AMR Lectures part1

Motion control of WMR's:

In previous lectures we've discussed path and trajectory planning. This knowledge will need us to discuss the motion, which is executed according to trajectory and path. Motion control consider to problems:

→ Trajectory Tracking → Posture Regulation.

Trajectory Tracking.

- Motion Control:
  - → A desired motion is assigned for the WMR, and the associated nominal inputs have been computed (e.g. trajectory planning)
  - → To execute the desired motion, we need feedback control because the application of nominal inputs in open-loop would lead to very poor performance. (i.e. there is no ideal robot in the real world. In order to fix problems in terms of control, we will use feedback control). It is correcting the motion.

Note1: We will use only KINEMATIC MODELS in order to design control law. Why we don't use dynamic model?

Reason 1: We ride dynamic models in WMR's by using the control laws designed according to the Kinematic Model, easily.

WMR's are relatively simple, because they are single body and they don't have kinematic chains. This means, inertia doesn't depend on the configuration. Inertia of MR's is always the same (i.e. Inertia matrix is not a function of q → centrifugal and coriolis terms disappear, I→const.)

Note: I → inertia matrix.

Reason: → Typically MR's come with low-level PID control loop (built in system).

It means, we don't apply any torque to the motor. Instead of torque, the system demands velocity as input. By using the reference velocity, PID loop will do what is necessary to track the motion.

Block schemes are demonstrated in Figure 1:

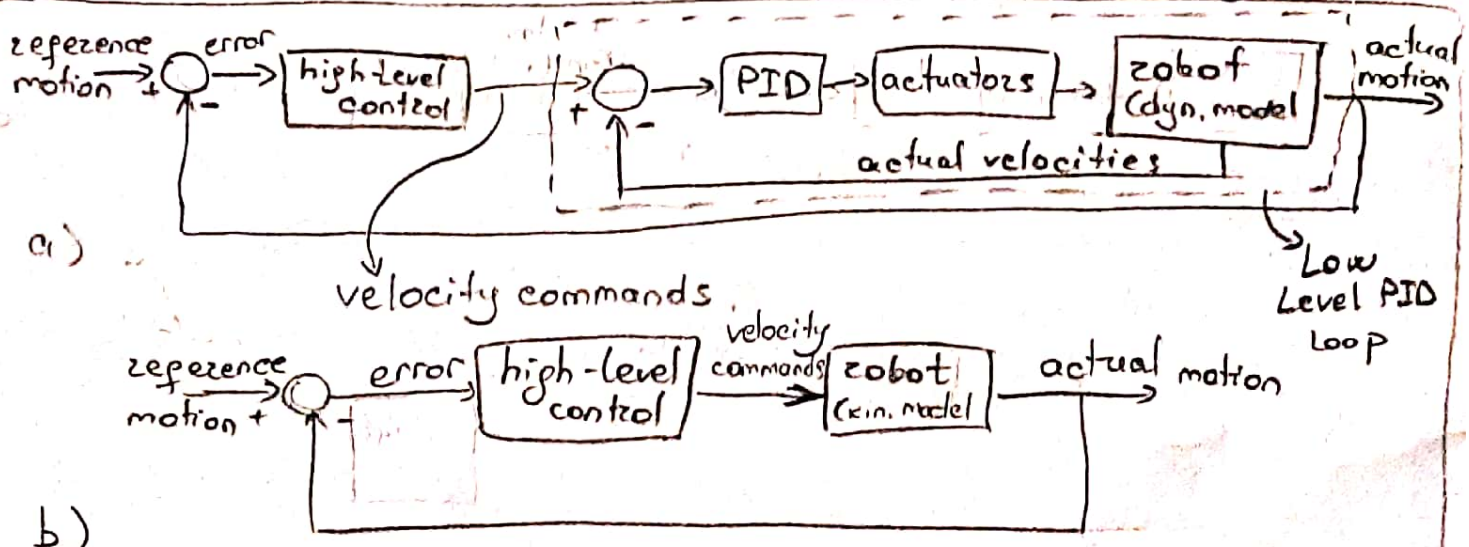

Figure 1
a: Actual control scheme for localization
b: Equivalent control scheme for localization

Note 2: Localization problem: Robot needs to localize itself in order to compute the error between the desired trajectory and where actually robot is.

We will use unicycle model in order to explain trajectory tracking. Unicycle model is given as below:

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} \cos\theta \\ \sin\theta \\ 0 \end{pmatrix} v + \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \omega \quad (1)$$

$$g_1 = \begin{pmatrix} \cos\theta \\ \sin\theta \\ 0 \end{pmatrix} (2) \qquad g_2 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} (3)$$

We will analyze 2 types of feedback control:

→ State error feedback, which uses states of the robot in order to control the robot

→ Output error feedback, which uses outputs of the robot in order to control the robot.

By saying the output, we mean cartesian trajectory of robot, because we determine the robot's trajectory by them, while the robot moves on the ground (x, y). Additionally, we introduce x_d and y_d, which are desired trajectory of the robot.

- State Error feedback:

The unicycle must track a Cartesian desired trajectory $(x_d(t), y_d(t))$ that is admissible. Unicycle must track such trajectory that is feasible, because of local mobility problem. In other words, there exist v_d (desired driving velocity) and w_d (desired steering velocity) such that

$$\dot{x}_d = v_d \cos\theta_d \quad (4)$$

$$\dot{y}_d = v_d \sin\theta_d \quad (5)$$

$$\dot{\theta}_d = w_d \quad (6)$$

Notice that, v_d and w_d are inputs to the system. (4), (5), (6) define our target unicycle (i.e. the vehicle we want to track). In other words, (4), (5), (6) define a unicycle under the nominal inputs.

Notes: Nominal inputs are inputs that associated to desired trajectory.

We know that unicycle is differentially flat system. That means, we can reconstruct states and inputs by using flat outputs which are x_d and y_d for unicycle (i.e. If we are given desired cartesian trajectory, we can compute whole desired states and inputs (Remember Differential Flatness)

According to **Differential Flatness**

$$\theta_d(t) = \text{ATAN2}(\dot{y}_d(t), \dot{x}_d(t)) + k\pi, \quad k=0,1 \quad (7)$$

$$v_d(t) = \pm\sqrt{\dot{x}_d^2(t) + \dot{y}_d^2(t)} \quad (8)$$

$$\omega_d(t) = \frac{\ddot{y}_d(t)\dot{x}_d(t) - \ddot{x}_d(t)\dot{y}_d(t)}{\dot{x}_d^2(t) + \dot{y}_d^2(t)} \quad (9)$$

Notice that, $v_d(t)$ and $\omega_d(t)$ are reference inputs to the system.

We can denote, the desired and current states as below, which are $q_d(t)$ and $q(t)$, respectively:

$$q_d(t) = \begin{pmatrix} x_d(t) \\ y_d(t) \\ \theta_d(t) \end{pmatrix} \quad (10) \qquad q(t) = \begin{pmatrix} x(t) \\ y(t) \\ \theta(t) \end{pmatrix} \quad (11)$$

By comparing $q_d(t)$ and $q(t)$, it is possible to compute an error vector that can be fed to the controller. However, rather than using directly the difference between $q_d(t)$ and $q(t)$, it is convenient to define the tracking error as below.

$$e = \begin{pmatrix} e_1 \\ e_2 \\ e_3 \end{pmatrix} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x_d - x \\ y_d - y \\ \theta_d - \theta \end{pmatrix} \quad (12)$$

e is the error vector which includes e_1 and e_2 (position errors) and e_3 (angular error).

The error can be demonstrated as in Figure 2. e_p is the positional error in Reference Frame.

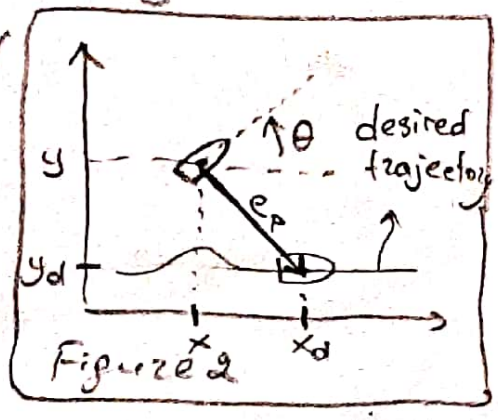

Figure 2

According to (12) and Figure (2), we can say that, the first 2 elements of e (e_1 and e_2) are representation of e_p vector in the frame that aligned to the current position of unicycle (current frame).

By using (12) we get following equations:

$$e_1 = (x_d - x)\cos\theta + (y_d - y)\sin\theta \quad (14)$$

$$e_2 = (x_d - x)\sin\theta + (y_d - y)\cos\theta \quad (15)$$

$$e_3 = \theta_d - \theta \quad (16)$$

In order to compute the changes in the error vector, we will differentiate e_1, e_2 and e_3 with respect to time.

Before going ahead we need to generate the following expressions from the equation (1).

$$\dot{x} = v\cos\theta \quad (17)$$

$$\dot{y} = v\sin\theta \quad (18)$$

$$\dot{\theta} = \omega \quad (19)$$

We generate derivative of e wrt time as below, by using (4), (5), (6), (17), (18) and (19) (see Appendix 9.1):

$$\dot{e}_1 = v_d\cos e_3 - v + e_2\omega \quad (20)$$

$$\dot{e}_2 = v_d\sin e_3 - e_1\omega \quad (21)$$

$$\dot{e}_3 = \omega_d - \omega \quad (22)$$

We see that the system is time varying and nonlinear system, which is complicated to control. The main purpose of controlling the system is making the error zero. If error is zero, it means current and desired trajectories are same.

- How to design control law:

→ 1st approach:

- Design via approximate Linearization. The main idea of this method is stabilize the error in its origin (i.e. make error zero). We can follow the following steps as idea:
  - → One method to stabilize the nonlinear system at the origin or at any equilibrium is to compute approximate linearization of the system at the point.
  - → Then we can design control law for the linearized system
  - → If we achieve assymptotic stability, we can also translate this result to the original system (which will be assymptotically stable).
  - → This method is called indirect Lyapunov method which we study linearized system in place of the original system

Before going ahead we need to remember how does linearization work. The nonlinear system is defined as below:

$$\dot{x} = \varphi(x, u) \quad (23)$$

If we linearize it, the result will be:

$$\dot{x} = Ax + Bu \quad (24)$$

where

$$A = \left.\frac{\partial \varphi}{\partial x}\right|_{\substack{x=0 \\ u=0}} \quad (25) \qquad B = \left.\frac{\partial \varphi}{\partial u}\right|_{\substack{x=0 \\ u=0}} \quad (26)$$

(24) is the linear approximation at 0.

Note 4: Origin means where the error is zero.

Approximate linearization method can be generalized by following steps.

→ We need to make the reference trajectory an unforced equilbrium for the error dynamics. However, if we check the equations (20), (21) and (22), we will see they are not unforced equilbrium. Because, if we set v and ω to zero, e will change anyway, if v_d and θ_d are not zero.

→ In order to solve it we introduce the following input transformations:

$$u_1 = v_d \cos e_3 - v \quad (27)$$

$$u_2 = \omega_d - \omega \quad (28)$$

It is invertible transformation because we can generate old inputs by new, new inputs by old inputs.

→ In order to define error derivatives in terms of new (reconstructed) inputs, we will use (27) and (28) in (20), (21) and (22):

(27), (28) => (20): $\dot{e}_1 = u_1 + (\omega_d - u_2) e_2$ (29)

(27), (28) => (21): $\dot{e}_2 = \omega_d \sin e_3 + (u_2 - \omega_d) e_1$ (30)

(28) => (22): $\dot{e}_3 = \omega_d + u_2 - \omega_d = u_2$ (31)

Using these expressions we can generate the following matrix representation for $\dot{e}$.

$$\dot{e} = \begin{pmatrix} \omega_d e_2 \\ -\omega_d e_1 + \omega_d \sin e_3 \\ 0 \end{pmatrix} + \begin{pmatrix} 1 & -e_2 \\ 0 & e_1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} \quad (32)$$

Where

$$f(e,t) = \begin{pmatrix} \omega_d e_2 \\ -\omega_d e_1 + \omega_d \sin e_3 \\ 0 \end{pmatrix} \quad (33)$$

$$G(e) = \begin{pmatrix} 1 & -e_2 \\ 0 & e_1 \\ 0 & 1 \end{pmatrix} \quad (34)$$

$$u = \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} \quad (35)$$

$f(e,t)$, $G(e)\cdot u$ are representing drift term (nonlinear, time varying), input term (nonlinear in e, linear in u), respectively. We can rewrite (32) as below:

$$\dot{e} = f(e,t) + G(e)\, u \quad (36)$$

Notice that, drift term collects everything which are not related with input.

→ Now we can apply linearization as we mentioned above (check the equations (23) to (26))

In our case, A will be $\frac{\partial \varphi}{\partial e}\Big|_{\substack{e=0\\u=0}}$ and B will be $\frac{\partial \varphi}{\partial u}\Big|_{\substack{e=0\\u=0}}$.

$$A = \frac{\partial \varphi}{\partial e}\Big|_{\substack{e=0\\u=0}} = \frac{\partial f}{\partial e}\Big|_{\substack{e=0\\u=0}} + \frac{\partial G}{\partial e}\overrightarrow{0}\Big|_{\substack{e=0\\u=0}} = \frac{\partial f}{\partial e}\Big|_{\substack{e=0\\u=0}} =$$

$$= \begin{pmatrix} 0 & \omega_d & 0 \\ -\omega_d & 0 & v_d \cos e_3 \\ 0 & 0 & 0 \end{pmatrix}\Bigg|_{\substack{e=0\\u=0}} = \begin{pmatrix} 0 & \omega_d & 0 \\ -\omega_d & 0 & v_d \\ 0 & 0 & 0 \end{pmatrix} \quad (37)$$

$$B = \frac{\partial \varphi}{\partial u}\Big|_{\substack{e=0\\u=0}} = \frac{\partial f}{\partial e}\overrightarrow{0}\Big|_{\substack{e=0\\u=0}} + \frac{\partial G(e)\,u}{\partial u}\Big|_{\substack{e=0\\u=0}} = G(e)\Big|_{\substack{e=0\\u=0}} =$$

$$= \begin{pmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{pmatrix} \quad (38)$$

According to (37) and (38) we can write linearized system as below:

$$\dot{e} = \begin{pmatrix} 0 & \omega_d & 0 \\ -\omega_d & 0 & v_d \\ 0 & 0 & 0 \end{pmatrix}\begin{pmatrix} e_1 \\ e_2 \\ e_3 \end{pmatrix} + \begin{pmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{pmatrix} \quad (39)$$

(39) represents the approximate linearization at the origin of the error dynamics. This describes the evolution of error, provided that error is not large (Because we have linearized the error, by assumption of being close to linearization point).

On the other hand, (because of the presence of ω_d and v_d, it is still time varying system. Thus, it is time variant linear system. If ω_d and v_d are constant, the system will be time invariant linear system.

→ Now we can stabilize the given system. In order to do it, we will compute u_1 and u_2 (inputs) and feedback in such a way that, the system will be asymptotically stable and e will be asymptotically stable point. Since this is a linear system, if it is asymptotically stable, it is also exponentially asymptotically stable.

We will control the system by linear feedback in order to stabilize it. Stabilization by state feedback has such scheme that is demonstrated in Figure 3. According to the Figure 3, we see that:

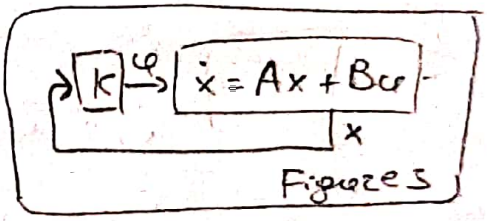

Figure 3

$$u = Kx \quad (40)$$

On the other hand, we know the linearized system is:

$$(24) \Rightarrow \dot{x} = Ax + Bu$$

If we use (40) in (24):

$$\dot{x} = Ax + BKx = (A+BK)x \quad (41)$$

Problem of stabilizing the system by static feedback becomes the problem of finding matrix K such that $(A+BK)$ has eigenvalues with negative real part.

In this case we propose such K matrix as below:

$$u = Ke = \begin{pmatrix} -k_1 & 0 & 0 \\ 0 & -k_2 & -k_3 \end{pmatrix} \begin{pmatrix} e_1 \\ e_2 \\ e_3 \end{pmatrix} \quad (42)$$

Because the system has 2 inputs K has 3 rows. According to (42), we can say that u_1 is related with e_1 and u_2 is related with e_2 and e_3. Notice that u is the reconstructed input. In order to use this control law for orginal inputs, we can invert results, easily.

If we apply results of (42) to (29), (30), (31):

$$\dot{e}_1 = \omega_d e_2 - k_1 e_1 \quad (43)$$

$$\dot{e}_2 = -\omega_d e_1 + v_d e_3 \quad (44)$$

$$\dot{e}_3 = -k_2 e_2 - k_3 e_3 \quad (45)$$

The equations system (43), (44), (45) is called closed loop error dynamics. Using these equations:

$$\dot{e} = A(t)\, e = \begin{pmatrix} -k_1 & \omega_d & 0 \\ -\omega_d & 0 & v_d \\ 0 & -k_2 & -k_3 \end{pmatrix} \begin{pmatrix} e_1 \\ e_2 \\ e_3 \end{pmatrix} \quad (46)$$

Notice that, reconstructed inputs dissappeared. By generating (46), we get state based system that is the same matrix as (A+BK) which we want to find eigenvalues of it.

We want to find K_1, K_2, K_3 in such way that $A(t)$ matrix has eigenvalues in the left half plane (i.e. $A(t)$ is Hourwitz matrix)

Notes: Hourwitz matrix has eigenvalues only the left half plane.

* We know $A(t)$ is time variant matrix. Sometimes it is constant, but due to v_d and ω_d it is time variant matrix. Notice that, eigenvalues of matrices matter for time invariant systems. However our system is time variant system. (we will come back to this)

→ At this step we will investigate K_1, K_2, K_3 gains that make the system stable. In order to do that we need to find characteristic formula of $A(t)$:

$$(\lambda I - A) = \begin{pmatrix} \lambda + k_1 & \omega_d & 0 \\ \omega_d & \lambda & -v_d \\ 0 & k_2 & \lambda + k_3 \end{pmatrix} \quad (47)$$

$$\det(\lambda I - A) = (\lambda + k_1)(\lambda(\lambda + k_3) - k_2 v_d) + \omega_d^2(\lambda + k_3) \quad (48)$$

In (48), we assume: $k = k_1 = k_3$ (49)

$$\det(\lambda I - A) = (\lambda + k)(\lambda^2 + \lambda k - k_2 v_d + \omega_d^2) \quad (50)$$

According to (50), we have result for λ_1:

$$\lambda_1 = -k \quad (51)$$

We will use the following expression as a generic formula for trynomials:

$$\lambda^2 + 2a\xi\lambda + a^2 = \lambda^2 + \lambda k - k_2 v_d + \omega_d^2 \quad (52)$$

(52) will give us imaginary eigenvalues.

We know that we can demonstrate imaginary values as in Figure 4. They have to have negative real values, in order to make the system stable.

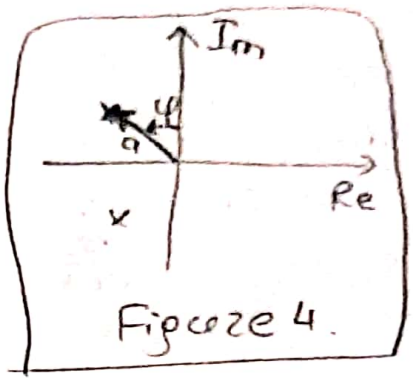

Figure 4.

According to generic formula that is given by the equation (52):

$$\xi = \sin\varphi \quad (53)$$

On the other hand we know that K must be greater than zero, because λ_1 must be negative. Additionally angle φ must change between 0 and $\pi/2$, otherwise the vector will point right hand-side of real axis (Re). If it will point right half-plane, then eigen values will have positive real part. Notice that, a is the length of the vector points to eigen value. According to the equation (52) we can generate the following expressions:

$$k = 2a\xi \quad (54)$$

$$\omega_d^2 - k_2\omega_d = a^2 \quad (55)$$

Because, φ must change between 0 and $\pi/2$, due to the equation (53):

$$0 < \xi < 1 \quad (56)$$

Additionally we know that K must be greater than zero. Because of (54):

$$a > 0 \quad (57)$$

On the other hand we can compute K_2 gain, by using (55):

$$k_2 = \frac{\omega_d^2 - a^2}{\omega_d} \quad (58)$$

To sum up, choosing a and ξ (damping factor) as in equations (56) and (57), respectively, will provide us eigenvalues with negative real part. On the other hand, we also computed gains by equations (54) and (58).

Notice that, eigenvalues are constant and K_2 is time varying (i.e. gains are time variant).

From stability point of view, having constant eigenvalues does not guarantee that the system is stable. If system is time invariant, then having eigenvalues in left half-plane guarantees that the system is assymptotically stable. However, this is not sufficient condition for time variant systems. There are tons of systems have eigenvalues in left half plane which are not stable, because they are time-variant systems.

There are two caviates:

1st caveat:

Choosing k_1, k_2, k_3 gains as we found make eigenvalues constant, but this does not guarantee that the system will be assymptotically stable. This may guarantee assymptotic stability for those trajectories that makes $A(t)$ matrix time invariant (e.g. circular, linear trajectories).

2nd caveat:

Even if the time variant system is stable we should remember that, we designed the control law by using Lyapunov's indirect method. In other words, we used approximate linearization in order to design control law. We designed controller for the linear system. Thus, the original system will be only LOCALLY stable. The result we obtain will only be valid for CERTAIN INITIAL CONDITIONS, where initial conditions are sufficiently close to the origin.

There are 2 possible scenarios that this control scheme may not work!

1st scenario:

If v_d and ω_d are time variant and they change so fast wrt time. In this case, the stability of $A(t)$ is not guaranteed, even if the eigenvalues are in the left half-plane.

2nd scenario:

We are starting unicycle so far from the desired trajectory. Thus e is large and the approximation does not describe well the evolution of the system. Because of that, the system might not very well converge.

These scenarios come because of the choice of the control method we use. We expect these scenarios that can happen.

Final comments on this scheme (or method):

→ 1st: Actual inputs will be evaluated by using obtained gains. Notice that by saying actual inputs, we mean original inputs of unicycle, which are v and ω.

Computation of v:

→ Using (42) we get:

$$u_1 = -k_1 \cdot e_1 \quad (59)$$

→ Using (59) in (47):

$$v = v_d \cos e_3 - k_1 e_1 \quad (60)$$

where $k_1 = k_3 = k$, from (49).

Computation of ω:

→ Using (42) we get:

$$u_2 = -k_2 e_2 - k_3 e_3 \quad (61)$$

→ Using (61) in (48):

$$\omega = \omega_d + k_2 e_2 + k_3 e_3 \quad (62)$$

→ Using (49) and (58) in (62):

$$\omega = \omega_d + \frac{\omega_d^2 - a^2}{v_d} e_2 + k e_3 \quad (63)$$

Thus, (60) and (63) are expressions for actual inputs.

2nd: As expected, when e goes to zero, v and w should become v_d and w_d, respectively. Because vehicle converges the desired trajectory (i.e. the robot will be moving as exactly the target robot). Its input becomes the same as target's input.

In fact if we analyze the equations (60) and (63) we can see the exact result, by assuming error is zero. If $e = 0$, then $e_1 = e_2 = e_3 = 0$. Thus:

$$(60) \Rightarrow v = v_d \cos e_3 - k_1 e_1 = v_d \cos(0) = v_d \quad (64)$$

$$(63) \Rightarrow w = w_d + \frac{w_d^2 - a^2}{v_d} e_2 + k_3 e_3 = w_d \quad (65)$$

According to this information, we can say that when error goes to zero, these two inputs becomes feedforward inputs.

3rd: When v_d is zero, from (58) we see that k_2 goes to infinity. Thus, we can say that, this controller can't be applied for non-persistent trajectories. Because when v_d is zero, it means the cartesian motion has stopped. That is why, this controller can only be used with persistent cartesian trajectories (stops are not allowed).

4th: There is a nonlinear version of this controller which guarantees the global stability. In that case reconstructed inputs (u_1, u_2) are expressed as below with static feedback:

$$u_1 = -k_1(v_d, \omega_d)\, e_1 \quad (66)$$

$$u_2 = -k_2 v_d \frac{\sin e_3}{e_3} e_2 - k_3(v_d, \omega_d)\, e_3 \quad (67)$$

In this case, we see that, k_1, k_3 are the function of v_d and ω_d. Additionally in this case e_2 is related with u_2 by nonlinear term which is $v_d \frac{\sin e_3}{e_3}$.

As long as k_1 and k_3 are bounded, positive and have bounded derivatives, this controller will provide global stability of the error dynamics.

→ Block scheme:

The final block scheme for trajectory tracking via state error feedback and approximate linearization is demonstrated in the figure 5.

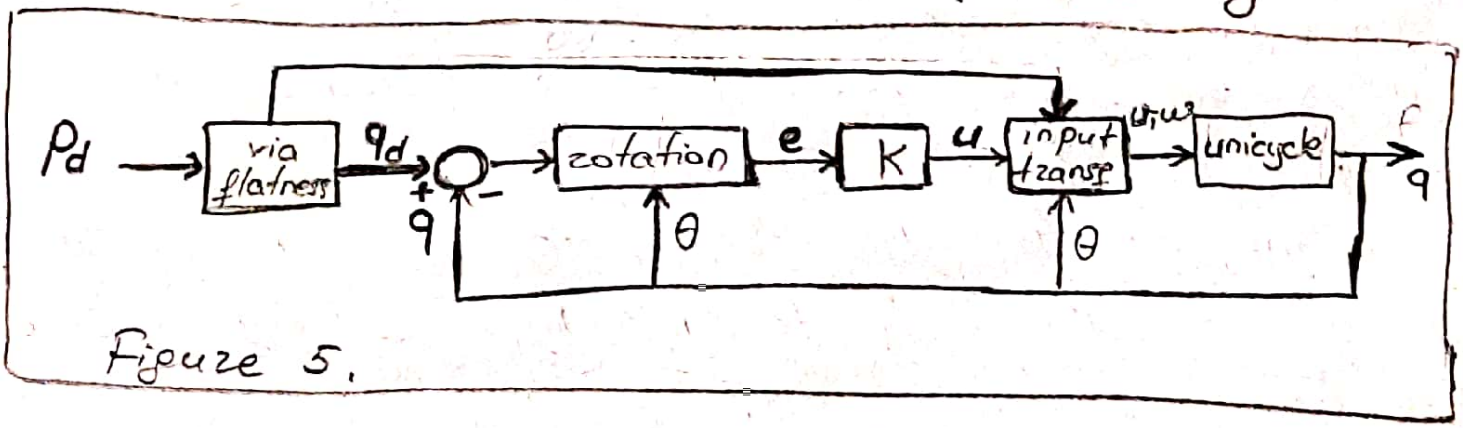

Figure 5.

- Output error feedback

This is another approach to trajectory tracking. We will develop the feedback action from the output (Cartesian) error only, without computing a desired state trajectory, while the feedforward term is the velocity along the reference trajectory.

We will design control laws with following steps:

→ For the unicycle, the map between the velocity inputs and Cartesian outputs is defined as below:

→ decoupled matrix

$$\begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} = \begin{pmatrix} \cos\theta & 0 \\ \sin\theta & 0 \end{pmatrix} \begin{pmatrix} v \\ \omega \end{pmatrix} \quad (68)$$

In (68), velocity inputs are v and ω, and Cartesian outputs are $\dot{x}$ and $\dot{y}$. It's seen that there is no relation between Cartesian output and ω. On the other hand, decoupled matrix is singular. Because it is singular, input-output linearization is not possible in this case.

→ The solution for this singularity is to change the output slightly so that the new input-output map is invertible and exact linearization becomes possible.

In this case will consider the point B in order to control unicycle, instead of the contact point (because, contact point causes singularity in decoupled matrix). Then the cartesian position of B (y_1 for x, y_2 for y) can be given as below:

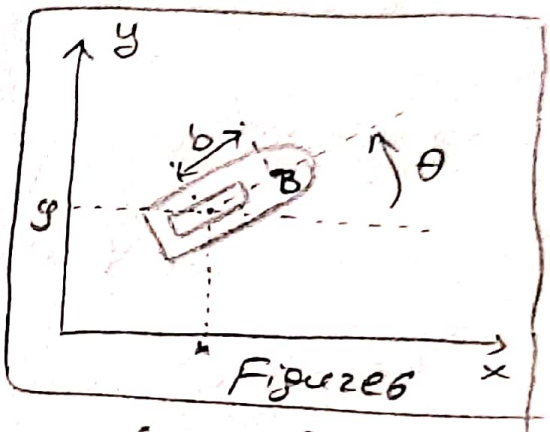

Figure6

$$y_1 = x + b\cos\theta \quad (69)$$
$$y_2 = y + b\sin\theta \quad (70)$$

x and y are cartesian coordinates of contact point, as demonstrated in figure 6. If we find the derivatives of y_1 and y_2 wrt time:

$$\dot{y}_1 = \dot{x} - b\sin\theta\,\dot{\theta} \quad (71)$$
$$\dot{y}_2 = \dot{y} + b\cos\theta\,\dot{\theta} \quad (72)$$

Using (68) in (71) and (72), and using $\dot{\theta} = \omega$ we can generate the following expressions:

$$\dot{y}_1 = v\cos\theta - b\sin\theta\,\omega \quad (73)$$
$$\dot{y}_2 = v\sin\theta + b\cos\theta\,\omega \quad (74)$$

$\dot{y}_1$ and $\dot{y}_2$ are cartesian outputs and, v and ω are velocity inputs. Thus we can generate the following matrix, in order to represent Input-Output map:

$$\begin{pmatrix}\dot{y}_1\\ \dot{y}_2\end{pmatrix} = \begin{pmatrix}\cos\theta & -b\sin\theta\\ \sin\theta & b\cos\theta\end{pmatrix}\begin{pmatrix}v\\ \omega\end{pmatrix} = T(\theta)\begin{pmatrix}v\\ \omega\end{pmatrix} \quad (75)$$

Note: B has to be chosen along the sagittal axis for simplicity.

→ In next step we will choose u_1 and u_2 as new inputs, which are $\dot{y}_1$ and $\dot{y}_2$, in such way that will provide us velocity inputs. In this case, we should know that $b \neq 0$. The following expression can be generated:

$$\begin{pmatrix} \varphi \\ \omega \end{pmatrix} = T^{-1}(\theta)\, u = \begin{pmatrix} \cos\theta & \sin\theta \\ -\frac{\sin\theta}{b} & \frac{\cos\theta}{b} \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} \quad (76)$$

From (76), we obtain:

$$u_1 = \dot{y}_1 \quad (77)$$

$$u_2 = \dot{y}_2 \quad (78)$$

$$\dot{\theta} = \frac{u_2 \cos\theta - u_1 \sin\theta}{b} \quad (79)$$

(79) is the evolution of θ which is nonlinear.

By computing (77) and (78) we stabilized of simple integrator.

→ At this step we will compute feedforward and feedback to design the control law! As feedforward we will take desired cartesian trajectory and cartesian error will be considered as feedback. Then u_1 and u_2 can be derived as below:

$$u_1 = \dot{y}_{1,d} + k_1 (y_{1,d} - y_1) \quad (80)$$

$$u_2 = \dot{y}_{2,d} + k_2 (y_{2,d} - y_2) \quad (81)$$

In (81) and (82), k_1 and k_2 must be positive. The main idea of this method is given in Appendix 9.2.

Notice that θ is not controlled by this scheme. It means it will evolve as what was computed by the equation (73). This method is based on output error feedback.

On the other hand, every control depends only on the corresponding error ($u_1 \to e_1$, $u_2 \to e_2$) It is like that, because we have integrators in parallel. There is no mixing of terms. That is why, it is called "decoupling" matrix. Every component of the output depends only on one component of the input.

Notice that, the desired trajectory for B can be arbitrary; in particular, square corners may be included. There are constraints on cartesian coordinates of contact point, but there is not any constraint on B.

Final block scheme for this controller is demonstrated by Figure 7.

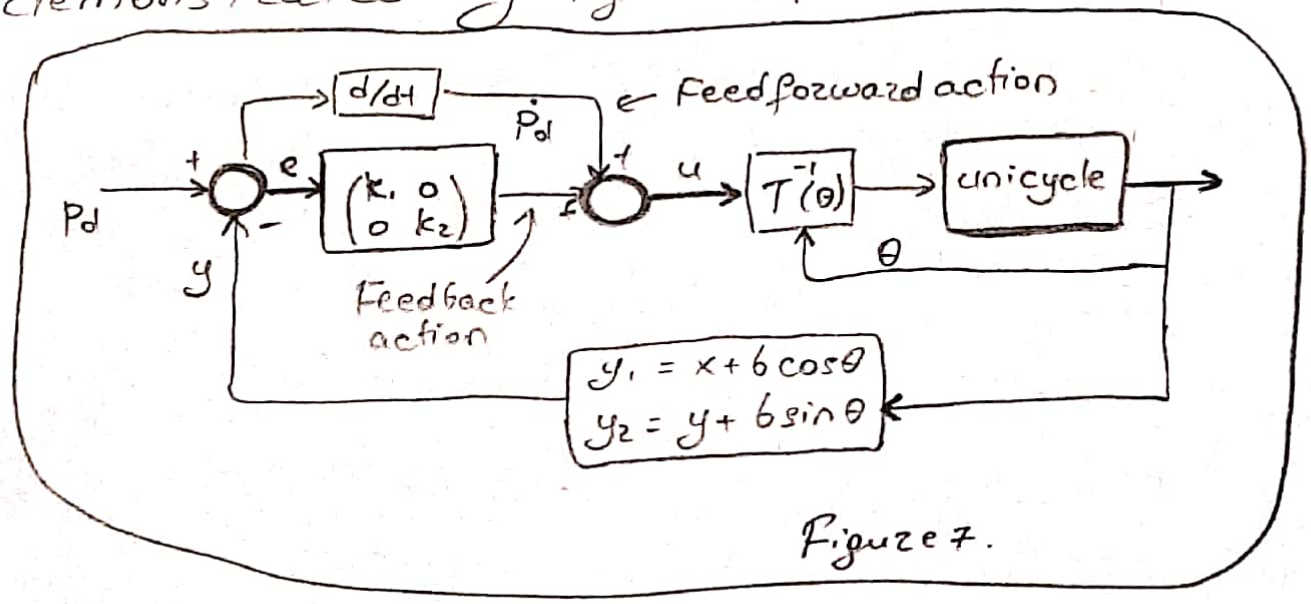

Figure 7.