari si ottiene il seguente modello cinematico:

$$\begin{pmatrix} \dot\rho \\ \dot\varphi \\ \dot\beta \end{pmatrix} = \begin{pmatrix} -c_\beta \\ \dfrac{s_\beta}{\rho} \\ \dfrac{s_\beta}{\rho} \end{pmatrix} v + \begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix} \omega ,$$

il quale possiede una singolarità per $\rho = 0$, che può essere evitata ridefinendo l'ingresso $v = \rho\, v^*$, con cui ottiene:

$$\begin{pmatrix} \dot\rho \\ \dot\varphi \\ \dot\beta \end{pmatrix} = \begin{pmatrix} -\rho\, c_\beta \\ s_\beta \\ s_\beta \end{pmatrix} v^* + \begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix} \omega .$$

▶ Una possibile scelta come funzione candidata di Lyapunov è la seguente:

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

la cui derivata direzionale è

$$\begin{aligned} \dot V &= -\rho^2\, c_\beta\, v^* + \varphi\, s_\beta\, v^* - \beta\,(\omega - s_\beta\, v^*) = \\ &= -\rho^2\, c_\beta\, v^* - \beta\,(\omega - (s_\beta + \varphi\, \mathrm{sinc}(\beta))\, v^*) . \end{aligned}$$

Una possibile scelta per la $\dot V$ è rappresentata dalla funzione semi-definita negativa

$$\dot V = -k_v \rho^2 c_\beta^2 - k_\beta\, \beta^2 ,$$

che si ottiene scegliendo le seguenti espressioni per i feedback da applicare attraverso gli ingressi del sistema:

$$\begin{pmatrix} v \\ \omega \end{pmatrix} = \begin{pmatrix} \rho\, v^* \\ \omega \end{pmatrix} = \begin{pmatrix} \rho\, k_v\, c_\beta \\ k_\beta\, \beta + k_v\, (s_\beta + \varphi\, \mathrm{sinc}(\beta))\, c_\beta \end{pmatrix} .$$

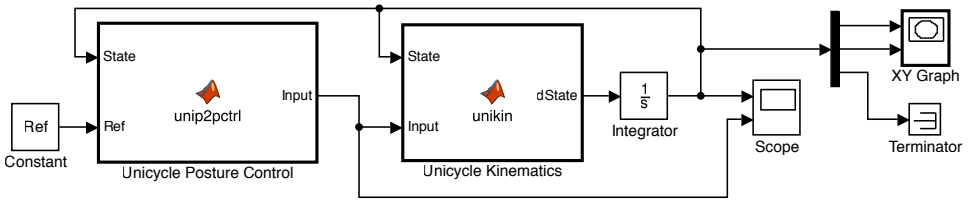

Figura 2: *Schema Simulink che consente di simulare il comportamento del veicolo durante l'inseguimento del generico waypoint P_i.*

▶ Uno schema Simulink che permette di simulare il comportamento del veicolo con controllore di posizione può essere costruito usando due Matlab function. La prima di queste contiene il codice della cinematica del veicolo:

```
function dState = unikin(State, Input)
%#codegen
x = State(1);
y = State(2);
psi = State(3);

v = Input(1);
omega = Input(2);

dState = [v*cos(psi); v*sin(psi); omega];
end
```

La seconda funzione contiene invece il codice del controllore:

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

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

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

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

v = rho*cos(beta);
omega = beta + (phi*sinc(beta)+sin(beta)) * cos(beta);

Input = [v; omega];
end
```

Lo schema Simulink è riportato in Fig 2.

▶ Il sottosistema pianificatore consiste di una macchina a stati finiti, responsabile di aggiornare la variabile discreta waypoint attuale, *wpIdx*. Tale sottosistema consiste di una fase di inizializzazione,

$$\text{wpIdx} = 1$$

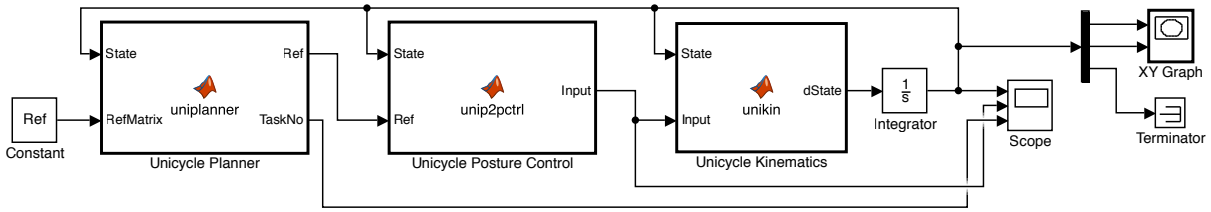

Figura 3: *Schema Simulink che consente di simulare il comportamento del veicolo controllato e supervisionato dal pianificatore.*

e di una fase di aggiornamento

$$\text{wpIdx} = \begin{cases} \text{wpIdx} + 1 & \text{if wpIdx} < \text{WP-MAX}\,, \\ \text{wpIdx} & \text{otherwise}\,, \end{cases}$$

che viene attivata ogni qualvolta si raggiunge la condizione

$$\rho < \text{TOLERANCE}\,,$$

in cui TOLERANCE è una tolleranza.

▶ Il pianificatore è descritto da un'ulteriore Matlab function:

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

persistent wpIdx WP_MAX % Waypoint index, Number of waypoints
if isempty(wpIdx)
    wpIdx = 1;
    WP_MAX = 6;
end

TOLERANCE = 0.01;

Ref = RefMatrix(wpIdx,:)';

error = State(1:2)-Ref;

if abs(error(1)) + abs(error(2)) < TOLERANCE && wpIdx < WP_MAX
    wpIdx = wpIdx + 1;
    disp(RefMatrix(wpIdx,:))
end

TaskNo = wpIdx;
end
```

▶ Avendo definito lo script di inizializzazione

```
clearvars, close all, clc
Ref = [10 5; 20 10; 15 20; 0 15; 5 5; 0 0]
State0 = zeros(3,1)
```

lo schema Simulink è riportato in Fig. 3.

Addendum per Matlab/Simulink 2017b

▶ Per evitare l'uso di variabili persistenti si può ricorrere alla seguente soluzione. Si realizza un blocchetto in grado di memorizzare un numero e di incrementarlo nel momento in cui riceve un segnale positivo in ingresso. Tale blocchetto può essere realizzato, evitando di dover discretizzare il modello dinamico, con una Matlab S-Function il cui codice è inserito nel file *incrementer.m* ed è il seguente:

```
function incrementer(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  = 1;
      block.InputPort(1).DatatypeID  = 0;  % double
      block.InputPort(1).Complexity  = 'Real';
      block.InputPort(1).DirectFeedthrough = false;

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

      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            = 'wpIdx';
      block.Dwork(1).Dimensions      = 1;
      block.Dwork(1).DatatypeID      = 0; % double
      block.Dwork(1).Complexity      = 'Real'; % real
```

```
      block.Dwork(1).UsedAsDiscState = true;
%end DoPostPropSetup

function InitializeConditions(block)
%end InitializeConditions

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

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

function Update(block)
      if block.InputPort(1).Data > 0
            block.Dwork(1).Data = block.Dwork(1).Data + 1;
      end
%end Update

function Derivatives(block)
%end Derivatives

function Terminate(block)
%end Terminate
```

Coerentemente è necessario modificare il codice del pianificatore come segue:

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

WP_MAX = 6; % Number of waypoints
TOLERANCE = 0.01;

Ref = RefMatrix(wpIdx,:)';
incrementSignal = 0;

error = State(1:2)-Ref;
if abs(error(1)) + abs(error(2)) < TOLERANCE && wpIdx < WP_MAX
    incrementSignal = 1;
end

end
```

Lo schema Simulink diventa infine quello riportato in Fig. 4. La simulazione è invece riportata in Fig. 5.

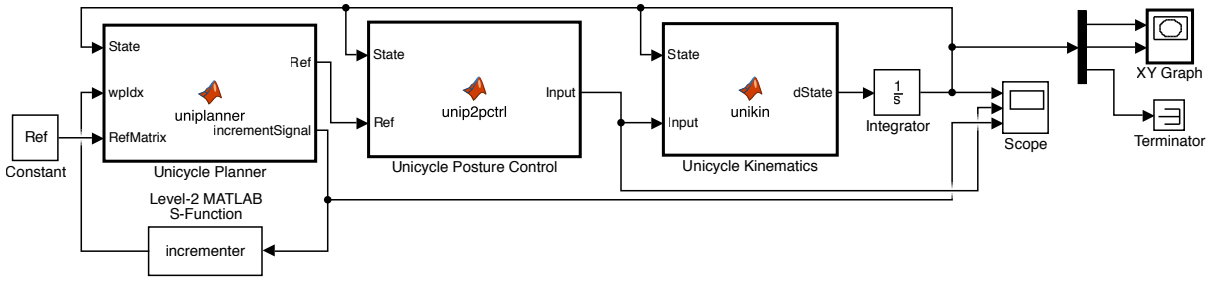

Figura 4: *Schema Simulink che consente di simulare il comportamento del veicolo controllato e supervisionato dal pianificatore.*

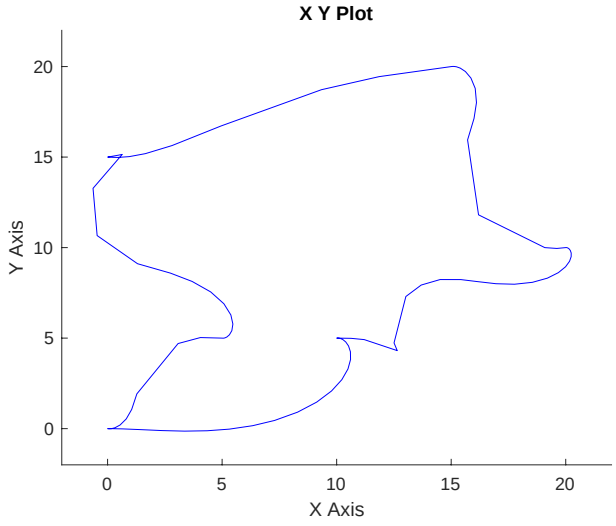

Figura 5: *Traccia della traiettoria eseguita dal veicolo.*