Homework 3 – out 9/17/03; due 9/24/2003
15-491 CMRoboBits

1 Introduction

This homework covers robot motion. The assignment includes some written questions and some programming. You will learn the robot's walking and frame based motion. The assignment provides you with a chance to outperform the fastest walk developed by the CMPack team. Enjoy it and good luck!

2 Background

You will learn how different walk parameters affect the motion of the robot. There are 51 walk parameters, which are enumerated below. To set the walk parameters, you will need to modify genwalk.cc in the directory: ~/dogs/agent/Motion/genmot. Tweaking parameters is a commonly occurring task in the field of robotics. There are a variety of ways to go about it, but no one method has been proven always suitable. Some of the ways are one or a combination of the following methods:

- Implement an iterative procedure to change parameters according to some given rule.
- Develop a search algorithm that walks through the space of parameters in an informed manner according to an objective function to optimize.
- Develop a learning approach in which the robot experiments with different parameters, gets feedback and processes it into automatic changes of the parameters.
- Manual tweaking of the parameters through trial and error, and acquire a feeling for how to change the parameters manually to achieve the desired performance.

In Homework 2, you had a chance to issue a simple motion command, but for part 2 of this assignment, you will be required to develop a more difficult motion. To create the types of motions that will be required in this lab, a frame-based motion is used. Please see the lecture slides. Some sample motions are located in genkicks.cc and genmisc.cc in the directory: ~/dogs/agent/Motion/genmot. The key-framing technique specifies a set of fixed positions for either the joints or end position, and an amount of time to get from one frame to the next. In this manner, complex motions are decomposed into an ordered set of frames.

3 Questions

Background: A revolute joint is one which is allowed to rotate and may have joint limits specifying boundaries on its rotation. An example of a revolute joint is the human elbow. The other main type of joint in robotics is the prismatic joint. This type of joint is allowed to slide up and down, and may also have joint limits. When not specified, assume that a prismatic joint can never achieve a negative distance.

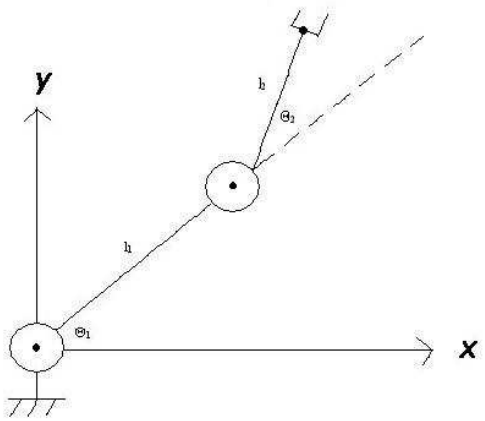
1) Forward Kinematics – RR Arm (7 points)

If we have the same revolute-revolute arm as we had in lecture, with joint limits:

$0 \leq q_1 \leq 180$

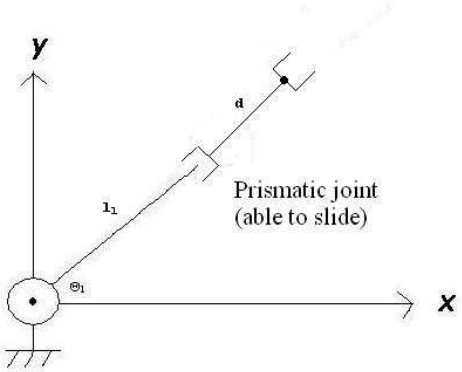
$-90 \leq q_2 \leq 90$

Draw a new coordinate axes with the all the possible positions of the tool shaded in.



2) Forward Kinematics – RP Arm (7 points)

This time we have a revolute-prismatic arm. Solve the forward kinematics problem by solving for (x, y, Θ) of the tool in terms of Θ_1 and d. You may also use the constant l_1 in your equations.



3) Inverse Kinematics – RP Arm (6 points)

We will be using the same diagram as the last question. This time we give you the position of the tool which is (x, y, Θ) = ($-l_1$, 0, -). In other words, $x = -l_1$, $y = 0$, and we don't care about the orientation. What are the values of Θ_1 and d to satisfy this position of the tool?

4 Robot Programming

Complete at least one task from each of the following two parts:
Part 1: (40 points)

- Walk competition 1 (Develop the fastest walk possible)
  - Extra Credit will be awarded to the fastest walk.
- Walk competition 2 (Develop the fastest upright walk)
  - An upright walk is defined as always having both forearms not touching the ground.

Part 2: (40 points)

- Climb onto the charging station
- Roll over (Tuck head and roll over sideways like a dog, starting & ending on stomach)

5 Grading

Questions: 20 points
Part 1: 40 points – Robot walks forward
20 points – Any motion
Part 2: 40 points – Task completed
35 points – Robot almost completes task
30 points – Robot makes progress toward task
20 points – Any motion

6 Programming Information

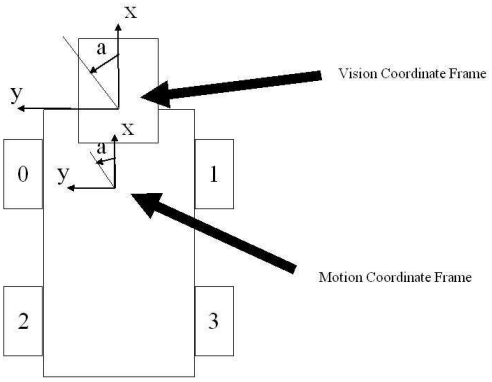
- Working With Walk Parameters
  - All of the walk parameters exist in the file: ~/dogs/agent/Motion/genmot/genwalk.cc . Note that this file contains three different parameter sets that get saved to three different files:
    - /walk_xy.prm /- executed using the MOTION_WALK_TROT command from behaviors, used when desired angular velocity is less than 1 radian/second or have high translational velocities (for example va = .5, vx = 100, vy = 0)
    - /walk_a.prm/ - executed using the MOTION_WALK_TROT command from behaviors, used when the desired angular velocity is greater than 1 radian/second and the desired translational velocity is small (for example if va = MAX_DA, vx=0 and vy=0)
    - /walk_d.prm/ - executed using the MOTION_WALK_DRIBBLE command from behaviors
  - We have created a separate file that contains many of the walk parameter sets that have been used in the past called old_walk_prm.txt. By looking at the file you can get a feel for the range of values that have been used for parameters, and you can try running the old parameters to see what the different walks look like.

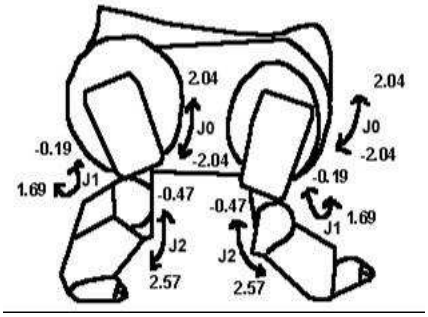
- Testing your own parameters: Edit the parameters in genwalk.cc and save the file. Make sure you know which parameter file your values are saved to, since mostly we are interested in forward motion it is probably best to use walk_xy.prm in order to keep your behavior code simple. Execute the following commands:
  - ~/dogs/agent/Motion/genmot> make
  - ~/dogs/agent/Motion/genmot> ./genmot
  - ~/dogs/agent/Motion/genmot> mount /memstick
  - ~/dogs/agent/Motion/genmot> cp walk_xy.prm /memstick/motion/walk_xy.prm
  - ~/dogs/agent/Motion/genmot> umount /memstick
- Kinematic Errors: Note that the term "errors" can be somewhat confusing. Kinematic errors occur when the walk parameters ask the walk engine to position the leg in a way that is not physically possible. In these cases the walk engine will bound the motion of the leg so that it is within the physical limits of the robot. It is important to make sure that when testing the validity of your new parameters you are testing against the type of motion you plan to be executing. Check in genmot.cc to see what velocities are being tested! Note that genmot needs always to be executed, independently of checking errors, as it generates the actual input for the walk engine.

- Working With Frame-Based Motions
  - All frame-based motions exist in the files ~/dogs/agent/Motion/genmot/genkicks.cc (for kicks) and ~/dogs/agent/Motion/genmot/genmisc.cc (get-up motions, dances, blocks, etc). You will find very many examples here of how to do motions, the supporter and goalie blocks in the genmisc.cc file are pretty short examples to get you familiar with how it all works.
  - Creating your own motion: Add your new motion somewhere in genmisc.cc, it doesnt matter where. Its not hard to figure out leg angles once you do a few. We have also written a behavior that would allow you to get the current joint angles of the legs over wlan. To do this:
    - In the file MotionObject.cc uncomment the line that says "#define LIMPDOG". If LIMPDOG is defined, once you boot the robot the gains of the motors will be set to 0, and the robot will not move at all. You can tell that it is on because it will have a yellow LED flashing at the top of its face. Once the robot is on you can position it in whatever position you need to get the joint angle values.
    - Compile your code, and set the behavior on the stick to be "JointDog".
    - Once the robot is on and you are ready to receive wireless messages, press the button on the robot's chin to see a printout of the current joint angles.

        - Unfortunately this method has the drawback that between testing your code and using this behavior you will have to recompile in order to turn off LIMPDOG. If possible talk to a different group about borrowing their stick if they are not currently working, that way you can have one stick for testing the motion and one for using JointDog.
    - Testing your own motion: Save your new motions to a filename that ends in .mot . Then execute the following commands:
        - ~/dogs/agent/Motion/genmot> make
        - ~/dogs/agent/Motion/genmot> ./genmot
        - ~/dogs/agent/Motion/genmot> mount /memstick
        - ~/dogs/agent/Motion/genmot> cp yourmotion.mot/ /memstick/motion/somefile.mot
        - ~/dogs/agent/Motion/genmot> umount /memstick
        - somefile.mot - What should this be? Well, you have two options:
            - Option1: MotionInterface.h contains a list of all possible motion commands. You can create a new motion command, and specify how it is to be executed in Motion.cc and Motion.h (try using grep to find all the places you need to edit).
            - Option2: Use a motion command that already exists, and simply overwrite its .mot file with yours. For example use MOTION_KICK_BUMP in your behaviors and copy /yourmotion.mot/ to k_bump.mot on the memstick.
            - We dont care which method you use for this homework.
- Using the Wireless Network*
    - Configure your stick
        - Determine the host name of the robot being used. There should be a two digit number printed on the robot's back (01,02,03..). The robot's hostname will be ersXY where XY is the robot's number. Configure the stick for the specific robot IP address. Run the program wlan_setup and specify only the host name of the robot (the domain is not needed)
        - example: wlan_setup -i ers01
    - Run the dog
    - Compile the GUI (once)
        - cd ~/dogs/util/chokechain
        - make
    - Run The GUI
        - ./chokechain2 -n ersXY.wv.cs.cmu.edu
        - where XY is the robot number

7 Motion Parameters





Count	Parameter	Measured in	Description
12	**neutral**	mm	Position of leg on ground at some point in walk cycle
1	**period**	ms	Time to complete one cycle where each leg takes a step
12	lift_vel	mm/s	Velocity at which foot is picked up
12	down_vel	mm/s	Velocity at which foot is set down
4	lift_time	% thru period	Percentage thru cycle foot is picked up
4	down_time	% thru period	Percentage thru cycle foot is set down
1	**body_angle**	radians	Set angle of bottom of robot (+ means back is higher than front)
1	**body_height**	mm	Set height of shoulders from ground
1	hop	mm	Amplitude of bounce up and down
1	sway	mm	Amplitude of sway left and right
1	front_height	mm	Set a maximum height for raising front legs
1	back_height	mm	Set a maximum height for raising back legs
51			***(Bold specifies which parameters will make most difference)**