Lab#8: Inverse Kinematics application

Leonardo Fusser (1946995)

Objectives: create a thread to customize specific geometric figures and movements of the robotic arm.

Material: same as previous labs.

Pre-requisite: POSIX pthreads and pdcurses in Windows.

To hand in: see previous lab.

File system structure:

See the following hierarchy. You must populate the files in **orange.**

- **multithread_mover4_v6\pdcurses_test\mover4_v6**
  - can.c
  - task_controller.c
  - log
  - state
  - ncurses_init.c
  - adc.c
  - **sol_mover4**
    - build
    - mover4.c
    - taskKeybd.c
    - taskAuto.c
    - taskDisplay.c
    - kinematic.c
    - **taskInvKin.c**
  - **header**
    - can.h
    - adc.h
    - kinematic.h
    - **public.h**
    - task_controller.h
    - ncurses_init.h
  - **demo**
    - exec
  - **excel_sim**
    - animation_v2
    - export

In the animation_v2 excel file, change the link lengths of the robot arm. They must be set to the mover4 link's lengths.

Lab Work

Working **ALONE**, you must create a new thread named `taskInvKin`

`taskInvKin` thread:

When in kinematics mode, it generates a specific algorithm inside an infinite loop.

It generates all angles by calling the inverse kinematic function `to_angle()`

It moves the robot by calling `move_until()`.

When the 'k' key is pressed, the system runs the `taskInvKin` thread and when the 'j' key is pressed, the system stops running the thread.
Also, switching from one mode to another should be almost immediate: response time should be less than one second.

Also, the menu should be modified accordingly:

Main menu display area **Line 0 to Line 9**

```
*****************************************************************************
* Use the following keys to jog the joints or grip                          *
*                                                                           *
*    q         w        e       r       t                                   *
*  Joint1   Joint2   Joint3   Joint4  Gripper                               *
*    a         s        d       f       g                                   *
*                                                                           *
*  Use also the following keys:                                             *
*  Exit: x - Jog mode: j - Auto mode: n - Kinematics mode: k                *
*****************************************************************************
```

Test your taskInvKin thread

Inside the `taskInvKin` thread try the following scenario in simulation mode:

Using the inverse kinematics function `to_angle()`, write the following in an infinite loop:

```
angles = to_angle(10, 10, 15, -45);
move_until(angles.data[0], angles.data[1], angles.data[2], angles.data[3]);
angles = to_angle(15, -10, 10, 45);
move_until(angles.data[0], angles.data[1], angles.data[2], angles.data[3]);
```

Circular movement in the y-z plane

Write a new API `circle_yz_plane()` that makes a circular movement in the y-z plane with a specified radius and offset:

```
void circle_yz_plane(void)
```

```
#define OFFSET_Z                  15.0
#define OFFSET_Y                   0.0
#define RADIUS                     5.0
#define X0                        10.0
```

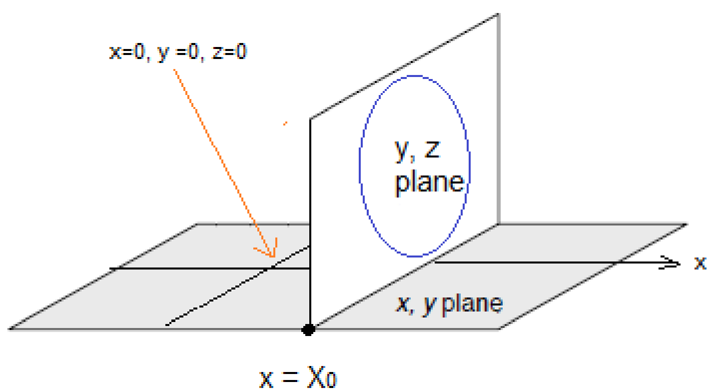
The grip angle must be -40 degrees.

The arm must move by step of 45 degrees.

Therefore, there are 8 possible positions: 0º, 45º, 90º, 135º, 180º, 225º, 270º, 315º, 0º, 45º ...

For every 45º step, the gripper must open for 1 second.

For a smooth movement, reduce the speed to medium.



You must give a demo on the robot.

Upgrading to your private repository

Now it is time to update your repo.

But before, erase the following folder to save space: `.vs\pdcurses_test\v16`

Questions:

Explain what you learned in this lab. What went wrong and what did you learn from your mistakes.

- For the most part, not many issues were encountered. Typical syntax errors occurred, but more specifically, issues with the logic behind writing the movement of the circle in C was present.

  Initially, instead of following basic math principles, the code that was written was overcomplicated, rather than what should have been just a few lines of code. Since the circle movement had to be done in the x, y plane, the only thing that really had to be done was to constantly calculate the x and y values using cosine and sine respectively (x value and grip angle were constant).

  The solution was easily implemented inside one for loop that iterated from 0-degrees to 360-degrees (with 45-degree stepping). Using the to_radians() function populated from before, the x and y values were constantly calculated using cosine and sine for each 45-degree step. The to_angle() and move_until() functions were used to calculate the Mover4 joint angles and to set the Mover4 joint angles respectively.

  The to_radians() function was used in conjunction with the cosine and sine function in C because the angle passed needs to be in radians format.

  The main takeaway here is to not overthink a problem, and to apply basic mathematical principles. Especially for manipulating the Mover4 robot arm.