FIELD AND SERVICE ROBOTICS (FSR) – a.y. 2023/2024

Homework N.2

Vincenzo Palomba
P38000180

University of Naples Federico II
Department of Electrical Engineering and Information Technology

Abstract

The report summarizes the completion of the second homework assignment, consisting of five exercises on the mobile robot section. Topics covered include distributions, kinematic model, while the main part focus on the developing on a path planning algorithm, coupled with an input-output linearization control for a unicycle. Moreover it has been developed also a posture regulator based on polar coordinates for the same unicycle.

Exercise 1

The goal of the first exercise is to state whether each of the following distributions are *involutive* or not and, if possible, find the *annihilator* for each distribution. A distribution is involutive if the Lie bracket of every pair of its vector fields belong to the distribution itself.

Consider a matrix $G = [f_1, \dots, f_m]$, where the vector fields f_i of the distribution are arranged as columns. Let $F = [f_1, \dots, f_m, [f_1, f_2], \dots, [f_{m-1}, f_m]]$ be a matrix containing the vector fields of the distribution along with every possible Lie bracket between them. If the rank of F is equal to the rank of G, implying that every Lie bracket can be expressed as a linear combination, then the distribution is said to be involutive.

It has been developed a matlab function that compute all $m(m-1)/2$ possible Lie brackets (assuming the skew-symmetry properties $[f, g] = -[g, f]$):

```
if size(G,2) == 1  % if there is only one vector in the distribution
    B(:,1) = zeros(size(G));
else % if there are more vector fields in the distribution
    for i = 1 : size(G,2) % Loops to generate n*(n-1)/2 lie brackets
        % compute the lie bracket between the i-th vector and every j>i vector
        for j = i+1 : size(G,2)
            % Lie bracket matrix
            B(:,i) = lieBracket(G(:,i), G(:,j), x);
        end
    end
end

% Matrix with vector fields and lie brackets
F = [G, B];
```

The *annihilator* is the set of co-vectors such that $w\ G(x) = 0$ with $w = [w_1, ..., w_m]$. It is easy to assume the following, with some manipulation:

$$\begin{aligned} q^T H^T = (Hq)^T = 0^T \implies Hq = 0 \\ q^T H^T \implies Hq = 0 \end{aligned} \tag{1}$$

so if $w = q^T$ and $G(x) = H^T$, it yelds:

$$w\ G(x) = 0^T \implies G^T(x)w^T = 0 \tag{2}$$

The null space tranpose vectors of the transpose of the matrix $G(x)$ are the co-vectors spanning the annihilator. In order to easily compute the annihilator, it has been used the *symbolic toolbox command null*:

```
null(G')'
```

- The first distribution is:

$$\Delta_1 = \left\{ \begin{bmatrix} 3x_1 \\ 0 \\ -1 \end{bmatrix} \right\}, U \in \mathbb{R}^3 \tag{3}$$

Noticing that, by definition, since $[f, f] = 0$ (where f_1 is the vector field spanning Δ_1), and the zero vector belongs to any distribution by default, any one-dimensional distribution is *involutive*. That means that Δ_1 **is involutive**. The **annihilator** $\Delta_1^\perp$ of this distribution should have dimension $dim(\Delta^\perp) = n - dim(\Delta) = 2$. Doing the computation by hands, it yields:

$$\Delta_1^\perp = \left\{ \begin{bmatrix} \alpha & 0 & 3\alpha x_1 \end{bmatrix}, \begin{bmatrix} 0 & \beta & 0 \end{bmatrix} \right\} \tag{4}$$

in which α and β are to parameters. Running the matlab script *'distribution.m'*, the result is equivalent:

```
>> null(g1')'

ans =

[1/(2*x3), 0, 1]
[       0, 1, 0]
```

Notice that with the solver that i developed, α and β are equal to one. Moreover the vector $[1/(3x_1), 0, 1]$ can be multiplied by $3x_1$ (if $x_1 \neq 0$) to obtain the computed vector $[\alpha, 0, 3\alpha x_1]$.

- The second distribution is:

$$\Delta_2 = \left\{ \begin{bmatrix} 1 \\ 0 \\ x_2 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ x_1 \end{bmatrix} \right\}, U \in \mathbb{R}^3 \tag{5}$$

Taking into account the lie bracket $[f_1, f_2]$ (where f_1 and f_2 are the vector fields spanning Δ_2), the matrix $F = [f_1, f_2, [f_1, f_2]]$ has rank 2, so the vector field generated by the lie bracket of f_1 and f_2 belongs to the distribution Δ_2. That means Δ_2 **is involutive**. The **annihilator** $\Delta_2^\perp$ of this distribution should have dimension $dim(\Delta^\perp) = n - dim(\Delta) = 1$. It yields:

$$\Delta_2^\perp = \left\{ \begin{bmatrix} 0 & \alpha & 0 \end{bmatrix} \right\} \tag{6}$$

with α a free parameter. Likewise the code gives:

```
>> null(G2')'

ans =

[0, 1, 0]
```

- The third distribution is:

$$\Delta_3 = \left\{ \begin{bmatrix} 2x_3 \\ -1 \\ 0 \end{bmatrix}, \begin{bmatrix} x_2 \\ x_1 \\ 1 \end{bmatrix} \right\}, U \in \mathbb{R}^3 \tag{7}$$

Taking into account the lie bracket $[f_1, f_2]$ (where f_1 and f_2 are the vector fields spanning Δ_2), the matrix $F = [f_1, f_2, [f_1, f_2]]$ has rank 3, so also the vector field generated by the lie bracket of f_1 and f_2 is a basis to the distribution Δ_3. Notice that the case when $x_3 = \pm\sqrt{3/4}$ gives $det(F) = 0$, but to define a distribution as involutive it must be involutive $\forall x$. That means Δ_3 **is not involutive**. The **annihilator** $\Delta_3^\perp$ of this distribution should have dimension $dim(\Delta^\perp) = n - dim(\Delta) = 1$. It yields:

$$\Delta_3^\perp = \left\{ \begin{bmatrix} \alpha & 2x_3\alpha & -\alpha(x_2 + 2x_1x_3) \end{bmatrix} \right\} \tag{8}$$

while the code gives:

```
>> null(G3')'

ans =

[-1/(x2 + 2*x1*x3), -(2*x3)/(x2 + 2*x1*x3), 1]
```

Similar assumption to the first distribution can be made here.

Exercise 2: kinematic model of an omnidirectional robot

The omnidirectional robot mechanical system is subject to the following *pfaffian constraints*, where $q = [x\, y\, \theta\, \alpha\, \beta\, \gamma]$:

$$A^T(q) = \begin{bmatrix} \frac{\sqrt{3}}{2}\cos(\theta) - \frac{1}{2}\sin(\theta) & \frac{1}{2}\cos(\theta) + \frac{\sqrt{3}}{2}\sin(\theta) & l & r & 0 & 0 \\ sin(\theta) & -cos(\theta) & l & 0 & r & 0 \\ -\frac{\sqrt{3}}{2}\cos(\theta) - \frac{1}{2}\sin(\theta) & \frac{1}{2}\cos(\theta) - \frac{\sqrt{3}}{2}\sin(\theta) & l & 0 & 0 & r \end{bmatrix} \tag{9}$$

The **kinematic model** can be now computed taking taking a basis of $Null(A^T(q))$ such us $\{g_1(q), g_2(q), ..., g_{n-k}(q)\}$ obtaining the system in the form:

$$\dot{q} = G(q)u \tag{10}$$

with u the input vector and q the state vector. The matrix G is obtained with the *null* command on matlab, leading to the following matrix:

$$G(q) = \begin{bmatrix} -\frac{2rcos(\theta+\pi/6)}{3} & -\frac{2rsin(\theta)}{3} & -\frac{2rsin(\theta+\pi/3)}{3} \\ -\frac{2rcos(\theta-\pi/3)}{3} & -\frac{2rcos(\theta)}{3} & -\frac{2rcos(\theta+\pi/3)}{3} \\ -\frac{r}{3l} & -\frac{r}{3l} & -\frac{r}{3l} \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{11}$$

Now, let's take into account also the lie brackets between g_1, g_2, g_3 (the column of the kinematic matrix G), obtaining:

$$[g1, g2] = \begin{bmatrix} \frac{2\sqrt{3}r^2 sin(\theta+\pi/3)}{9l} \\ -\frac{2\sqrt{3}r^2 cos(\theta+\pi/3)}{9l} \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} [g2, g3] = \begin{bmatrix} -\frac{2\sqrt{3}r^2 cos(\theta+\pi/6)}{9l} \\ -\frac{2\sqrt{3}r^2 sin(\theta+\pi/6)}{9l} \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} [g1, g3] = \begin{bmatrix} \frac{2\sqrt{3}r^2 sin(\theta)}{9l} \\ -\frac{2\sqrt{3}r^2 cos(\theta)}{9l} \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{12}$$

in order to compute the Lie Bracket between the vector field it has been used the following snippet of code:

```
function h = lieBracket(f,g,x)

    % Jacobian of the two vector fields respect to the symbolic variables
    Jg = jacobian(g,x);
    Jf = jacobian(f,x);

    % Output vector field
    h = Jg * f - Jf * g;

end
```

Obtaining the following distribution Δ_a, containing the column vector of the matrix G and their lie brackets:

$$\Delta_a = span\{g1, g2, g3, [g1, g2], [g2, g3], [g1, g3]\} \tag{13}$$

With the matlab command *rank* i can find the dimension of Δ_a that is equal to 5. That means that the number of generalized coordinates $n = 6$ is greater then the dimension of the distribution $\nu = 5$, while the number of admissible direction $m = 3$ is less then the dimension of the distribution:

$$m < dim(\Delta_a) < n \tag{14}$$

leading to the conclusion that the kinematic system is not controllable, thus some model configurations are prevented. Moreover the system has some *holonomic constraint* and only partially *integrable constraints*. Beside that, the system is still **nonholonomic**.

Exercise 3: path planning algorithm for a unicycle

The code containing the unicycle path planning algorithm is in the matlab file *'pathPlanning.m'*. As timing law, the go-to choice was the trapezoidal velocity profile with $s(t_i) = 0$ and $s(t_f) = 1$, assuming $t_i = 0$ and $t_f = 10s$. As a result we obtain the following arc length expression:

$$s(t) = \begin{cases} s_i + \frac{1}{2}a_c(t - t_i)^2 & t_i \leq t \leq t_c + t_i \\ s_i + a_c t_c(t - t_i - tc/2) & t_c + t_i < t \leq t_f - t_c \\ s_f + \frac{1}{2}a_c(t_f - t)^2 & t_f - t_i < t \leq t_f \end{cases} \tag{15}$$

The timing law matlab function has been taken from my *Foundations of Robotics* prioject and modified in order to allow the scaling by T:

```
function [sp, spdot, spddot, t] = trapezioidalProfile(ti, tf, ts, T)
```

The path law is instead based on a *cubic cartesian polynomial*, developed in the function:

```
function [x, y, theta, v, w] = cartesianPolyPlan(s, sdot, T, qi, qf)
```

in which i choose the following parameters, taking into account that $q_i = [x_i \ y_i \ \theta_i]^T = [0 \ 0 \ 0]^T$ and $q_f = [x_f \ y_f \ \theta_f]^T = [0.68 \ 0.71 \ 0.16]^T$ (generated automatically by the code such that $norm(q_f, q_i) = 1$), while $k = 2$:

$$\begin{array}{cc} x(s_i) = x_i = 0 & \alpha_x = kcos(\theta_f) - 3x_f \\ x(s_f) = x_f = 0.68 & \alpha_y = ksin(\theta_f) - 3y_f \\ y(s_i) = y_i = 0 & \beta_x = kcos(\theta_f) + 3x_i \\ y(s_f) = y_f = 0.71 & \beta_y = ksin(\theta_f) + 3y_i \end{array} \tag{16}$$

Regarding the *uniform scaling*, it has been developed an algorithm in which is possible to obtain the optimal value of T that minimize the trajectory time, maximizing the velocity of the unicycle, respecting speed constraints.

```
function [Topt, T] = optScaling(ti, tf, ts, qi, qf, vmax, wmax)
    %% Optimal T
    % Timing law with uniform scaling, using tf - ti as scaling factor
    [s, sdot] = trapezioidalProfile(ti, tf, ts, tf - ti);

    % v and w without scaling to obtain v(s)*sdot and w(s)*sdot
    [~, ~, ~, v, w] = cartesianPolyPlan(s, sdot, 1, qi, qf);

    % Computing the optimal period to be in the velocity bounds (v(s)*sdot
    % / vmax = T)
    Topt = max(max(abs(v))/vmax, max(abs(w))/wmax);

    %% Suboptimal T
    % Timing law without uniform scaling, using tf - ti as scaling factor
    [s, sdot] = trapezioidalProfile(ti, tf, ts, tf - ti);

    % v and w without scaling to obtain v(s)*sdot and w(s)*sdot
    [~, ~, ~, v, w] = cartesianPolyPlan(s, sdot, 1, qi, qf);

    % Computing the new period only if needed (i.e. the velocities are
    % greater then the bounds)
    if max(abs(v))>vmax || max(abs(w))>wmax
        T = max(max(abs(v))/vmax, max(abs(w))/wmax);
    else
        T = tf - ti;
    end
end
```

The figure 1 shows the timing law $s(t)$ and its derivatives $\dot{s(t)}$ and $\ddot{s(t)}$, while figure 2 shows the *heading and angular velocities*. Notice how both velocities, due to *uniform scaling*, **don't exceed the limit imposed of** $|v(t)| \leq v_{max} = 2 \ m/s$ **and** $|w(t)| \leq w_{max} = 1 \ rad/s$. Finally, in figure 3, one can notice the **geometric path of the unicycle**.

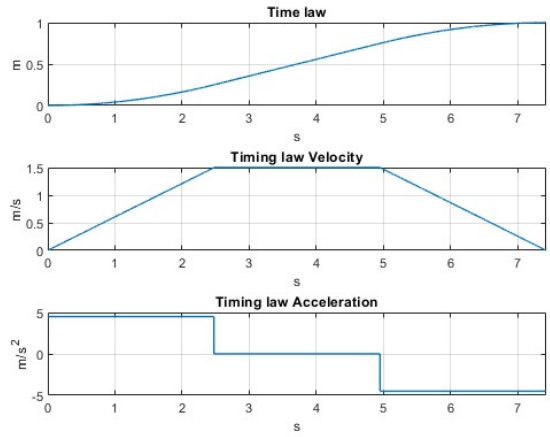


Figure 1: Timing law

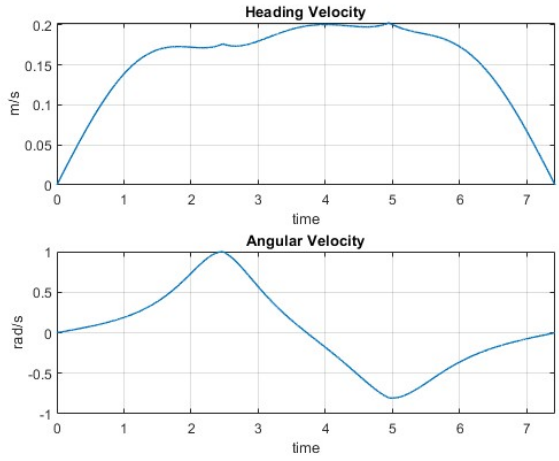


Figure 2: Velocities

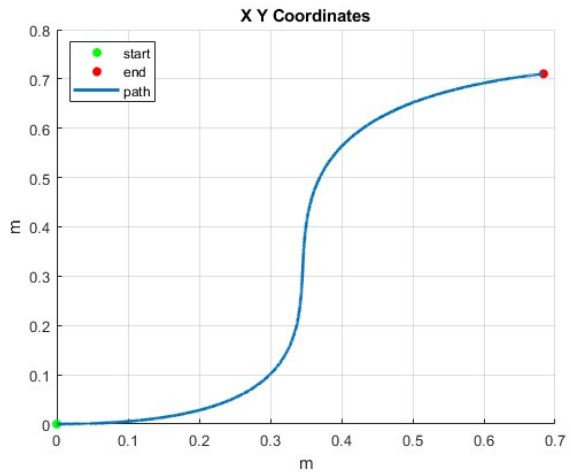


Figure 3: Geometric path

The value $k = 2$ is an optimal choice for the unicycle trajectory, due to the smoothness of the geometric path, coupled with a total time $T = 7.42\ s$ and a bounded velocity profile. Taking as example respectively $k = 10$ and $k = 0.1$ in figure 4 and figure 5, one can notice that in the first case the total travel time is too high, while in the second scenario, the trajectory lacks smoothness (think about the difference for example with the

reeds-sheep curves), particularly when considering the integration of additional paths, in particular the robot should stop and change his orientation.

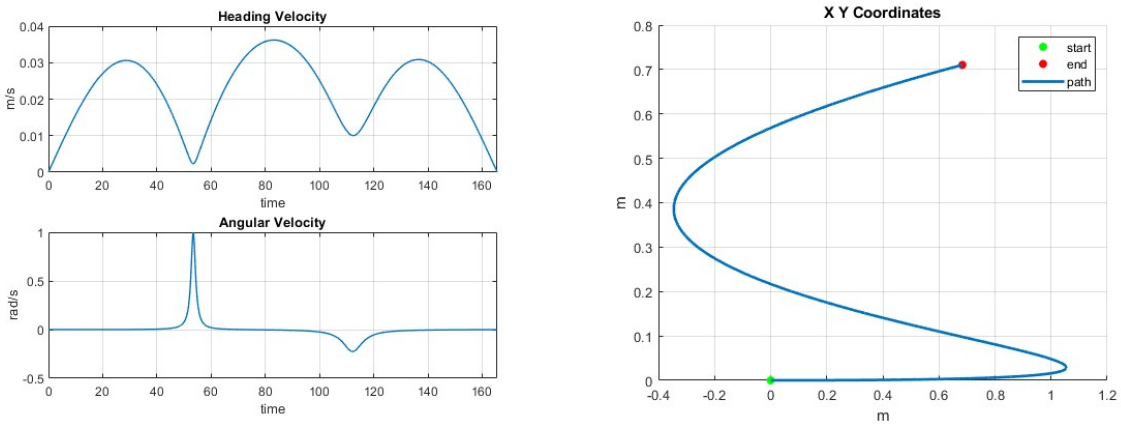


Figure 4: Trajectory with k = 10

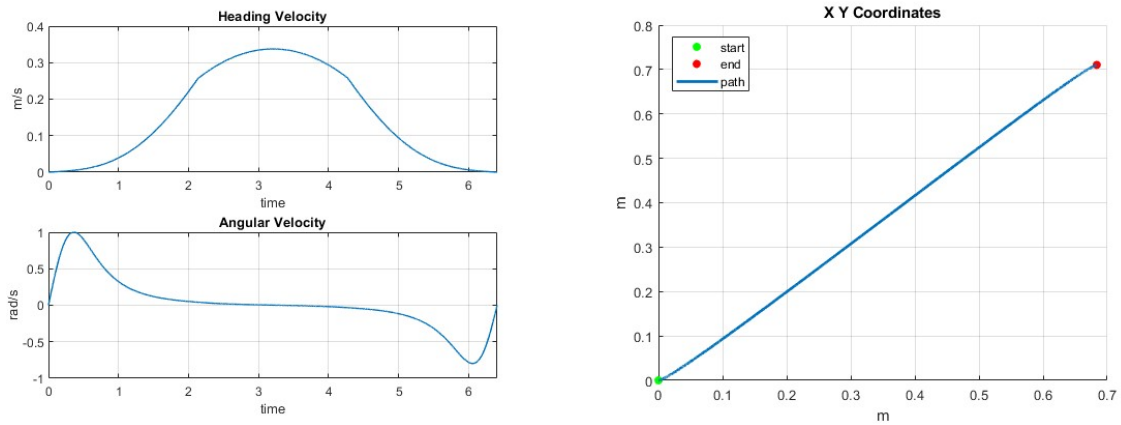


Figure 5: Trajectory with k = 0.1

Exercise 4: input/output linearization control for a unicycle

To accomplish the goal of developing an *input output linearization control*, the trajectory was firstly assigned as a x y θ function. Then it was mapped on y_1 and y_2, that are the x and y components of a point B at a distance $b = 0.1$ from the unicycle center C. So, the desired trajectory that our system has to track is:

$$\begin{aligned} y_{1d} &= x_d + b\cos(\theta_d) \\ y_{2d} &= y_d + b\sin(\theta_d) \end{aligned} \tag{17}$$

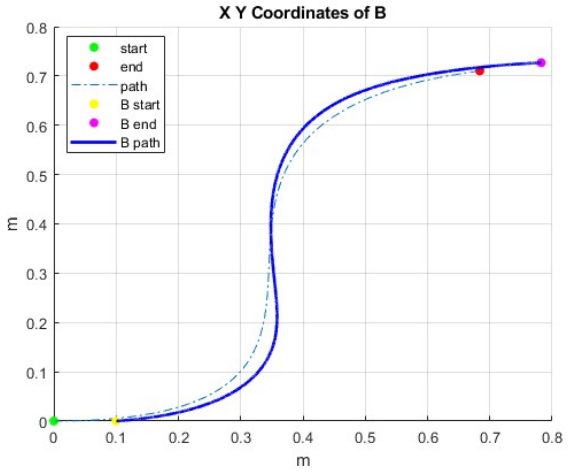
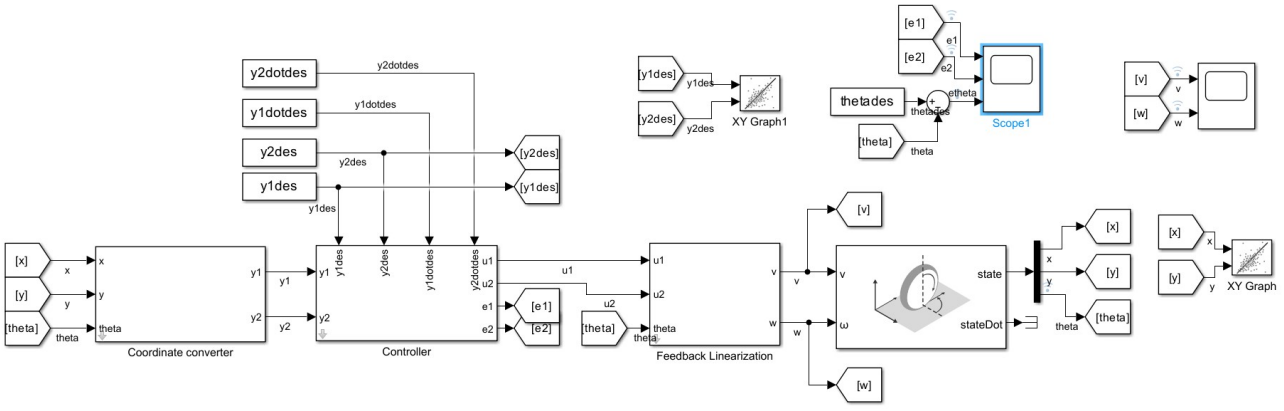


Figure 6: Geometric path of B

Now, it is possible to feedback linearize the system, through the use of the matrix $T(\theta)$, that maps the kinematic input to the x and y velocities of the point B. In this way the inputs of this new system become u_1 and u_2, such as:

$$\begin{aligned} u_1 &= \dot{y_{1d}} + k_1(y_{1d} - y_1) \\ u_2 &= \dot{y_{2d}} + k_2(y_{2d} - y_2) \end{aligned} \tag{18}$$

The model was developed in the simulink file *'inputOutputControl.slx' (run also 'pathPlanning.m')*, in which the unicycle kinematic model (Robotics Toolbox - Mobile Robot Algorithms) was used to easily simulate the robot kinematics.



As one may notice, this approach controls the position of the point B only, leaving the orientation uncontrolled. The real input for the unicycle kinematic model is very similar to the input calculated as a function of the flat outputs and their derivatives. Therefore, the flat input was putted into an ideal simulation system, starting at time zero with the robot in the initial configuration q_i. In order to **track the desired trajectory of B**, the choice $k_1 = 5$ and $k_2 = 5$ seems to be a good fit for the control loop, giving small errors and robustness, leading to the following results:

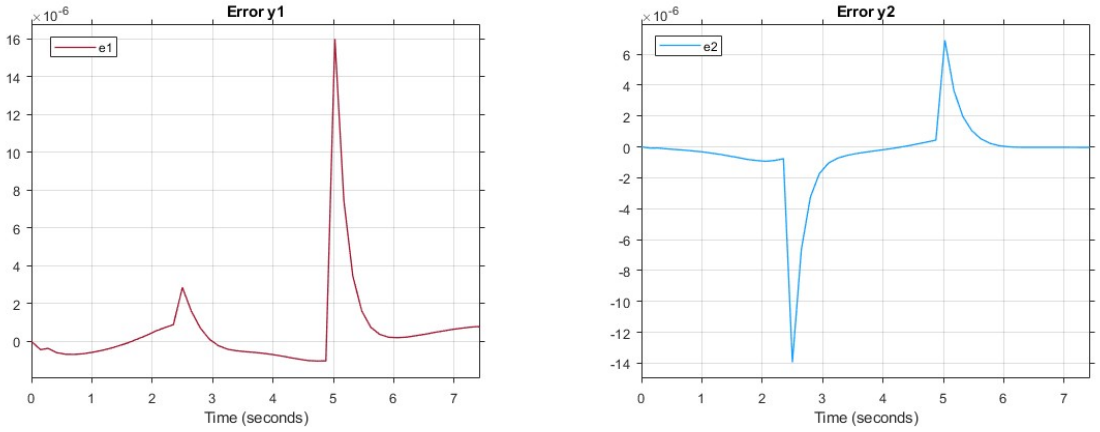


Figure 7: y1 error and y2 error

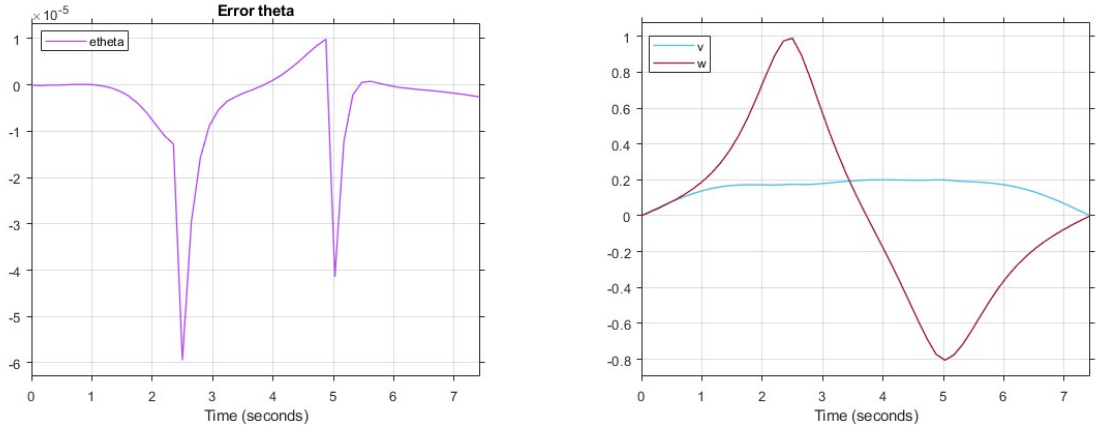
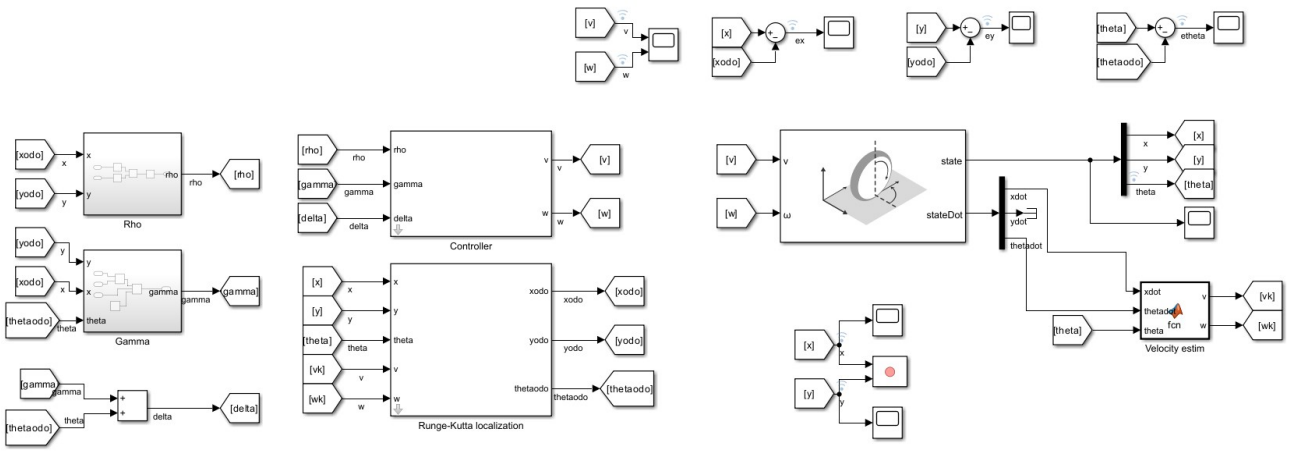


Figure 8: theta error and input velocities

Exercise 5: unicycle posture regulator based on polar coordinates with Runge-Kutta odometric localization

The scope of the last exercise is to implement via software a unicycle *posture regulator* based on polar coordinates, with the state feedback computed through the *Runge-Kutta odometric localization method*. Starting and final configurations are $q_i = [1\ 1\ \pi/4]$ and $q_f = [0\ 0\ 0]$. The model is contained in the simulink file *'postureRegulator.slx'*.



In order to simulate the real model more accurately, in which the state of the robot is not measured, but only

the velocities can be measured using the encoders, a *localisation function* was implemented using the *second-order Runge-Kutta method*. The code was implemented in the *'Runge-Kutta localization'* block, following the relations below:

$$\begin{aligned} x_{k+1} &= x_k + v_k T_s cos(\theta_k + w_k Ts/2) \\ y_{k+1} &= y_k + v_k T_s sin(\theta_k + w_k Ts/2) \\ \theta_{k+1} &= \theta_k + \theta_k + w_k Ts \end{aligned} \tag{19}$$

Note how in the model, unlike in reality, the velocities are obtained by means of a function *'(Velocity estim)'* which, from the state and its derivatives, derives v_k and w_k. In this way, once the x_{k+1}, y_{k+1} and θ_{k+1} have been estimated, the control law can be implemented:

$$\begin{aligned} v &= k_1 \rho cos(\gamma) \\ w &= k_2 \gamma + k_1 sin(\gamma) cos(\gamma)(1 + k_3 \delta/\gamma) \\ k_1 &= 0.25 \; k_2 = 0.25 \; k_3 = 0.25 \\ T_s &= 0.05 \end{aligned} \tag{20}$$

The choice of coefficients and sampling time was dictated by a trial-and-error approach, in which different combinations were tried in order to minimise the error and the tracking time. In fact increasing too much the gains can lead to some unwanted trend of θ, that oscillates too much, as can be seen in figure 12 Moreover,the control law is defined everywhere, the problem is only in the mapping between Cartesian and polar coordinates, where in $\rho = 0$ there is a singularity.

It can be seen from the figure 9 that the localization error is not brought exactly to zero at steady state, because the second order Runge-Kutta localization give an approximation of the real generalised coordinates q, but still is very small. It can be seen from Figure 10 that i chose not to exceed the limit on w and v defined in exercise 3, and of course in this way we bring q to regulation reference in more time. In fact it can be noted from figure 11 that θ go to zero approximately in $100s$, while x, y approximately in $40s$. This behaviour leads to the choice of $100s$ as simulation time. In figure 11 we also notice that with the posture regulator we don't care about the trajectory (both position and orientation) of the unicycle, but only the initial and final configuration and as seen the trajectory itself is not optimal.

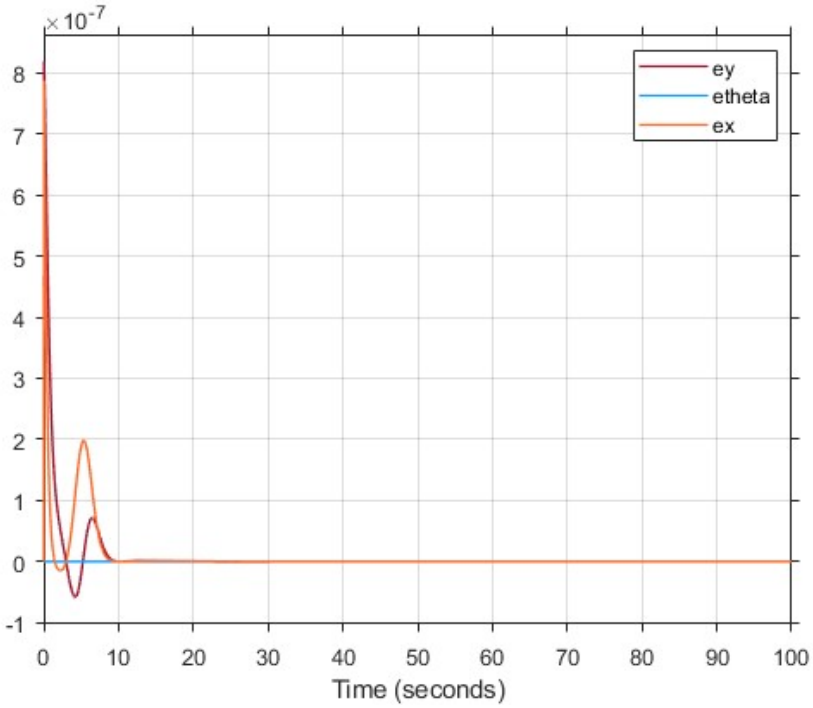


Figure 9: error between q and q_{loc}

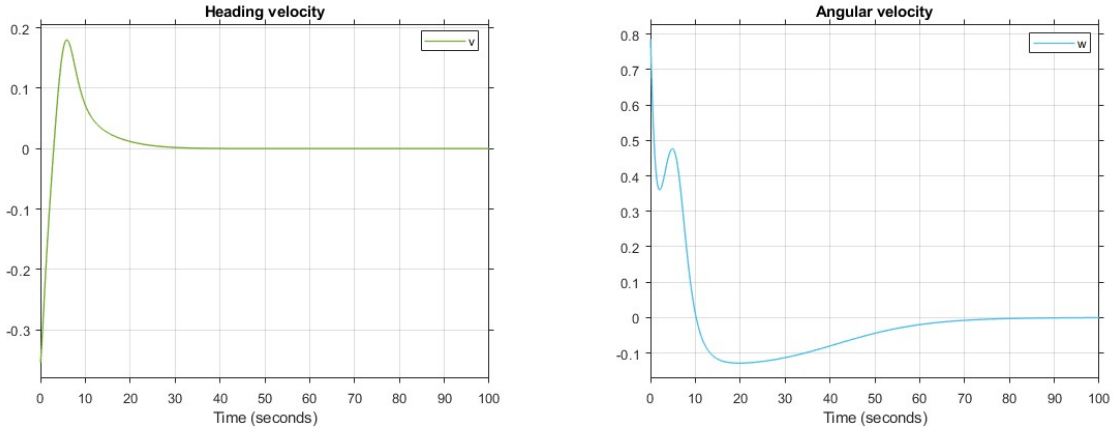


Figure 10: Heading velocity and Angular velocity

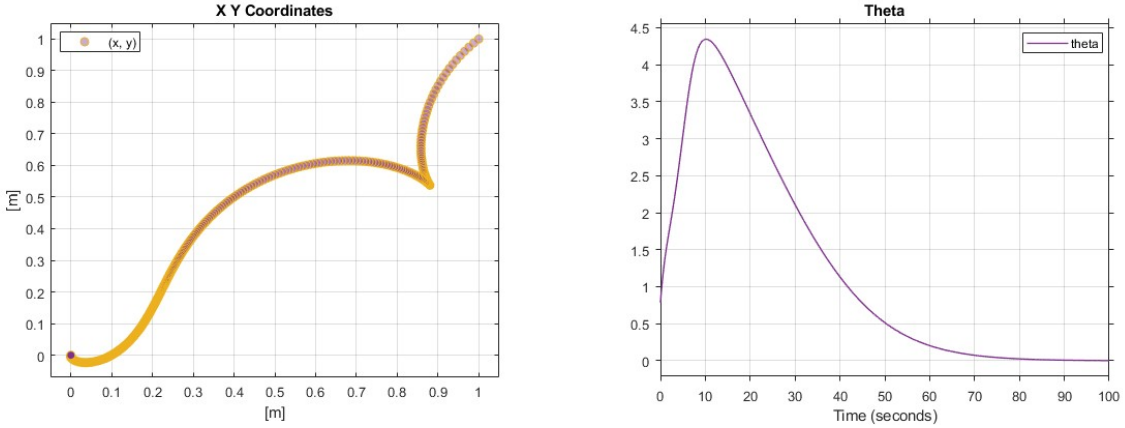


Figure 11: x and y coordinates and orientation with $k_1 = k_2 = k_3 = 0.25$

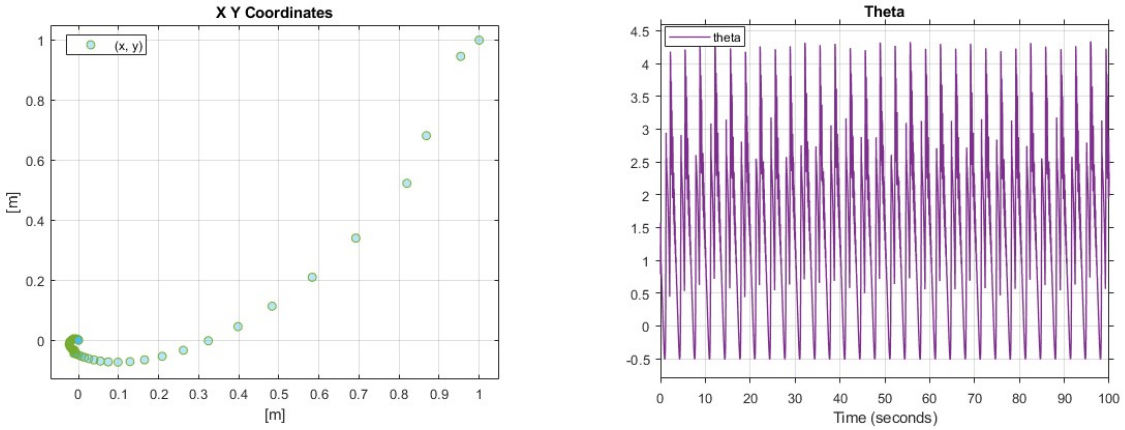


Figure 12: x and y coordinates and orientation with $k_1 = k_2 = k_3 = 5$