Assignment in Computer Graphics II
– Assignment 13 –

Computer Graphics and
Multimedia Systems Group
David Bulczak, Christoph Schikora

Assignment 1 [2 Points] Forward Kinematics

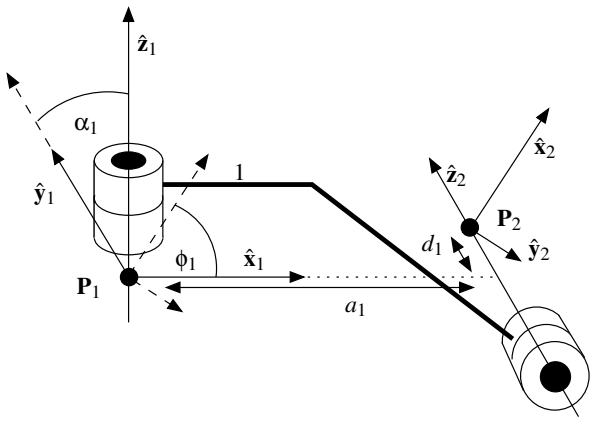
Given the two-dimensional, three-limb (dreigliedriges) model:$\phi_1 = 45°$, $\phi_2 = 270°$, $\phi_3 = 90°$ and

$$P_1 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \quad l_1 = 6,\ l_2 = 3,\ l_3 = 2$$

1. Calculate the end effector X_1, by successively calculating the intermediate points P_2 and P_3 in global coordinates.
2. Specify the workspace of the end effector X_1 and explain briefly your claim.

Assignment 2 [2 Points] Denavit-Hardenberg

Given the imaged three-dimensional model with the values $a_1 = 4$, $\alpha_1 = -\frac{\pi}{2}$, $d_1 = 1$, $\phi_1 = -\frac{\pi}{3}$.



1. Determine a transformation matrix that maps points in the coordinate system $\{P_2, \hat{\mathbf{x}}_2, \hat{\mathbf{y}}_2, \hat{\mathbf{z}}_2\}$ on points relative to the base $\hat{\mathbf{x}}_1 = (1,0,0)^T, \hat{\mathbf{y}}_1 = (0,1,0)^T, \hat{\mathbf{z}}_1 = (0,0,1)^T$.
   Note: Calculate to do this the following matrices:
   $R((1,0,0)^T, \alpha_1)$: Map $\hat{\mathbf{z}}_1$ to $\hat{\mathbf{z}}_2$ ab
   $T(a_1, 0, d_1)$: Move $\mathbf{P}_1$ to $\mathbf{P}_2$
   $R((0,0,1)^T, \phi_1)$: Map $\hat{\mathbf{x}}_1$ to $\hat{\mathbf{x}}_2$ ab.

2. Determine the unit vectors $\hat{\mathbf{x}}_2$, $\hat{\mathbf{y}}_2$, $\hat{\mathbf{z}}_2$ using the previously calculated matrix.
   Note: Check the result based on the sketch.

Assignment 3 [2 Points] Kinematic

In this task you will implement a simple robot. Download the framework `kinematic-framework.zip` and take an initial look on the code.

All relevant files for this programming task can be found in the `Kinematic` folder. Everything else can be assumed to be a black box.

To build the project in your preferred development environment use the included CMake project ("CMakeLists.txt"). CMake can be downloaded from the following website: http://www.cmake.org/. Use the instructions on the page http://www.cmake.org/cmake/help/runningcmake.html and the tutorial page to create the project.

The program uses forward and inverse kinematic to realize a simple robot. The forward kinematic is used to estimate all effectors / joint positions from the robot description with angels and bone lengths. Without a forward kinematic you will see nothing on the screen!. The inverse kinematic recomputes new angels for user movement of the end effector. Also the workspace of the robot is visualized.

Implement the kinematic in `Kinematic/Kinematic.cpp` as presented in the lecture. The code contains some hints and outlined code example which should help you to use right data structures and to store your results to the from use presupposed structures.
Important: In the `KinematicTool.cpp` `initialGL()` method is a relative path to `Robot.txt` it can be necessary to adjust this path to run the project.
Important: Implement the methods in the following order.

1. `getMinWorkspace`: Estimates the minimal distance to the pivot point the robot can reach. This is part of the workspace estimation.
   Hint: controlRobot_ is a pointer, thus you can directly manipulate his values with out using a copy.

2. `getMaxWorkspace`: Estimates the maximal distance to the pivot point the robot can reach. This is part of the workspace estimation.
   Hint: controlRobot_ is a pointer, thus you can directly manipulate his values with out using a copy.

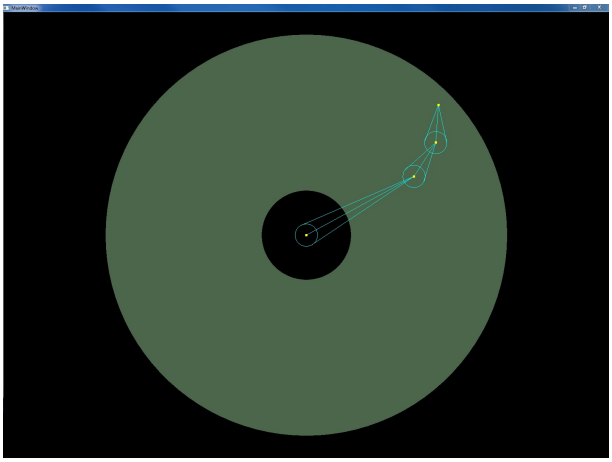
3. `evalEffectors`: In this function you have to implement the forward kinematic, which estimates the effectors / joint positions of the robot. The end effector has to be stored.
   Hint: controlRobot_ is a pointer, thus you can directly manipulate his values with out using a copy.
   Hint: The pivot point is stored severally and not stored initially in the effectors structure.
   Important: The end effector has to be stored a second time (see hints in the code), besides the effector structure.

4. `estimateInverseKinematic`: In this function you have to implement the inverse kinematic, which recomputes the robot angels. You will have to do some matix calculations. We have prepared a small C++ matrix class for you `MatrixC` which has all mathematical operation you will need and a print function for debugging. In `Kinematic/Kinematic.cpp` you can find some hints and examples for the use of `MatrixC` (the `testMatrix` method shows some matrix output to help you when you try to move the end effector). As further help: you have to do the following steps in this method:
   4.1. Get pivot point, current angels and current end effector of the robot and store them in matrix form. Do the same for the new end effector (method value).
   4.2. Create and fill the Jacobian matrix. **Hint:** We used a triple `for loop` for that.

4.3. Do the matrix calculations

4.4. Loop the calculation with a small distance error and a maximal loop count, otherwise the loop will not terminate. **Hint:** You can use the forward-kinematic `evalEffectors()` to recalculate your current end effector. Before doing the distance estimation store your current values for this in the robot an call `evalEffectors()`.

4.5. If you not use the previous hint store the new angels in the robot and execute `evalEffectors()` to update the effectors / joint positions.



Hand in: 1.2.2016, at beginning of the lecture or until 12:00 in the mailbox of the chair (next to room H-A 7107) and send files corresponding to the programming task to johnfr93@gmail.com.