Mathematic Modeling of Path Tracking with Adaptive Backstepping

1 Introduction

The equation of motion for the constrained robotic manipulator with n degrees of freedom is given in the joint space as follows:

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) = \tau \tag{1}$$

where:

- $\boldsymbol{q} \in \mathbb{R}^{n\times 1}$ denotes the joint angles or link displacements of the manipulator
- $M(\boldsymbol{q}) \in \mathbb{R}^{n\times n}$ is the robot inertia matrix which is symmetric and positive definite
- $C(\boldsymbol{q}, \dot{\boldsymbol{q}}) \in \mathbb{R}^{n\times n}$ contains the centripetal and Coriolis terms
- $G(\boldsymbol{q}) \in \mathbb{R}^{n\times 1}$ are the gravity terms
- $\boldsymbol{u} \in \mathbb{R}^{n\times 1}$ denotes the torque or the force

We want the manipulator to follow the track described by the the equation $C(\epsilon) = 0$, with $\epsilon = (x, y, z)^T$, position of the end effector (e.e.). For example a circular track in the (x, y) plane can be described as:

$$C(\epsilon) = (x - x_0)^2 + (y - y_o)^2 - R^2 \tag{2}$$

2 Backstepping Control

We are looking for a control strategy such that the e.e. follows the track with a desired velocity α. Let's define a Lyapunov candidate function:

$$V = \frac{1}{2} C(\epsilon)^T C(\epsilon)$$

Its derivative is:

$$\dot{V} = C(\epsilon)^T C_\epsilon J \dot{q} = C(\epsilon)^T C_\epsilon J u'$$

with J, the Jacobian and $C_\epsilon = {}^{\partial C(\epsilon)}\!/\!_{\partial \epsilon}$. We are also supposing that the joint velocity can be imposed (neglecting the dynamics).
The expression of the control which makes the derivative of the Lyapunov candidate negative defined is:

$$u' = \Gamma_{\dot{q}}(\epsilon) = -J^R C_\epsilon C(\epsilon)^T \nu + J^R S(\epsilon)\alpha$$

with ν and α scalar parameters and $S(\epsilon)$ vector tangent to $C(\epsilon) = 0$, such that $C_\epsilon^T S(\epsilon) = 0$. The imposed $S(\epsilon)$ does not deviate the e.e. from the track.
The derivative of the Lyapunov candidate will be:

$$\dot{V} = -C(\epsilon)^T C_\epsilon C_\epsilon^T C(\epsilon)\nu$$

which is negative definite.
This was the first part of the backstepping. Now we introduce the dynamics of the system.
Let's consider a new Lyapunov candidate function for the state $(\epsilon, \dot{q})$:

$$W(\epsilon, \dot{q}) = \frac{1}{2}C(\epsilon)^T C(\epsilon) + \frac{1}{2}(\dot{q} - \Gamma_{\dot{q}})^T(\dot{q} - \Gamma_{\dot{q}})$$

Now we evaluate the derivative:

$$\dot{W}(\epsilon, \dot{q}) = C(\epsilon)^T C_\epsilon J\dot{q} + (\ddot{q} - \dot{\Gamma}_{\dot{q}})^T(\dot{q} - \Gamma_{\dot{q}})$$

Let's note that we can not substitute $\dot{q}$ with u', since we are considering the model dynamics.
Now we try to obtain a derivative similar to $\dot{V}$.
We sum and subtract the term $-C(\epsilon)^T C_\epsilon C_\epsilon^T C(\epsilon)\nu$ and we obtain:

$$\dot{W}(\epsilon, \dot{q}) = -C(\epsilon)^T C_\epsilon C_\epsilon^T C(\epsilon)\nu + C(\epsilon)^T C_\epsilon(J\dot{q} + C_\epsilon^T C(\epsilon)\nu) + (\ddot{q} - \dot{\Gamma}_{\dot{q}})^T(\dot{q} - \Gamma_{\dot{q}})$$

We substitute the model equation of motion:

$$\ddot{q} = M(q)^{-1}(\tau - C(q, \dot{q})\dot{q} - G(q))$$

And we obtain:

$$\dot{W}(\epsilon, \dot{q}) = -C(\epsilon)^T C_\epsilon C_\epsilon^T C(\epsilon)\nu + C(\epsilon)^T C_\epsilon(J\dot{q} + C_\epsilon^T C(\epsilon)\nu) + (M(q)^{-1}(\tau - C(q, \dot{q})\dot{q} - G(q)) - \dot{\Gamma}_{\dot{q}})^T(\dot{q} - \Gamma_{\dot{q}})$$

So we can choose the following τ expression to make $\dot{W}$ negative definite:

$$\tau = C(q, \dot{q})\dot{q} + G(q) + M(q)(\dot{\Gamma}_{\dot{q}} - (\dot{q} - \Gamma_{\dot{q}})^{-1}(C(\epsilon)^T C_\epsilon(J\dot{q} + C_\epsilon^T C(\epsilon)\nu)) - K_b(\dot{q} - \Gamma_{\dot{q}})$$

In fact, we obtain:

$$\dot{W}(\epsilon, \dot{q}) = -C(\epsilon)^T C_\epsilon C_\epsilon^T C(\epsilon)\nu - K_b(\dot{q} - \Gamma_{\dot{q}})^T(\dot{q} - \Gamma_{\dot{q}}) \tag{3}$$

3 Implementation

The control technique was implemented in Python and tested with ROS and Gazebo.
We defined a class based on *urdf2casadi* which is able to load the motion matrices and the forward kinematics of the manipulator from an URDF file.
The track $C(\epsilon)$ is defined by the user (as in the following image) and the derivative is evaluated in Python using CasADi library. The control is then evaluated as in expression 3. The state value is repeatedly updated through a ROS subscriber.

```
def C_function(x, y):
    return (x - x0)**2 + (y - y0)**2 - R**2
```

It is important to notice that ϵ value is obtained from the forward kinematics and that $\dot{\Gamma}_{\dot{q}}$ is evaluated as:

$$\dot{\Gamma}_{\dot{q}} = \frac{\partial \Gamma}{\partial q}\frac{\partial q}{\partial t} = \frac{\partial \Gamma}{\partial q}J$$

Where $\partial\Gamma/\partial q$ is again obtain through CasADi symbolic derivation.

A video of the implemented control can be seen here, two snippets are pasted below. The control is implemented on a RRbot which is following a circular path.

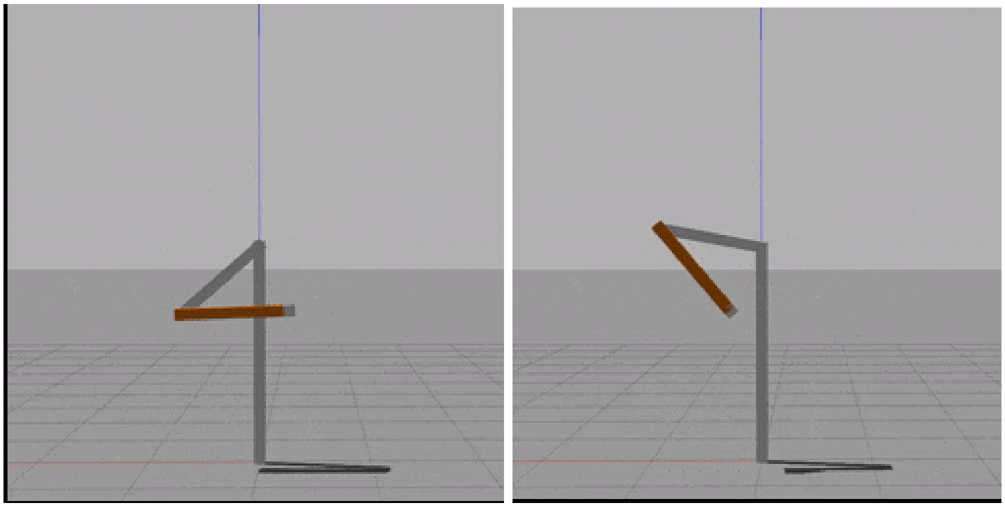

Figure 1: Two instants of the Rrbot following its path

4 Adaptive version

We tried to implement also an adaptive version of the algorithm. To reduce the complexity of the adaptive model, we consider just an unknown value of gravity (which representative of a condition of tilted arm). The gravity matrix $G(q)$ will be affected by this modification.
Since the gravity acceleration is a common factor in all the terms of the gravity matrix, the dynamics can be easily linearized as follows:

$$\tau = M(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) = M(q)\ddot{q} + C(q,\dot{q})\dot{q} + \bar{G}(q)g$$

So we can define the dynamics parameter $\pi = g$, the estimated gravity value $\hat{\pi} = \hat{g}$ and the difference between them $\tilde{\pi} = \hat{\pi} - \pi = \hat{g} - g$. Let's note that g is the true value of g, perceived in the plane of the motion of the robot (along its first arm).
Then, we introduce a control for the estimated gravity term, which will simply be:

$$\dot{\tilde{\pi}} = +\dot{\hat{\pi}} = +u_\pi$$

So, according to the estimated value of gravity, the imposed control will be:

$$\tau = C(q,\dot{q})\dot{q} + \hat{G}(q) + M(q)(\dot{\Gamma}_{\dot{q}} - (\dot{q} - \Gamma_{\dot{q}})^{-1}(C(\epsilon)^T C_\epsilon (J\dot{q} + C_\epsilon^T C(\epsilon)\nu)) - K_b(\dot{q} - \Gamma_{\dot{q}})$$

The new Lyapunov candidate should also consider the $\tilde{\pi}$ term:

$$W(\epsilon, \dot{q}, \hat{\pi}) = \frac{1}{2}C(\epsilon)^T C(\epsilon) + \frac{1}{2}(\dot{q} - \Gamma_{\dot{q}})^T(\dot{q} - \Gamma_{\dot{q}}) + \frac{1}{2}R\tilde{\pi}^T\tilde{\pi}$$

And the resulting derivative will be:

$$\dot{W}(\epsilon, \dot{q}, \tilde{\pi}) = -C(\epsilon)^T C_\epsilon C_\epsilon^T C(\epsilon)\nu - K_b(\dot{q} - \Gamma_{\dot{q}})^T(\dot{q} - \Gamma_{\dot{q}}) + (M(q)^{-1}(-G(q) + \hat{G}(q)))^T(\dot{q} - \Gamma_{\dot{q}}) + R\tilde{\pi}u_\pi$$

$$\dot{W}(\epsilon, \dot{q}, \tilde{\pi}) = -C(\epsilon)^T C_\epsilon C_\epsilon^T C(\epsilon)\nu - K_b(\dot{q} - \Gamma_{\dot{q}})^T(\dot{q} - \Gamma_{\dot{q}}) + (M(q)^{-1}\bar{G})^T\tilde{\pi}(\dot{q} - \Gamma_{\dot{q}}) + R\tilde{\pi}u_\pi$$

In order to obtain a definite negative derivative, we choose u_π:

$$u_\pi = -\frac{1}{R}(M(q)^{-1}\bar{G})^T(\dot{q} - \Gamma_{\dot{q}})$$

And we obtain the final derivative:

$$\dot{W}(\epsilon, \dot{q}, \tilde{\pi}) = -C(\epsilon)^T C_\epsilon C_\epsilon^T C(\epsilon)\nu - K_b(\dot{q} - \Gamma_{\dot{q}})^T(\dot{q} - \Gamma_{\dot{q}}) \tag{4}$$

Let's note that the $\tilde{\pi}$ term is not present. The function is just semi-negative definite and the acceleration estimate will go to the real value just if the track is persistently expiatory.

A video of the implemented control can be seen here. The control is implemented on a RRbot which is following a circular path.
In the model, the initial value of the estimated gravity is $\hat{\pi} = 9.8\frac{m}{s^2}$, the standard value for g. Now we are considering a different situation: the robot is tilted of $\frac{\pi}{4}$, as presented in the figure below. In this way, the value of the gravitational acceleration which the motors have to contrast is $g = 9.8 \times sin(\frac{\pi}{4})\frac{m}{s^2} = 6.9\frac{m}{s^2}$. This can me obtained projecting the gravity vector along a plane perpendicular to rotoidal joints.

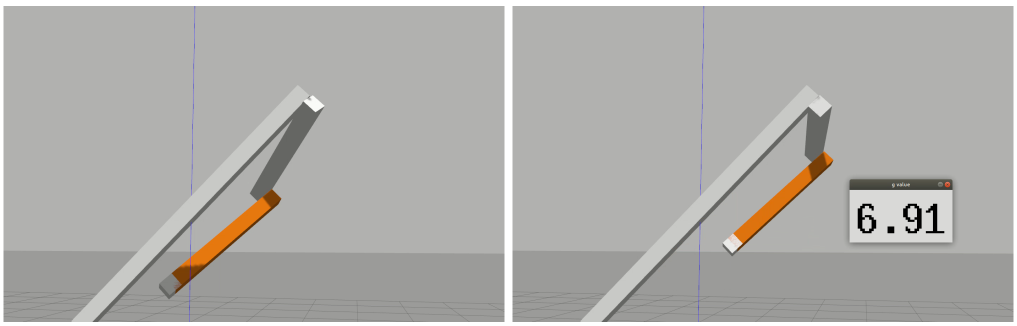

Figure 2: Two instants of the RRbot following its path. Now the robot is tilted of $\frac{\pi}{4}$ at its base.

As we can see in the video and in the graph below, the gravity value evolves from the standard value to the attended one. In blue, we represent the value of estimated gravity $\hat{g}$, in red the expected value of $6.9\frac{m}{s^2}$. After a transient phase, the parameter oscillates around the expected vale. This error is due to the friction forces which are not considered in the model (and which can be intensified in some configurations by the tilted position). Performing a time average, we can correctly evaluate the actual gravity value and also the tilt angle.

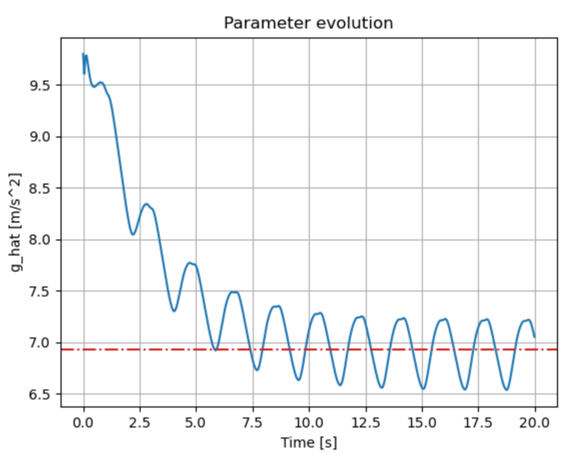

Figure 3: Two instants of the RRbot following its path

It is interesting to see that the control is robust to wrong gravity values. In fact, the robot follows a path similar to the desired one, also while gravity value is significantly different from the one perceived in the simulation.
In addition, since the gravity value is continuously updated, we can conclude that this path is persistently exciting.