TU Berlin Robotics

Lab Assignment #1

Please check the deadline for the solution of this sheet on the ISIS page. Remember that this is a hard deadline; extensions are impossible!

Preliminaries

We encourage you to get the book "Introduction to Robotics" by John J. Craig (from the library or your favorite book store...)! It will be a big help throughout the semester, especially for the first three assignments.
Please follow these **guidelines** (we may deduct points, otherwise):

- Do not change method arguments and predefined names.
- Write all angles in radians, not in degrees, unless stated explicitly.
- Approximate values to 2 decimal places (e.g. 12.25).
- Simplify your terms, i.e. use trigonometric identities [1]
- In the documentation, use the following abbreviations for trigonometric terms: $s_i = \sin(q_i)$, $c_i = \cos(q_i)$, $s_{ij} = \sin(q_i + q_j)$, $c_{ij} = \cos(q_i + q_j)$, $s_{123} = \sin(q_1 + q_2 + q_3)$, $c_{123} = \cos(q_1 + q_2 + q_3)$. Declare any additional abbreviations clearly at the beginning of your assignment!
- Follow the coding guidelines[2]. Note in particular the requirement to **use only C++98 compatible code features** and to **remove all print-outs** before submitting your solution!

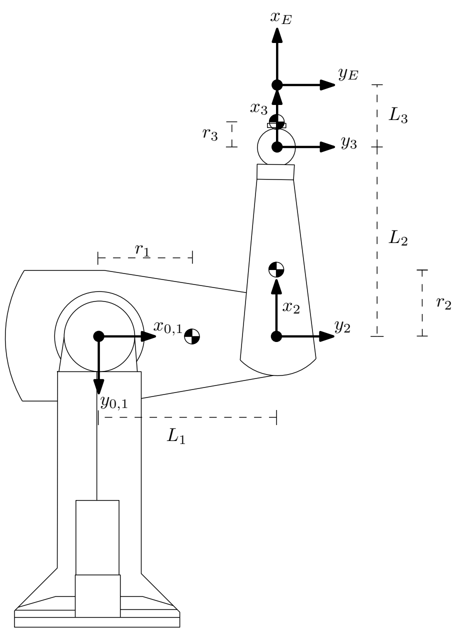

Figure 1: Puma RRR planar manipulator in zero configuration.

[1] `https://en.wikipedia.org/wiki/List_of_trigonometric_identities`
[2] See "Notes and Restrictions on Coding.pdf" in Chapter 1 on ISIS course

A Calculations (20 points)

In this assignment, you will derive the gravity vector of the RRR planar manipulator shown in Fig. 1, which corresponds to the 3-DOF mode of the Puma 560. The three joints in this mode correspond to the joints 2, 3, and 5 of the robot. Each link i has a mass m_i. Each center of mass (COM) is located at distance r_i in X-direction of frame i. The i-th joint angle is called q_i.

1. **[20 Points] Compute the gravity vector $\mathbf{G(q_1, q_2, q_3)}$= $[?\ ?\ ?]^{\mathbf{T}}$** which estimates the torque ($\frac{kg \cdot m^2}{s^2}$) caused by gravity at each joint.

B Implementations (80 points)

Note:

- In this assignment, the robot should operate in 3-DOF mode. By default, it starts in "6-DOF (quaternions)". You will have to select "3-DOF" from the drop-down menu in the simulator.
- In the 3-DOF mode you can only access the joints 2, 3 and 5 as `gv.q[0]`, `gv.q[1]` and `gv.q[2]` respectively. You can check the currently active DOF mode with `gv.dof`.

1. **[12 Points] njmoveControl()**
   Implement a P-controller for the joints 2, 3 and 5 of the Puma in `njmoveControl()`.

2. **[10 Points] Tune the controllers**
   Execute a 10° step in all three joint angles and manually adjust the proportional gains (`kp`) for the joints. Find `kp` values such that:

   - q_{des} is reached as fast as possible
   - there is no oscillation/overshoot in any of the values of q,
   - the robot's torque limits are not exceeded

   Questions:

   (a) What kind of behaviors do you observe with different gains?
   Please list in the pdf the tuned gains.

   (b) Why are well tuned gains different for each joint?

   Note: Do not hard code torque limiting, but instead tune the controller gains in a way that the torque limits are not exceeded. You will have to disable automatic torque limiting by the simulator (Settings tab→uncheck "torque limits") and compare the torque values you record with the Puma torque limits (see the table at the end of the Assignment sheet).

   To test your controller select the `njmove` control mode in the first drop down box. You can then specify the desired joint configuration q_{des} in the fields next to it. Click "Start" to activate the controller.

   Hint: You will not be able to achieve all three goals perfectly.

3. **[10 Points] Document Behavior**
   Now document the robot's behaviour for a step from $q_{init} = [0°\ 0°\ 0°]^T$ to $q_{des} = [10°\ 10°\ 10°]^T$. Plot the desired angles qd, the actual joint angles q, and the applied torques τ.

   You can use the plotting feature of the simulator or any other plotting tool you like: Octave/Gnuplot, SciPy, Matlab, etc. For a well tuned controller, the resulting graph for q should look similar to Fig. 2. For all plots, list the gains you used in the caption/title.

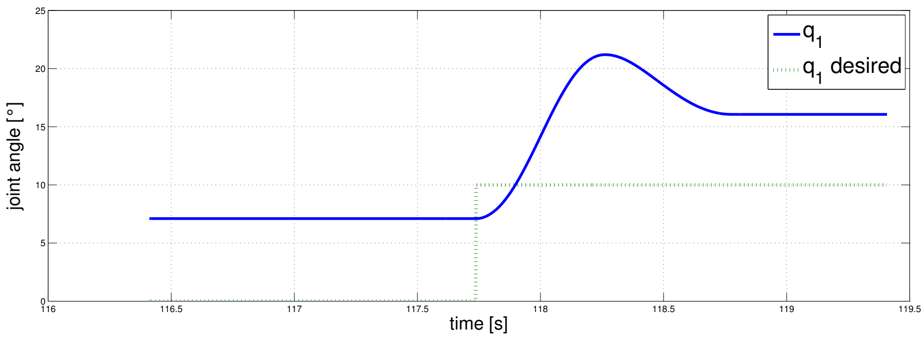

Figure 2: A well-tuned P-controller step response

Note: To record data in the simulator, set the simulator to 3-DOF and disable torque limiting (Settings tab→uncheck "torque limits"). Select the *nj-move* control mode in the first drop down box, set the angles to $(0, 0, 0)$, and click on "Start". TODO: plot the best gain version To begin the recording of data enter a filename, e.g. "data.mat", and click on "Start recording". Then select njmove control mode in the second drop down box, set the angles to $(10°, 10°, 10°)$, and click on "Start". Click on "Stop recording" once the robot has reached its goal. The resulting data file has the format: `time q(1..n) dq(1..n) qd(1..n) tau(1..n) x(1..m) dx(1..m) xd(1..m)`.

4. **[12 Points] Calculate the gravity vector in `PreprocessControl()`**

   Use the function you derived in section A "Calculations" to compute the effect of gravity on the links with the joint torques. Save the torques into the vector `gv.G`.

   The parameters for the actual geometry of the Puma robot are defined in param.h. Use these link parameters for your 3-DOF implementation:

   - $r1 = R2, \quad r2 = 0.189738, \quad r3 = R6$
   - $l1 = L2, \quad l2 = L3, \quad l3 = L6,$
   - $m1 = M2, \quad m2 = M3 + M4 + M5, \quad m3 = M6$
   - $g = -9.81$

5. **[12 Points] floatControl()**

   Use the gravity vector function you computed to compensate the effect of gravity on the links in `floatControl()`. In *float* mode, the robot should keep its current pose and only move when an external force is applied.

   To test your controller in gravity compensation, set the simulator to 3-DOF and start the `float` control mode. "Pull" on the robot in the visualization by dragging on the links with the mouse. It should move while you are pulling and stop when you let go.

   Note: Your implementation **must** also work with the 6-DOF Puma so that it can be tested on the real Puma (test this in the simulator!).

6. **[12 Points] njgotoControl()**

   Implement a P-controller with gravity compensation in `njgotoControl()`. Tune the position gain and plot the response to a 10° step in desired angles (q, qd, and τ). Remember to report the gains you chose.

7. **[12 Points] jgotoControl()**

Implement a PD-controller with gravity compensation in `jgotoControl()`. Now your controller should be able to make the 10° step without oscillation or overshoot.

Choose the `kp` and `kv` as high as possible (with respect to the torque limits). Plot the response to a 10° step in desired angles (q, qd, and τ). Why can the gains `kp` now be higher compared to the P-controller?

Deliverables

- The `control.cpp` file: containing the control code (from `controlDLL/`)
  - Do not add, modify or upload any other source code files
  - All handed in source code is checked for plagiarism!
- The `gains_1.txt` file: containing the **controller gains** for 3-DOF mode (from the directory where the simulator was called from, usually `build/`)
  - The file will only be created when you click on "Store gains" in the GUI
  - For each controller, the gains are stored in separate lines of a single file: `gains_1.txt`. Make sure to save gains for each of the controllers in this assignment. (`njmoveControl(), njgotoControl(),` and `jgotoControl()`)
  - **Important:** You must **not** hard-code the gains inside `control.cpp`; always use the appropriate global variables (`gv.kp` etc.) such that the gains can be stored in the file `gains_1.txt`.
- **One** PDF file containing:
  - the solutions for the **Calculations** part
  - text answers of the **Implementation** part
  - **Figures** illustrating the Controller behavior. All plots must have a legend and all axes must be labeled!
  - the tuned controller **gains**.
  - A **table** listing for all **implementation tasks** which team member(s) implemented them (see explanation and template below).

Explanation and template for implementation table

- Every group member needs to be able to **answer the "high level" questions about *ALL* tasks** of the assignment.
  We will check that during the presentations with general questions. Everyone needs to be able to answer these.
- For **implementation**, please **specify which team member** worked on which (sub-)task.
  - We will check that during the presentations with implementation questions.
  - You can split up implementation of (sub-)tasks. But overall, every one needs to contribute equally to the implementation. (Writing the report does **NOT** count as "contributing to the implementation".)
- Please use the following template in your submission:

Student Name	B1	B4	B5	B6	B7
Albert Albono	x			x	x
Betty Barlow	x	x	x	x	

Note

- Do not limit the torques in your control code and disable the torque limiting in the simulator (Settings tab → "torque limits").

- The maximum torque produced during the 10° step depends on the gains. Your controller should not produce torques higher then the maximum torque for the joints:

Joint:	1	2	3	4	5	6
τ_{max}:	$97,6Nm$	$156,4Nm$	$89,4Nm$	$24,2Nm$	$20,1Nm$	$21,2Nm$