Assignment 1 – ECE1658 Geometric Nonlinear Control of Robotic Systems

Modelling, Simulation, and Basic Control Design for a Five-Link Biped Robot

Main concepts covered in this assignment

- Lagrangian modelling of robots
- Modelling of impulsive impacts
- Numerical simulation of systems with impacts
- Relative degree and control design to enforce VHCs

1 Introduction

This assignment is a simulation laboratory that will allow you to gain a concrete understanding of the modelling of biped robots. The robot you will work with is the five-link biped depicted in Figure 1 which has the same structure of the robot Rabbit used by researchers at the University of Michigan and in Grenoble, France to test the first VHC-based control strategies. In this lab there is a minor conceptual component and mostly coding. You will be guided through the detailed steps of the code development. The last section of the document details the submission guidelines.

Consider the five-link biped robot in Figure 1. We make the following assumptions:

- The links are point masses m_i with massless rods, and these point-masses are positioned in the manner illustrated in Figure 1. The total kinetic energy is given by $K = 1/2 \sum_{i=1}^{6} m_i \|\dot{r}_i\|^2$, where $\dot{r}_i$ is the inertial velocity of mass m_i.
- The two legs are identical: they are composed of links with identical length and mass, and therefore $m_1 = m_5$, $m_2 = m_4$.
- The two feet are geometric points with masses m_1 and m_5.
- The angles $q_1, \ldots, q_5$ are measured counterclockwise, and they are defined in Figure 1.

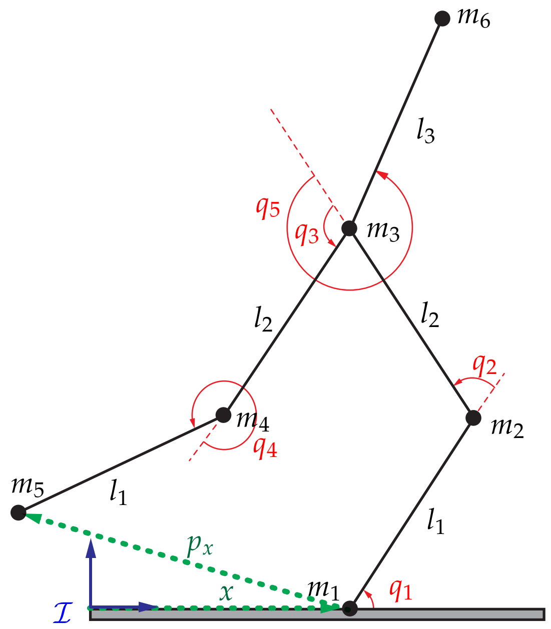

Figure 1: Five-link biped robot.

- The ground is perfectly flat, and the gravitational acceleration points downward.
- There's no friction.
- There are four actuators: two of them for the knees; one of them for the relative angle between the stance leg and the torso, and one for the relative angle between the legs.

Table 1: Biped parameters

$m_1 = m_5$	0.05 kg
$m_2 = m_4$	0.5 Kg
m_3	0.3 kg
m_6	0.5 Kg
l_1	0.5 m
l_2	0.5 m
l_3	0.3 m

In what follows, you will be guided through steps leading to the writing of Matlab code. Create a script `biped_main.m`.

2 Symbolic computations

In this section you will set up all the symbolic quantities that make up the robot model. Before we proceed, a warning. Symbolic computations with a five-link robot can easily give rise to long execution times. If you follow the steps below closely, you will end up with an efficient code that runs in seconds. Avoid using the Matlab command `simplify` as it can significantly slow down execution.

1. Define *numerical* variables `l1,l2,l3,m1,m2,m3,m4,m5` containing the robot parameters in Table 1, and a variable g equal to 9.81.

2. Define *real symbolic* variables `t,q1,q2,q3,q4,q5,x1,x2,q1dot,q2dot,q3dot,q4dot,q5dot,x1dot`. Define vectors `q,qdot,tau` containing the variables `qi,qidot,taui`, respectively. Define vectors `qbar,qbardot` containing the state of the unpinned robot, $\bar{q} := [q; x]$ and $\dot{\bar{q}} := [\dot{q}; \dot{x}]$, respectively.

3. We next find the total kinetic energy of the unpinned robot (recall that for the impact map, we need the matrix $\bar{D}$ stemming from this kinetic energy).

   (a) Find expressions for the positions of the six masses, r_i, $i \in \{1, \ldots, 6\}$, with respect to the inertial frame $\mathcal{I}$ in Figure 1. Your expressions should be in terms of $\bar{q}$, and determined assuming that the stance foot is at $x \in \mathbb{R}^2$, the vector shown in Figure 1.

   (b) In your script, define variables `r1,...,r6` containing the symbolic expressions you've just found. Then define variables `r1dot,...,r6dot` obtained by symbolically differentiating `ri` with respect to time through the chain rule. For instance, `r1dot=jacobian(r1,qbar)*qbardot`.

   (c) Now define a variable `K` containing the total kinetic energy of the unpinned robot. This will be the sum of six terms $(1/2)m_i\dot{r}_i^\top \dot{r}_i$.

   (d) Using the command `hessian`, extract the matrix of the kinetic energy, `Dbar`. This will be a 7×7 matrix.

   (e) The matrix $D(q)$ of the kinetic energy of the *pinned* robot is simply the submatrix of `Dbar` formed by the first five rows and columns. Define a variable `D` containing this submatrix.

4. Write the total potential energy of the pinned robot in terms of q and create a variable `P` containing it. Create a column vector `G` containing the symbolic gradient (i.e., the transposed Jacobian) of `P`,

5. Define the input matrix `B` of the robot, of dimension 5×4. Note that the first configuration variable, q_1 in unactuated, while $q_2, \ldots, q_5$ are directly actuated.

6. Using symbolic differentiation and the formula given in class for the Christoffel coefficients of `D` (note: `D`, not `Dbar`), find the Coriolis matrix $C(q, \dot{q})$ and name it `C`.

7. The impact map Δ requires $\bar{D}(q)$, which you've already computed, and the matrix-valued function

$$E(q) = \begin{bmatrix} (dp_x)_q & I_2 \end{bmatrix},$$

   where $(dp_x)_q$ is the Jacobian of $p_x(q)$, the vector with head at the swing foot and tail at x (see Figure 1). Define symbolic variables `px` and `E` containing $p_x(q)$ and the expression above for $E(q)$.

8. Turn the symbolic expressions in `D,Dbar,px,E,C,G` into Matlab functions with appropriate inputs. For instance, `Dfun=matlabFunction(D,'Vars',[q1;q2;q3;q4;q5])`. Note that `D,Dbar,E`, and `G` will functions of q, while `C` will be a function of $[q; \dot{q}]$. If you need, say, to numerically evaluate D at a numerical value of the vector q, you can now do it using the command `Dfun(q)`.

9. Create a structure array named `data` containing these objects: `Dfun,Dbarfun,Efun,Cfun,Gfun,B`. For instance, the command `data.D=Dfun` will create a field `D` in the structure containing the function `Dfun`. Later on, you will pass this structure array to the `ode45` function for numerical integration, and to the `impact_map` function for computation of the impact map.

3 ODE function and control design

Now that you have defined all quantities pertaining to the mathematical model of the biped robot, it's time to design a basic controller and set up the ODE function implementing the closed-loop system model.

We begin with the control specifications. Since we do not yet have the tools to design controllers for stable walking, we'll simply keep various relative angles constant. Specifically, since the four control inputs directly actuate the joint angles $q_2, \dots, q_5$, we'll regulate these four angles to desired constants $q_2^{\text{ref}}, \dots, q_5^{\text{ref}}$, and we'll do that defining an output function and using input-output linearization.

Consider the output function[1]

$$e = h(q) = \begin{bmatrix} q_2 - q_2^{\text{ref}} \\ q_3 - q_3^{\text{ref}} \\ q_4 - q_4^{\text{ref}} \\ q_5 - q_5^{\text{ref}} \end{bmatrix} = Hq - q^{\text{ref}}. \tag{1}$$

with $H =: \begin{bmatrix} 0_{4\times 1} & I_4 \end{bmatrix}$.

We want to check that the robot with this output has a well-defined vector relative degree $\{2,2,2,2\}$ on $\{(q,\dot q) \in T\mathcal{Q} : h(q) = 0\}$, or in other words the set $\mathcal{C} = h^{-1}(0)$ is a VHC. Note that $\mathcal{C}$ is a closed curve.

We could check relative degree by verifying that the 4×4 matrix $dh_q D^{-1}(q) B$ is invertible at each $q \in \mathcal{C}$, but doing so symbolically in Matlab is prohibitively time consuming. There is an equivalent test, presented in class, that is much more computationally efficient because it avoids the inversion of D, and that is to check that the real-valued function

$$B^{\perp} D(\sigma(\theta))\sigma'(\theta)$$

is nonzero for all θ. In the above, $\sigma : [\mathbb{R}]_{2\pi} \to \mathcal{Q}$ is the parametrization of the curve $\mathcal{C}$ given by

$$\sigma(\theta) = \begin{bmatrix} \theta \\ q_2^{\text{ref}} \\ \vdots \\ q_5^{\text{ref}} \end{bmatrix},$$

[1]Wait! This output function is problematic because its codomain is $[\mathbb{R}]_{2\pi}^4$, and instead we need the codomain to be $\mathbb{R}^4$. We could modify the output and use $\sin(q_i - q_i^{\text{ref}})$ as its elements. This would have the effect of making $q_i(t)$ converge to either $[q_i^{\text{ref}}]_{2\pi}$ or $[q_i^{\text{ref}} + \pi]_{2\pi}$, depending on the initial conditions. This problem is actually unavoidable when we deal with angular variables, unless we use discontinuous control which is of questionable practical value. In this assignment, we'll keep the output as stated, thinking of its codomain as being $\mathbb{R}^4$, but modify the feedback linearizing controller to account for the fact that $q_i \in [\mathbb{R}]_{2\pi}$.

and $B^{\perp}$ is a rank 1 left-annihilator of B, i.e., $B^{\perp} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \end{bmatrix}$.

Once we have established (and you will) that $\mathcal{C}$ is indeed a VHC, we proceed to defining an input-output feedback linearizing controller[2] that sends $e(t)$ to zero. In the expression below, we use the fact that for the function in (1), we have $dh_q = H$ and the Hessians of the components of h are all zero:

$$\tau(q,\dot{q}) = \left(HD^{-1}(q)B\right)^{-1}\left(HD^{-1}(q)(C(q,\dot{q})\dot{q} + \nabla_q P) - K_p \sin(Hq - q^{\text{ref}}) - K_d H\dot{q}\right). \tag{2}$$

Now you'll code the controller above.

1. In the main script `biped_main.m`, define numerical variables

   `q2ref=pi/6; q3ref=pi+pi/6; q4ref=-pi/6; q5ref=-pi/6;`

   and place them in a column vector `qref`. Define two control gains `Kp,Kd` for the controller (2) placing the roots of the polynomial $\lambda^2 + K_d\lambda + K_p$ at $\{-20, -20\}$. Create the 4×5 matrix `H` in (1).

   Add the variables `qref,Kp,Kd,H` to the structure array `data` that you've created earlier.

2. Compute the quantity $B^{\perp}D(\sigma(\theta))\sigma'(\theta)$ (you'll find that it is actually a constant, independent of θ), and verify that it is not zero.

3. Create an ODE function biped.m accepting the structure array `data`. The function declaration should be this: `function xdot=biped(t,x,data)`. Here, `x` is the robot state, $[q;\dot{q}]$ and `xdot` is its time derivative. The function therefore implements the closed-loop vector field.

   Within this function, extract from `x` the subvectors `q` and `qdot`, and extract from the structure `data` the variables you need for the ODE. You need to compute τ as in (2) and compute $\ddot{q} = D^{-1}(q)(-C(q,\dot{q})\dot{q} - \nabla_q P(q) + B\tau)$. Once you've done that, you'll set `xdot = [qdot;qddot]`.

4 The impact map

The impact map Δ is the composition of two maps, $\Delta = \Delta_2 \circ \Delta_1$. The map Δ_1 represents the effects of an impulsive reaction force, while the map Δ_2 represents the transition to a new state model in which the two legs have swapped identity. For our choice of x, the map Δ_1 is given by

$$\Delta_1([q;\dot{q}]) = \left[q; \Delta_{\dot{q}}(q)\dot{q}\right],$$

where

$$\Delta_{\dot{q}}(q) =: \begin{bmatrix} I_5 & 0_{5\times 4} \end{bmatrix} \begin{bmatrix} \bar{D}(q) & -E(q)^{\top} \\ E(q) & 0_{2\times 2} \end{bmatrix}^{-1} \begin{bmatrix} \bar{D}(q) \begin{bmatrix} I_5 \\ 0_{2\times n} \end{bmatrix} \\ 0_{2\times 5} \end{bmatrix}.$$

On the other hand, the map Δ_2 is given by

$$\Delta_2([q;qdot]) = \left[Rq + d; R\dot{q}\right],$$

[2]Note the proportional feedback term $K_p \sin(Hq - q^{\text{ref}})$, inserted to guarantee that the controller treats q_i are variables in $[\mathbb{R}]_{2\pi}$.

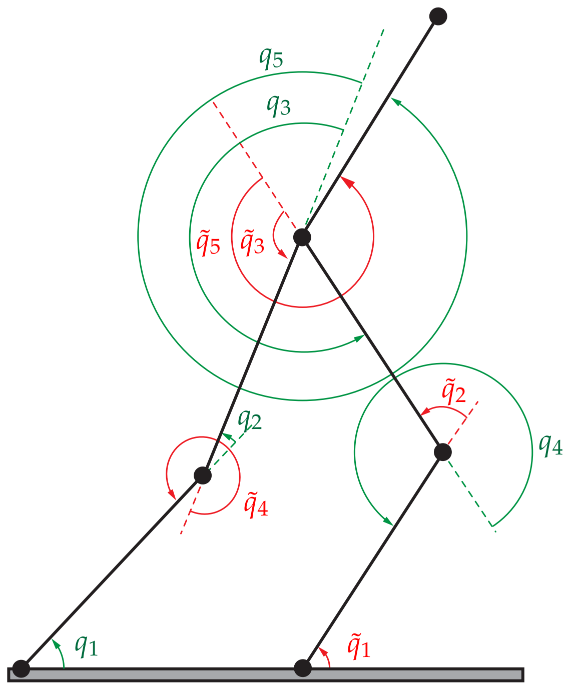

Figure 2: Relabelling of configuration variables, where q_i is the old variable and $\tilde{q}_i$ is the new one.

where $R \in \mathbb{R}^{5\times5}$ and $d \in \mathbb{R}^5$ are quantities you need to determine. The function $\tilde{q} = T(q) := Rq + d$ relabels the configuration variables as a consequence of the swapping of identity of the two legs. Using Figure 2 as a reference and basic geometry, you need to find the matrix R and the vector d. In deriving T, do not assume that the swing foot is on the ground, and measure $\tilde{q}_1$ relative to a horizontal line passing through the foot.

Carefully double-check the correctness of your derivations before proceeding.

Now you'll code the impact map.

1. Create a function called `impact_map`, whose declaration will be `function Delta=impact_map(x,data)`. Note that we are passing the structure array `data` to this function. In the call, the vector `x` is the robot state.
2. Extract from `x` the components `q` and `qdot`, then extract from `data` the variables you need for the impact map computation.
3. In the function, compute $\Delta([q; \dot{q}])$ as detailed above, and place the result in a 10×1 vector `Delta` which the function will return to the main script.

5 Numerical simulation

Now we return to your main script `biped_main.m`, and simulate the closed-loop system.

1. First off, you need to create an events function telling Matlab to stop integrating when the swing foot has reached the ground, that is, when the second component of the vector `px` crosses zero

from a positive to a negative value. Check out the Matlab help to see how events functions are defined, and call this function `ground_impact`.

2. Using the command `odeset`, set up integrations options in a structure named `ops`: the absolute and relative tolerances should be 10^{-8}, and the events function should be `@ground_impact`.
3. Define the initial condition `q0=[pi/3;pi/4;pi-pi/6;-pi/3;-pi/6]; qdot0=zeros(5,1);`
4. Now simulate the robot until first impact occurs using the call

   `[t1,x1,te1,xe1]=ode45(@(t,x) biped(t,x,data),0:5e-2:10,[q0;qdot0],ops).`

   Parse the syntax above. First, we tell Matlab that we are to integrate the ODE contained in `biped`, but we want to pass to it the structure array `data`. Matlab wants to see an ODE function of `(t,x)`, so we pass to it an anonymous function of `(t,x)` that is parametrized by `data`. Next, the time interval of integration is $[0, 10]$, and we want Matlab to return solution samples equally spaced at time intervals of $5 \cdot 10^{-2}$. This does not in any way correlate with the time step of numerical integration, since Matlab adapts that automatically. Next, the initial condition, and finally the integration options. The integrator `ode45` returns the time and state samples, `t1,x1`, and the time and state, `te1,xe1` when the impact event has occurred and integration has been terminated.
5. Next, take the impact state `xe1`, transpose it to turn it into a column vector, and using the function `impact_map(xe1',data)` compute a post-impact state. This will be the initial condition for the next integration.
6. Integrate a second time until next impact, saving time and state samples in vectors `t2,x2`. In principle you could have this integrations in a loop for multiple steps, but our robot is not ready to walk so we stop here.
7. Using the provided code snippet as a reference, create an animation of your results.

6 Submission guidelines

Please submit a zipped folder containing the following components:

- Your code, clearly organized and with abundant commenting. The code must run in seconds. The instructor's code, including the animation, runs in 10.7 seconds on a Macbook Pro with a 2 GHz Quad-Core Intel Core i5 processor and 16GB of RAM.
- A concise hand-written document in which you explain your derivation of the position vectors r_i, the potential function $P(q)$, and the relabelling function $T(q)$