Transfer of *RoboSim* from C to Java

A Software Practical by Johannes Schützner under the direction of Dr.Thomas Bräunl. .

1.)Informations to RoboSim

This program has been generated for a software practical by Johannes Schützner, under the direction of Dr.Thomas Bräunl, University of Stuttgart, in 1996. It is based on the "Studienarbeit" by Rainer Pollak, made in 1992 and programmed in C.

1.1)Introduction

RoboSim is a program which simulates a six-link robot. You can control the robot in several ways and manipulate it, for example giving the links new lengths or changing the weights belonging to the indicated angles. It is also possible to make the robot save pre-allocated positions and to teach the robot a sequence of movements. Also, this software can be used to control a real robot.. This robot is located in the robot laboratory of the Department for Computer Science, University of Stuttgart. In the following passage the possible commands are explained in more detail.

1.2)Useful Advices

This Java applet needs a reserved place of 920x670 pixels. Otherwise parts of the drawing are hidden by the panel on the right side. If you can't see a robot between the control panels, it is useful to click several times between the panels. If the robot seems to stop while doing a move and the scrollbar labels disappear, it might be helpful to move the mouse across the robot. This applet takes three arguments from the calling html-page. The first two arguments, which are called "about" and "where", take the address of the corresponding file path or html-page. This depends on the third argument ("where") which can take the value "netscape" or "appletviewer". If you are starting this program with "netscape", nothing has to be done.If you want to strart it with the Applet Viewer, you should change the value of "where" from "netscape" to "appletviewer" and give "about" and "help" the path to the file in which these text documents are located in your home directory.

1.3)Restrictions

- Since there exist a lot of security restrictions, the loading and producing of control programs can only be done within the Applet Viewer. Unfortunately, applets started with Netscape are not allowed to do this.
- The real robot can only be controlled by a special RoboSim version, because its control needs C-code which has to be integrated into this Java program. This integration would make it impossible for this program to run on every computer system. So the integration has been realized only on a limited version.

1.4)The Commands

Manual Control: This command is destinated to control the robot and its rotation links directly by modifying the angles corresponding to these links. The angle values are restricted to sensible ranges which are shown above the scollbars. They are as follows:

Angle 1: 180.0 degree to -180.0 degree

Angle 2: 30.0 degree to -210.0 degree
Angle 3: 30.0 degree to -210.0 degree
Angle 4: 180.0 degree to -180.0 degree
Angle 5: 90.0 degree to -90.0 degree
Angle 6: 180.0 degree to -180.0 degree

The real robot has not got the facilities of angles 4 and 6, so these angles are not available to the user.

Auto Control: You can control the robot by giving it a target position (x-, y-, and z-value) with this command. The movement is executed after a click on the “execute button” just below the scrollbars of the auto control. If a target point is unreachable, the robot moves back to the last position. By pushing on the other button in the auto control area, you can select, whether the robot should go directly or not to the target.

Determination of the arm lengths: This command allows you to vary the length of the arms of the robot. Also there exists a crash test . If this one succeeds, the change of the arm lengths is not executed. The execution is only performed, if the corresponding “execute button” is pressed. The possible new lengths are limited to sensible ranges shown above each scrollbar. They are as follows:

Arm 1: 15.0 to 4.5
Arm 2: 15.0 to 2.5
Arm 3: 15.0 to 2.5

If the real robot is controlled, this command can‘t be selected, because the real robot has got arms with unchangeable lengths.

Angle weights: Again the values given to the text fields are only accepted after a push on the “execute button”. Then you have new angle weights. They are needed to obtain the best solution of the inverse cinematic. The values have to be positve integers.

Point of view: You can change the point of view by dragging the robot with the mouse to the left and to the right side. By using the zoom scrollbar, you are able to encrease and decrease the size of the robot.

Speed: You can select your required speed with the corresponding scrollbar. By switching off the “double buffering” mode, the robot motion is much faster.

Define/Go to positions: Three positions can be defined simply by pushing the “define pos button” of your choice. By doing this, the six actual angles of the robot are saved. This position can be restored by pushing the equivalent “go to button”. The first button (“Go to Init”) holds the angles of the robot at the beginning. If you want to see an example movement, push the “example button”.

Virtual/Real robot: You can choose between the virtual robot as shown in the middle of your screen and the real robot of the Department for Computer Science. Selecting the real robot, the arm lengths and the angles four and six can‘t be changed, since the real robot consists of only four links.

Open File: By selecting this button, you can load a file which either has been written “by hand” or has been produced with the “Teach button”. How to write executable files and use the “Teach button”, see the following links.

Teach: After having activated this button, a file dialog window appears in which you can choose the name and the directory of the file which will be produced and written automatically. By doing this, the “Teach button” will be replaced by the “Enter button” and the “Stop button” will be enabled. Now you can control the robot as usual, but every time you push the “Enter button”, the actual robot angles are written into the indicated file. This mode is only stopped, if the “Stop button” is pressed. At the beginning of this mode, the actual linklengths, speed, and zooming factor are stored.

Writing own control programs: You can write control programs such like these ones being produced by the “Teach button”, but you have got more command power. Every command starts with a command character and is completed by the arguments needed for the command. Every input, e.g. command character and arguments, must be in a seperate line. Otherwise the required motion of RoboSim cannot be guaranteed. The possible commands are as follows:

- *m* This character stands for manual control. It has to be followed by six double numbers representing the new angles. Remember that every command and every number has to be in a row of its own.
- *a* indicates auto control. In the next three lines there must be the three coordinates of the target point.
- *l* By choosing this character, which represents linklengths, you can determine new lengths for the links. These lengths has to be in the allowed ranges.
- *w* is the abbreviated form for angle weights. The arguments of this command are the six new angle weights which are integers.
- *s* represents speed. It has only one argument - the new speed. The range of possible values starts at 1(slow) and goes up to 82 (fast).
- *h* signifies that the content of the following line should be in the range of the aperture of the robot’s hand. 0.0 means a fully open hand and 0.5 a totally closed one.
- *z* indicates the zooming factor. The zooming factor range is between 0.1 (very tiny) and 3.5 (large).
- # Comments start with # and last till the end of line.

The system is not case sensitive.

2.)Documentation of the Program

The source code of the Java applet *RoboSim* is encapsulated in the package “RoboPackII”. This package consists of tenJava files which are called:

- RoboProto.java
- Initialize.java
- SmallClasses.java
- Cinematic.java
- MoveMethods.java
- ThreeD.java
- Matrix3D.java
- FileHandle.java
- ErrorWindow.java
- Win.java

On the following pages, the tasks of these files are explained more comprehensively. Since this program is based on a C-program, its structure isn‘t pure object-oriented. The influence of imperative programming is quite often getting visible.

2.1)RoboProto.java

The RoboProto class is the “main class” of this program package. The user interface is defined in this class, and the reaction to most of the events such as mouse clicks is determined here. The user interacts with this class.To make this possible, this class is derivated from the “Applet class”. The most important methods are part of this file and the specific values of the robot are also held in this class.

The head of this class looks like this:

```
package RoboPackII;

import java.awt.*;
import java.io.RandomAccessFile;
import java.io.IOException;
import java.applet.Applet;

public class RoboProto extends Applet
{ ...
```

The methods of this class are:

- public void init() : mainly initializes the user interface
- public void start() : repaints the robot only
- public void stop() : makes nothing
- public void repaint() : calls the repaint method of the “ThreeD” instance
- public void menuEnable() : enables the menu panels
- public void menuDisable() : disables the menu panels
- private double roundTo2(double arg) : rounds arg to two places behind the colon
- private String formatTo5(double number) : returns number as a string of size 5
- private void updateManualLabel(int no), updateLinkLabel(int no), updateAutoLabel(int no): each of these methods updates the corresponding label number “no”
- private void updateOldLinkLabel(int no, double actValue) : this method is called, if the linkLenth is changed, but the change is not yet executed
- private void updateScrolledAutoLabel(int no): this method is called, if an auto scrollbar has been used, but the robot hasn‘t yet moved to the new position, because the execute button has to be pushed to make it do that
- public void writeError(String text) : creates an instance of “ErrorWindow” and passes text as argument
- private void actualizeAutoPanel() : updates the whole auto panel at once
- public void manualInput(double newAngles[]) : the angle dates read out of a file and given in newAngles are interpreted and, if it is possible the change is done by a „manual-Change“-call
- public void linkLengthInput(double newLength[]) : the same with link lengths
- private void linkLengthChange(double newLength[]) : all work and calls of other methods that have to be done to execute a linklength change can be found here
- private void manualChange(double newValues[]) : the same with angles
- private void manualChange(int no, double newValue) : makes the same like the previous method, but realizes the change of only one angle, the „no“ specified angle
- public void autoInput(double input[]) : the target points read out of a file are interpreted,

especially an „autoChange“- call is done

- private void autoChange(double input[]) : all work and calls of other methods that have to be done to reach a target can be found here
- public void weightsInput(int newValues[]) : “newValues” are interpreted as new angle weights
- private void installRealRobot() : the modification of the user interface which follow the selection of the real robot mode finds place here
- private void tidyUpRealRobot() : inverse method to installRealRobot, the virtual robot is rebuilt again
- private void exampleExection(): executes an example motion
- private void saveActualPosition(int where) : the actual angle values are saved in the position number „where“
- private void loadPositionNo(int which) : the position number „which“ is loaded
- private void newSpeed() : a new speed depending from the speed scrollbar is calculated
- public void speedInput(int newSpeed) : the new speed read in from a file is interpreted
- public void zoomInput(float newValue) : the new zooming factor (“newValue”) read in from a file is interpreted
- public void handInput(double newValue) : the new hand openness factor (“newValue”) read in from a file is interpreted
- private void handChange(double newValue) : the hand change is executed with „newValue“ as new hand openness factor, to open or close the hand, it‘s sufficient to change the x-axis value only
- public boolean mouseDown(Event e, int x, int y) , mouseDrag(Event e, int x, int y) : if the event target is the robo canvas, then the corresponding method of the „ThreeD“instance is called, else nothing is done
- public boolean action(Event e, Object o) : this method is called by an event, if the event can‘t be interpreted, the event is passed to “handleEvent”
- public boolean handleEvent(Event e) : this method tries to interpret the events “action” failed. If handleEvent fails as well the event is passed to its super method

2.2) Initialize.java

This file consists of the class “RobotInit” which initalizes all robot specifying variables in the static method “now”. This method is called only once in a program session, in the “init” method of the “RoboProto” class.

The head of this class looks like this:

```
package RoboPackII;

final class RobotInit
{ ...
```

The method of this class is :

- public static void now(RoboProto rob) : the robot specifying values of “rob“ are instantiated here

2.3) SmallClasses.java

This file consists of some useful small classes which could have been created as structures in “C“ as well.

The head of this class looks like this:

```
package RoboPackII;
```

```
import java.awt.Point;
...
```

T*he classes of this file are:*

- class Points : This class holds the three coordinates‘ values of a three-dimensional point. These values are public.
- class Edges : This class holds two public short variables representing an edge.
- class AngleDates :This class is home of the dates corresponding to every angle. These dates are the double variables max, min, and act being public.
- class EdgesPointsList : This class holds Points p1, p2, and short visible (all public).
- class Areas : This class holds public short p1, p2, p3, p4.
- class AreasPointsList : This class holds public a four element array of Point and of Points, double zmin, nx, ny, nz, and C.
- class Sort : This class holds public double z and char part.
- class RefInt : This class is helpful, if you want an ‚int‘ argument of a method to be a ‚call-by-reference‘ one., since primitve data-types are always called-by-value in Java.
- class RefDouble : This class is helpful, if you want a ‚double‘ argument to be a ‚call-by-reference‘ one. Ihaven‘t found other possibilities to achieve this goal. A Double instance is an object and would be called by reference, but there exist no ways to manipulate the value of a Double or Integer object. Thus these objects aren‘t senseful to achieve the goal.

2.4) Cinematic.java

This file consists of the methods the robot needs to calculate his new angle values or his new position. The name of the only class of this file is CinematicCalculations.
The head of this class looks like this:

```
package RoboPackII;

final class CinematicCalculations
{...
```

The methods of this class are:

- public static void TO6(AngleDates[] angles, double[] linkl, double[][] T): The ‘angles’- and ‘linkl’-arrays have to consist of 6 elements, the ‘T’-array is a 3x4-array. This method calculates the values of the ‘T’-array.
- public static void calculatePos(AngleDates[] angles, double[] pos,RoboProto rob): ‘Angles has to be a 6-elements- and ‘pos’ a 3-elements-array. The hand position of the robot is calculated.
- public static int inverseAngle1(double[][] T, double[] angNew, AngleDates[] angOld, int reminder, RoboProto rob) : ‘T’ has to be a 3x4-array, ‘angNew’ consists of 2 elements and ‘angOld’ of 6. Returns the new value of reminder.
- public static int inverseAngle3(double[][] T, double[] angNew, double[] linkl, int reminder, RoboProto rob) : ‘T’ has to be a 3x4-array, ‘angNew’ consists of 2 elements and ‘linkl’ of 3. Returns the new value of reminder.
- public static int inverseAngle2(double[][] T, double[] angNew, double[] linkl, double angle1, double angle3, double[] s, double[] c, int reminder, RoboProto rob) : ‘T’ has to be a 3x4-array, ‘angNew’ consists of 2 elements and ‘linkl’ of 3. ‘s’ and ‘c’ consist of 6 elements. Returns the new value of reminder.
- public static int inverseAngle4(double[][] T, double[] angNew, AngleDates[] angOld, double angle1, double angle2, double angle3, double[] s, double[] c, int reminder, RoboProto rob) : ‘T’ has to be a 3x4-array, ‘angNew’ consists of 2 elements and ‘angOld’ of 6. ‘s’ and ‘c’ consist of 6 elements. Returns the new value of ‘reminder’.

- public static int inverseAngle5(double[][] T, double[] angNew, double a1, double a2, double a3, double a4, double[] s double[] c, int reminder, RoboProto rob) : ‘T’ has to be a 3x4-array, ‘angNew’ consists of 6 elements.’s’ and ‘c’ consist of 6 elements. Returns the new value of ‘reminder’.
- public static int inverseAngle6(double[][] T, AngleDates[] angles, double[] angNew, double a1, double a2, double a3, double a4, double a5, double[] s, double[] c, int reminder, RoboProto rob) : ‘T’ has to be a 3x4-array, ‘angles’ consists of 6 elements and ‘angNew’ of 6. ‘s’ and ‘c’ consist of 6 elements. Returns the new value of ‘reminder’.
- public static int inverseCinematic(double[][] T, double[] angNew,AngleDates[] angOld, double[] linkl, int result,int f_a_g, RoboProto rob) : ‘T’ has to be a 3x4-array and is frame 6 relatively to frame 0. ‘angNew’ consists of 6 elements as ‘angOld’ does.’linkl’ has got 3 elements. ‘angOld’ are the actual angles and ‘linkl’ the actual linklengths. If result is 0 then a valid combination of angles has been found. Returns the new value of ‘result’.

2.5) MoveMethods.java

This file consists of the methods which make the robot move around. The only class of this file is called AllAboutMoving.

The head of this class looks like this:

```
package RoboPackII;

final class AllAboutMoving
{...
```

The methods of this class are:

- public static void calculateAngle(double[] input, double[] aB,double[] aE, RefInt stepsB, RefInt stepsE, RoboProto rob) : stepsB and -E should be call-by-reference arguments, so it is sensible to declare them as RefInt (see SmallClasses.java). That means that a reference to the object is passed.
- public static void MoveRobo (boolean showTheMove, Points[] pl, Points[] line, double[] moveAngle,AngleDates[] angle,int steps, RefInt errorcode, RefDouble angleY, RefDouble angleX, RefDouble d, int show, RefInt indx, double[] location, RoboProto rob) : line consists of 2 elements, angle of 6 and location of 3. errorcode, angleY, angleX, d and indx should be of my special types, since these variables simulate call-by-reference- arguments. This method executes the movement depending on the number of steps.
- public static void RotationY(Points[] pl, double angle, int start, int number, RoboProto rob) : The robot’s points are newly calculated depending on ‘angle’.
- public static void RotationK (Points[] pl, double angle, Edges axis,int start, int number, RoboProto rob) : analogue to RotationY’
- public static void Nvector (Points[] pl, Areas[] al, AreasPointsList[] apl, int start, RoboProto rob) : ‘apl’ is calculated.
- public static boolean Crash(Points[] pl, int[] rpl, RoboProto rob) : Tests if the robot’s movement would cause a crash.
- public static void PosRobo (Points[] pl,double[] turnangle,AngleDates[] angle, RoboProto rob): The position of the robot is calculated.
- public static void angleCalculation(double x, double y, double z,RefDouble angleY, RefDouble angleX, RefDouble d) : ‘x’ is the new projection center, ‘angleY’ the rotation angle around y-axis, ‘angleX’ the rotation angle around x-axis and ‘d’ the position on the z-axis.
- public static void copyPointList (Points[] PL, Points[] pl, RoboProto rob) : The elements of ‘PL’ are copied to ‘pl’.

- public static void calculateLocation(double[] location,double angleY, double angleX): The location of the robot is calculated.

2.6) ThreeD.java

This file contains a set of classes to parse, represent and display a 3D wireframe model of the robot in Wirefront.obj format. It is based on a program example from 'Hooked on Java'.The classes within this file are :

- class FileFormatException extends Exception: Indicates a file format error
- class Model3D : The representation of a 3D model.
- class ThreeD implements Runnable : A class to put a 3D model into a page.

The head of this class looks like this:

```
package RoboPackII;

import java.applet.Applet;
import java.awt.Graphics;
import java.awt.Color;
import java.awt.Event;
import java.awt.Image;
import java.awt.*;
```

The method of the class FileFormatException :

- public FileFormatException(String s) : Constructor of this class which calls the super construtor only.

The methods of the class Model3D :

- Model3D (RoboProto datarob) : Construtor of this class which builds up the robot into a page.
- int addVert(float x, float y, float z) :Adds a vertex to this model .
- void add(int p1, int p2) : Adds a line from vertex p1 to vertex p2.
- void transform() : Transforms all the points in this model.
- private void sort(int lo0, int hi0) : Sorts the vertices from index lo0 through index hi0.
- void compress() : Eliminate duplicate lines.
- void paint(Graphics g) : Paint this model to a graphics context. It uses the matrix associated with this model to map from model space to screen space. This version uses double buffering If the user has selected the highest possible speed, this method only calls te following one.
- void paintWithoutDoubleBuffering(Graphics g) : This method mainlydoes the same thins like the previous method. The only difference is that this one doens't use double buffering, so it is much faster (at leat within Netscape).
- void findBB() : Find the bounding box of this model.

The methods of the class ThreeD :

- public ThreeD(Canvas inCanvas, RoboProto dataRobot) : Cunstructor of this class.
- public void setScale(float newScale) : Sets the new scale factor.
- public void run() : The 'run'-method of this thread. Initializes the Model3D instance.
- public void start() : Starts the thread.
- public void setVert(int no, Points newCoordinates) : Sets and changes the specified vertix.of the model.
- public boolean mouseDown(Event e, int x, int y) : The current mouse position is saved.
- public boolean mouseDrag(Event e, int x, int y) : A mouse drag causes a turn of the model. The necessary calculations to realize this are controlled here.

- public void repaint() : The paint()-method of this class is called with the graphics object of the canvas as argument.
- public void paint(Graphics g) : A few calculations and a call of the paint()-method of Model3D are done.

2.7) Matrix3D.java

This file is originary of ‘Hooked on Java’. It represents a fairly conventional 3D matrix object that can transform sets of 3D points and perform a variety of manipulations on the transform.
The head of this file looks like this:

```
package RoboPackII;

class Matrix3D
{...
```

The methods of this class are:

- Matrix3D () : Constructor.
- void scale(float f) : Scale by f in all dimensions.
- void scale(float xf, float yf, float zf) : Scale along each axis independently.
- void translate(float x, float y, float z) : Translate the origin.
- void yrot(double theta) : Rotate theta degrees about the y axis.
- void xrot(double theta) : Rotate theta degrees about the x axis.
- void zrot(double theta) : Rotate theta degrees about the z axis.
- void mult(Matrix3D rhs) : Multiply this matrix by a second: M = M*R.
- void unit() : Reinitialize to the unit matrix.
- void transform(float v[], int tv[], int nvert) : Transform nvert points from v into tv. v contains the input coordinates in floating point. Three successive entries in the array constitute a point. tv ends up holding the transformed points as integers; three successive entries per point.
- public String toString() : Returns the String representation of the matrix.

2.8) FileHandle.java

This file provides a file dialog to read and write files. It consists of the class FileHandle.
The head of this class looks like this:

```
package RoboPackII;

import java.awt.*;
import java.io.*;
import java.util.Date;

final class FileHandle
{...
```

The methods of this class are:

- public static void readFile(RoboProto rob) throws IOException : This method reads in a file that can be selected from the offered file browser. The robot is controlled according to the commands in the specified file.
- public static RandomAccessFile openFile(RoboProto rob) throws IOException : Opens the file being selected by the file browser for later writing with:
- public static void writeFile(RandomAccessFile ranFile, RoboProto rob) : Writes in the file specified by “ranFile”.

- public static void closeFile(RandomAccessFile ranFile) : Closes “ranFile”.

2.9) ErrorWindow.java

This file consists of the “ErrorWindow” class which shows an error text in a seperate window.
The head of this class looks like this:

```
package RoboPackII;

import java.awt.*;
import java.io.*;
import java.applet.Applet;

class ErrorWindow extends Frame
{...
```

The methods of this class are:

- ErrorWindow(String errorText, RoboProto rob) : Construtor of this class which takes the error text as argument and writes it in a newly created window. Disables the panels of the user interface.
- public boolean action(Event e, Object o) : Enables the panels of the user interface and destroys itself.

2.10) Win.java

This file contains the “Win”class. The “Win”class is used to show the “about text” or the “help text” in a window, if the program has been started from the Applet Viewer.
The head of this class looks like this:

```
package RoboPackII;

import java.io.*;
import java.awt.*;

public class Win extends java.awt.Frame
{...
```

The methods of this class are:

- public Win(RoboProto rob, int type) : Constructor of this class
- public void init() : The window is built up and a file is read in.
- public boolean action(Event e, Object o) : A click on the “O.K.” button causes the destruction of this window.