
(Spring 2010-11)

Lab Assignment #5

Due Lab #5

In this assignment you will improve the dynamic operational-space controllers to handle singularities. Though this handout describes only one possible approach, you may treat it as an open-ended assignment. For a basic test of your code, make sure you do not exceed torque limits, when "torque limit" is not checked in the Settings tab.

Overview

Usually, the PUMA's end-effector can move in all 6 directions: $(\dot{x}, \dot{y}, \dot{z}, \omega_x, \omega_y, \omega_z)$.

But in a singularity, the PUMA develops a singular direction — a direction in which it cannot move in operational space. This effectively reduces the PUMA from a 6-DOF robot to an n' -DOF robot, where n' equals 6 minus the number of singularities. The symbol n' will be used throughout this handout.

For every singular direction that becomes inaccessible in operational space, the PUMA gains a nontrivial solution to the equation $J\dot{q} = 0$ in joint space. Each of these solutions corresponds to a motion in joint space that causes no motion in operational space. This is the *null space* of the robot.

In this lab, we will handle singularities by decomposing the robot motion into two components:

- Motion orthogonal to the singular directions. This motion will be treated just like ordinary operational-space motion, except that there will be only n' degrees of freedom.
- Motion in the null space, which moves the PUMA within the singularity in a way to help it reach the goal. The desired null-space motion for each singularity is discussed in the next section.

Singularities

1. Write a function `void GetNonsingularSelection(PrMatrix& selectionMatrix)`, to create an $n \times 6$ selection matrix S to filter out the singular directions. See **Creating the selection matrix** below for tips.

The global variable `gv.singularities` indicates which singularities we are in:

- `(gv.singularities&HEAD_LOCK)` is set in head-lock.
- `(gv.singularities&ELBOW_LOCK)` is set in elbow-lock.
- `(gv.singularities&WRIST_LOCK)` is set in a wrist-lock.

2. Write a function `void GetEscapeTorque(const PrVector& fPrime, PrVector& escapeTorque)`, to calculate a 6×1 torque Γ_0 to escape from the current singularity(s). See **Finding a joint torque to escape the singularity** below for tips.
3. Rewrite `void OpDynamics(const PrVector& force)` to handle singularities better. Outside of a singularity, `OpDynamics()` should behave the same as before. Within a singularity, it should run the above two functions, and set the torque as follows, where f' is the argument of `OpDynamics()`, S is the selection matrix, Γ_0 is the escape torque, and I is the identity:

$$J_r = S * J \quad (1)$$

$$\Lambda_r = (J_r A^{-1} J_r^T)^{-1} \quad (2)$$

$$\tau = J_r^T \Lambda_r S f' + (I - J_r^T \Lambda_r J_r A^{-1}) \Gamma_0 + G \quad (3)$$

Make sure you put enough comments so that we can understand what you're doing!

Submit a report describing what you learned in the lab. Discuss special features in your code. Incorporate material from lectures.

Operational Space Framework in Brief

To aid your intuition on where the above formulas come from, let us we concisely go through more formulas. The motion of the end-effector can be generated by torque:

$$\Gamma_i = J_i^T F_i + N_i^T \Gamma_{i+1} \quad (4)$$

where J_i is the Jacobian matrix defined for a certain task or objective, F_i is the corresponding desired task force, N_i is the null space projection matrix of the task, and Γ_{i+1} is a recursive definition for the torque of lower priority tasks. In more detail:

$$N_i = (I - \bar{J}_i J_i) \quad (5)$$

$$\bar{J}_i = A^{-1} J_i^T \Lambda_i \quad (6)$$

$$\Lambda_i = (J_i A^{-1} J_i^T)^{-1} \quad (7)$$

$$F_i = \Lambda_i \ddot{x}_i + \mu(x, \dot{x}) + \hat{p}(x) \quad (8)$$

where Λ_i is the pseudo kinetic energy matrix, $\mu(x, \dot{x})$ is the Coriolis and Centrifugal matrix, and $\hat{p}(x)$ is the gravity compensation matrix in the task space. The μ term is omitted in the torque formula derived in the previous section.

Singularities in the PUMA

Singularities can be determined analytically by solving $\det(J) = 0$ or $\det(JJ^T) = 0$ for manipulators with non-square Jacobians. Defining the control point to be at the wrist, the PUMA has three types of singularities:

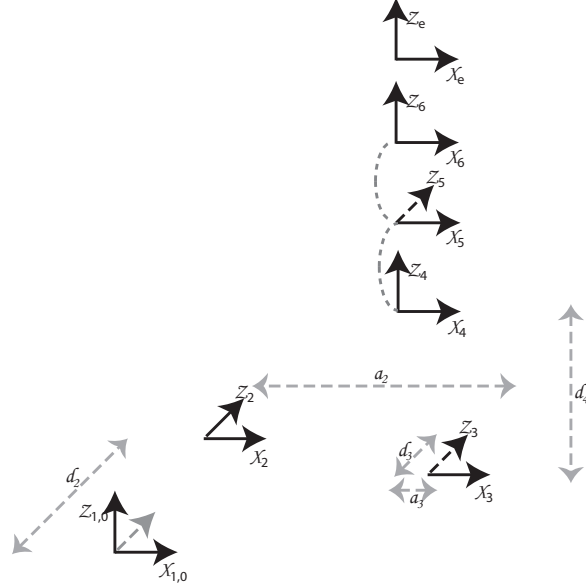


Figure 1: Joint frames in zero configuration.

Wrist lock ($s_5 = 0$) occurs when joint 5 is fully extended. The singular direction is angular: the end-effector cannot rotate around x_4 and maintain position of the end-effector simultaneously.

The null-space motion consists of rotating joints 4 and 6 in opposite directions. This does not move the end-effector, but it does change the singular direction. The goal is to change the singular direction so that x_4 is perpendicular to the desired rotation in operational space, at which point the singularity will no longer be an obstacle.

Head lock ($d_4 s_{23} + a_2 c_2 + a_3 c_{23} = 0$) occurs when the end-effector is over the base. The singular direction is y_1 (i.e. z_2, z_3).

The null-space motion consists of rotating joint 1, which changes the singular direction. Once again, the goal is to use the null-space motion to change the singular direction so that it is perpendicular to the direction you want to move in operational space.

Elbow lock ($d_4 c_3 - a_3 s_3 = 0$) occurs when joint 3 is fully extended. In approximation neglecting shoulder offsets, the singular direction is y_3 (i.e. z_4): toward the base.

The null-space motion consists of rotating joints 2 and 3 in opposite directions, which either bends or straightens the elbow. You should bend the elbow if the goal position is closer to the base than the end-effector, or straighten it as much as possible if the goal position is further from the base (remember that in the control the manipulator enters a singularity region long before the corresponding analytic singularity).

Unlike the other two singularities, the null-space motion of elbow lock does *not* change the singular direction. Instead, moving in the null space (i.e. bending the elbow) quickly takes you out of the singularity. This is the difference between a “type I” singularity such as Puma’s elbow lock, and a “type II” singularity such as Puma’s wrist lock and head lock.

Creating the selection matrix

You must separate the singular directions from the non-singular directions. To do this, create an $n \times 6$ selection matrix, where n is the number of orthogonal directions in which the Puma can move.

For example, suppose you are in a head-lock:

1. Write down the singular directions, as 6×1 vectors. Since we are in only one singularity in this example, there is only one singular direction:

$$\begin{bmatrix} -s_1 \\ c_1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

2. Find n orthonormal vectors that are orthogonal to the singular directions, where n is 6 minus the number of singularities. Since we are in only one singularity in this example, we can find 5 such vectors:

$$\begin{bmatrix} c_1 \\ s_1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

3. Turn those n vectors into row vectors, and make an $n \times 6$ matrix. Now you have a selection matrix for the head-lock singularity.

$$\begin{bmatrix} c_1 & s_1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Since the Puma has 3 singularities, and it can be in any combination of singularities, including all three at once. So you need to find $2^3 = 8$ different selection matrices (or 7, if you don't count the no-singularity case, when $S = I$). (Hint: If you align the singular directions with joint axis of certain frames, all you need then is to compute the rotation matrices into those frames and extract their columns.)

You can use a switch statement with 8 cases, or three "if" statements if you're reasonably clever about how you choose the nonsingular directions.

Finding a joint torque to escape the singularity

Now, if the goal position is outside the singularity, you must find a torque Γ_0 in null space that will move the PUMA out of the singularity(s) and towards the goal.

The steps are:

1. Find a vector Δx that represents the desired linear/angular displacement in operational space. A good approximation of this is the force f' that gets passed into **OpDynamics()**, divided by k_p .
2. For each singularity, find a displacement Δq in joint-space that will take the Puma out of the singularity and toward the goal. Scale Δq by the dot-product of Δx and the singular direction, so that Δq is proportional to how badly the Puma needs to escape the singularity.
3. Sum the Δq values together, treat the sum as if it were $q_d - q$, and use a dynamic PD controller to find the torque $\Gamma_0 = A(-k_p(-\Delta q) - k_v v)$. Better yet, use velocity saturation. Don't worry about B or G .

The bad news is step 2 is very ad-hoc and vague. There are some concrete suggestions below about how to do this for each singularity.

The good news is that you don't have to worry about how these joint-space torques will interact with the forces that you apply in operational space. The selection matrix will filter the joint-space torques, so that they occur entirely in the null space of the Jacobian. This means that you can afford to be a bit sloppy with your joint-space torques.

As an extreme example of sloppiness, suppose you're in elbow lock. The correct way to escape this singularity is to rotate joints 2 and 3 in opposite directions, so that the end-effector moves straight toward the base. But you could get away with only applying a torque to joint 3, and neglecting joint 2 entirely.

Here are some concrete suggestions for the null-space motions. Bear in mind, these are only suggestions; you might come up with a better idea.

Elbow Lock • Find the dot product between Δx , and a vector from the base to the end-effector (i.e. the vector $(gv.x[0], gv.x[1], gv.x[2])$). Call this the radial displacement.

- If the radial displacement is positive, then the elbow needs to straighten. Let Δq_3 be the difference between q_3 (the elbow angle) and $\pi/2$ (the straight angle), times the radial displacement. Let $\Delta q_2 = -\Delta q_3$, so that the two joints move in opposite directions. Make sure you get the signs right.
- If the radial displacement is negative, then the elbow needs to bend. Let Δq_3 be the radial displacement, possibly multiplied by a fudge factor to account for the fact that we're converting between meters and radians. Again, apply $-\Delta q_3$ to joint 2, and make sure the signs are right.
- If you get the rest of this working, then consider this: when you bend the elbow, there are two ways to bend it. One good choice is to take whatever small bend currently exists, and increase it. But a better choice is to bend the elbow in whichever direction moves joint 2 away from the nearest joint limit.

Head Lock In head lock, the singular direction is along the y_1 axis. The goal is to rotate joint 1 so that the projection of Δx in the (x_1, y_1) plane is perpendicular to y_1 , and parallel to x_1 .

- Project Δx into the (x_1, y_1) plane.

- Calculate the magnitude of the projected vector, and the angular difference between the projection and x_1 .
- Set Δq_1 to the magnitude times the angular difference, making sure that the sign brings the projection closer to x_1 .
- If you get the rest working, then (again) consider modifying the solution to avoid the q_1 joint limit, even if it means going the long way around.

Wrist Lock This is almost identical to head lock, except that the singular direction is angular instead of linear. The singular direction is the rotation around x_4 .

- Project the Δx axis of rotation (i.e. the last three components of the 6×1 vector, not the first three) into the (x_4, y_4) plane.
- Calculate the magnitude of the projected vector, and the angular difference between the projection and y_4 .
- Set Δq_4 to the magnitude times the angular difference, making sure that the sign brings the projection closer to y_4 . Set $\Delta q_6 = -\Delta q_4$.
- If you get the rest working, then (yet again) consider modifying the solution to avoid the q_4 joint limit.

References

- K. Chang, O. Khatib, "Manipulator Control at Kinematic Singularities: A dynamically consistent Strategy", *Proc. IEEE/RSJ Int. Conference on Intelligent Robots and Systems*, Pittsburgh, vol. 3, pp. 84-88, August 1995.
- D. Oetomo, M. H. Ang Jr., and S. Y. Lim, "Singularity Handling on Puma in Operational Space Formulation", *Experimental Robotics VII, LNCIS (Lecture Notes in Control and Information Sciences) 271, Proc. of Seventh Intl. Symp. of Experimental Robotics*, pp. 491-500, March 2001.