```
    conf.setx(20);
    conf.setjoint(1, 1);
    rob2.place(conf);
    draw(psarea, rob2.cur_shape());
    psarea.pageready();
}
```

3.4 File IO

Just like PlaGeo, RS2 provides the possibility to read or write its types from or to an iostream with the help of the rightshift or leftshift operator. So, to read a robot, a configuration and a geolist from standard input, the following suffices:

```
pl_robot robot;
pl_configuration conf;
pl_geolist obst;
cin >> robot >> conf >> obst;
```

Apart from writing separate objects, there is also the possibility of writing and reading simple scenes. A simple scene consists of a list of obstacles (a `pl_geolist`), one robot, a start and a goal configuration. The following routines do the trick:

```
void write_scene(ostream &ofile, pl_geolist &obstacles, pl_robot &robot,
     const pl_configuration & startpos, const pl_configuration & endpos)

void save_scene(char *filename, pl_geolist &obstacles, pl_robot &robot,
     const pl_configuration & startpos, const pl_configuration & endpos)

int read_scene(istream &ifile, pl_geolist &obstacles, pl_robot &robot,
                 pl_configuration & startpos, pl_configuration & endpos)

int load_scene(char *filename, pl_geolist &obstacles, pl_robot &robot,
                   pl_configuration &startpos, pl_configuration &endpos)
```

The first argument can either be an iostream (for `write_scene` and `read_scene`) or the name of a file (for `save_scene` and `load_scene`). Using this latter form, nothing about iostreams need be known; to read from standard input, supply NULL as filename, or a string consisting of a single hyphen ("-"). The same holds for writing to standard output.

The routines `read_scene` and `load_scene` return an integer value. This value indicates how much has been succesfully read. If the value is 0, an error has occurred (nothing is read). The return value 1 indicates that only obstacles have been read. If the value is 2, also a robot has been read and return value 3 states that everything has been read.

`c.get_pose()` returns the orientation and position of the main link.

`c.set_pose(tfm)` sets the orientation and position of the main link.

`c.set_pose(theta, vec)` also sets the orientation and position of the main link. Here the rotation and translation are given separately, instead of as a rotra.

`c.x()` returns the x-component of the translation part of the main link.

`c.set_x(x)` sets the x-component of the translation part of the main link. The y-part and other parameters are not influenced.

`c.y()` returns the y-component of the translation part of the main link.

`c.set_y(y)` sets the y-component of the translation part of the main link. The x-part and other parameters are not influenced.

`c.get_angle()` returns the rotation part of the transformation of the main link.

`c.set_angle(angle)` sets the rotation part of the transformation of the main link.

`c.get_joint(jointno)` returns the value for joint `jointno`.

`c.set_joint(jointno, x)` sets the value for joint `jointno` to `x`.

3.3 Drawing

Synopsis

```
void draw(pl_drawarea &drawing, const pl_geolist &gl)
```

Description

RS2 supplies a routine to draw a geometrylist in a drawarea. As is explained in the PlaGeo documentation, this means that this routine can be used to draw a geolist in every renderer that is derived from this class, for example in a postscript file or a GL window.

Robots can be drawn quite easily in any configuration. For example, figure 3.1 was generated with the help of the following program:

```
main()
{
    pl_psfile psarea(cout, 0, 0, 400, 300);
    pl_robot rob2;
    pl_configuration conf(2);
    rob2 = construct2();
    psarea.set_viewingarea(-5, -5, 40, 30);
    draw(psarea, rob2.cur_shape());
    conf.setx(10);
    conf.setjoint(0, -M_PI/4);
    rob2.place(conf);
    draw(psarea, rob2.cur_shape());
```

```
    pl_configuration conf(rob.jointcount());
    rob.place_non_checked(conf);            // force rob in zero position
    for (i=0; i<=rob.jointcount(); i++) {   //  because #links = #joints+1
        result.replace_link(i, rob.cur_link_shape(i));
    }
    return result;
}
```

3.2 Robot Configurations

Synopsis

```
class pl_configuration {
                    pl_configuration(int i)
                    pl_configuration()
    int             jointcount() const
    pl_rotra        get_pose()  const
    void            set_pose(const pl_rotra &tfm)
    void            set_pose(pl_angle theta, const pl_vec &vec)
    s_coord         x() const
    void            set_x(s_coord x)
    s_coord         y() const
    void            set_y(s_coord y)
    s_coord         get_angle() const
    void            set_angle(s_coord angle)
    s_coord         get_joint(int jointno) const
    void            set_joint(int jointno, s_coord pos)
}
```

Description

The type pl_configuration is used to represent configurations of robots. The configuration of a robot can be described by the position and orientation of the main (first) part and the position of the joints of the robot. Therefore a configuration consists of a pl_rotra and a number of joint values.

Constructors

The first constructor takes one parameter, the number of joints for this configuration. All values are initialized to zero, i.e. the main link is rotated nor translated and all joints have value zero.

The second constructor constructs a configuration with zero joints. Again, all values are initialised to zero.

Class Functions

`c.jointcount()` gives the number of jointvalues that `c` has.

```
    gl.pl.newsize(1);
    gl.pl[0] = pl_polygon(vl);
    robot.replace_link(0, gl);
// define geometry of second link
    vl.newsize(4);
    vl[0] = pl_vertex(1, 9);
    vl[1] = pl_vertex(1, 15);
    vl[2] = pl_vertex(-1, 15);
    vl[3] = pl_vertex(-1, 9);
    gl.pl.newsize(1);    // not really necessary, but it does no harm
    gl.pl[0] = pl_polygon(vl);
    robot.replace_link(1, gl);
// define geometry of third link
    gl.pl.newsize(0);
    gl.el.newsize(1);
    gl.el[0] = pl_edge(pl_vertex(0, 10), pl_vertex(0, 20));
    robot.replace_link(2, gl);
    return robot;
}
```

The following program reconstructs a robot using low level routines. Of course, the routine as such is ridiculous: it is a very complicated way to do an assignment. Nevertheless, it demonstrates how the low level routines can be used. Code like this would be needed to write a robot to file and then read it in again.

```
pl_robot reconstruction(pl_robot rob)
{
    s_coordreray jointdescription;
    int i;
    for (i=0; i<rob.jointcount(); i++) {
        if (rob.jointtype(i) == PRISMATIC) {
            pl_vec trans;
            rob.translate_params(i, trans);
            jointdescription.append(PRISMATIC);
            jointdescription.append(trans.x());
            jointdescription.append(trans.y());
        } else {
            pl_coord center;
            s_coord min_angle, max_angle;
            rob.rotate_params(i, center, min_angle, max_angle);
            jointdescription.append(REVOLUTE);
            jointdescription.append(center.x());
            jointdescription.append(center.y());
            jointdescription.append(min_angle);
            jointdescription.append(max_angle);
        }
    }
    pl_robot result(rob.jointcount(), jointdescription);
```

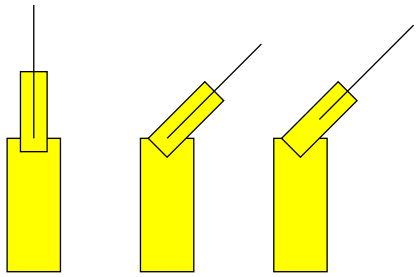

Figure 3.1: The robot in three positions

Examples

First we give an example of the definition of two robots. The first has no links, the second is a bit more complicated.

```
pl_robot construct_simple()
{
    pl_robot robot;
    pl_vertexreray vl(3);
    vl[0] = pl_vertex(10, 0);
    vl[1] = pl_vertex(-5, 3);
    vl[2] = pl_vertex(-5, -3);
    pl_geolist gl;
    gl.pl.append(pl_polygon(vl));
    robot.replace_link(0, gl);
    return robot;
}

pl_robot construct2()
{
    pl_geolist gl;
    pl_vertexreray vl;
    static float jd[] = { REVOLUTE, 0, 10, -M_PI/2, 0,
                          PRISMATIC, 0, 2 };
    pl_robot robot(2, 8, jd);
// define geometry of first link
    vl.newsize(4);
    vl[0] = pl_vertex(2, 0);
    vl[1] = pl_vertex(2, 10);
    vl[2] = pl_vertex(-2, 10);
    vl[3] = pl_vertex(-2, 0);
```

joint can take. Those angles are given in radians.

The third constructor is basically the same as the second, only the reray has been replaced by an array and a parameter indicating the length of the array.

The constructors check whether the joint description list is consistent, that is, whether enough joints are specified and whether all values in the list are valid. After those constructors are applied, the class variable `ok` can be inspected to see if the list was all right.

At the end of this section some examples are given where robots are defined.

Class Functions

`c.replace_link(i, gl)` replaces the `i`-th link of the robot with the geolist `gl`. The links of a robot are numbered from 0 upwards. The number of links is always one bigger than the number of joints. The geometry in the geometrylist describes the geometry of the link when the robot is in a configuration with all parameters equal to zero.[1] Normally, the links are replaced immediately after the construction of the robot, but it can be done anytime.

`c.place(conf)` places `c` in configuration `conf`, taking into account the maximum and minimum value that joints can take. Configurations are described in section 3.2. For revolute joints, the extremal values are specified during construction, for prismatic joints the extremal values are always 0 and 1 (no and maximal displacement). If a value is too big or too small, the value is clipped to the extremum. The function returns how many values had to be adapted.

`c.place_non_checked(conf)` places `c` in configuration `conf`. The configuration is not checked, so the robot may in this way be placed in an illegal position according to the extremal values.

`c.curconf()` returns the current configuration of `c`.

`c.cur_shape()` returns the shape of the robot in the current configuration.

`c.cur_link_shape(linkno)` returns the shape of link `linkno` in the current configuration.

`c.link_rotra(int linkno)` returns the transformation by which link `linkno` is transformed in the current configuration, i.e. the rotation and translation needed to move that link from its original position to its current position.

`c.jointcount()` returns the number of joints of `c`.

`c.jointtype(int jointno)` returns whether joint `jointno` is prismatic or revolute.

`c.rotate_params(jointno, center, min_angle, max_angle)` puts the parameters of joint `jointno` in the variables center, min_angle and max_angle. This routine supposes that this joint is indeed a revolute joint.

`c.translate_params(int jointno, max_trans)` puts the maximal displacement parameters of a prismatic joint in the variable.

[1] This does not mean, for revolute joints, that zero has to be in the range of allowed angles.

```
class pl_robot {
                    pl_robot()
                    pl_robot(int jnum, const s_coordreray &jd)
                    pl_robot(int jnum, int length, float *jd)
    static int      ok
    void            replace_link(int, const pl_geolist &)
    int             place(const pl_configuration &)
    void            place_non_checked(const pl_configuration &)
    pl_configuration curconf() const
    pl_geolist      cur_shape() const
    pl_geolist      cur_link_shape(int linkno) const
    pl_rotra        link_rotra(int linkno) const
    int             jointcount()  const
    int             jointtype(int jointno) const
    int             rotate_params(int jointno, pl_coord &center,
                          s_coord &min_angle, s_coord &max_angle) const
    int             translate_params(int jointno, pl_vec &max_trans) const
};
```

Description

The type pl_robot is used to describe moving bodies in the plane. A robot consists of a number of rigid bodies, called links, connected in a chainlike manner by joints. The joints come in two types: revolute and prismatic joints. The first type allows a link (and the other links attached to this link) to rotate with respect to the previous link, the second type allows a link to slide with respect to the previous link. Every link can be made out of points, edges and polygons.

The actual geometry of a robot depends on its configuration, that is, the placement of its first link in space and the position of the joints. When the geometry of a robot is specified, this is always done with the robot in its neutral configuration: the first link is placed, unrotated, in the origin and all the joints have a zero value.

Constructors

Robots can be quite complicated. In order not to make the constructors too complicated, only the structure of a robot is described by a constructor, not the geometry. This means that originally all links are empty. The geometry of links can be specified later with the routine `replace_link`.

The first constructor is simple: it makes a robot that has only one link and no joints.

The second expects the number of joints (`jnum`) and a reray of `s_coord`s (`jd`) that describes the joints. For each joint there are a number of numbers in the reray. The first describes the type of the joint. This number must be either `PRISMATIC` or `REVOLUTE`, two constants declared in the header file. When the type is prismatic, two other numbers follow. Those numbers indicate how far the joint can move in the x- and y- direction. When the type is revolute, four numbers follow. The first two of these give the center of rotation (x- and y-coordinate). Next follow the minimum and maximum angle that the

Chapter 3

Representation of Robots

This section describes how RS2 supports the modelling of robots. The parts of robots may contain points, edges and polygons. A robot can be made of different parts, called links, joined by prismatic or revolute joints. The arrangement of the joints and links must be that of a chain (or list), each link having at most one predecessor and successor.

A robot has a configuration which describes the position and orientation of the main link in space and the position of all joints.

3.1 Robots

Geometry Lists

First we describe a type that can hold vertices, edges and polygons, pl_geolist. This type is used for the representation of the links of a robot.

```
class pl_geolist {
    pl_vertexreray  vl;
    pl_edgereray    el;
    pl_polygonreray pl;
    void            transform(const pl_vec &)
    void            transform(const pl_rotra &tfm)
};
```

Description

A pl_geolist contains three rerays, one for the vertices, one for the edges and one for the polygons. Those rerays can be accessed just like a struct member.

The `transform` class functions transform all items of the three rerays according to the transformation.

Robots

Synopsis

```
const int PRISMATIC;
const int REVOLUTE;
```

Chapter 2

Use

The RS2 library is based on PlaGeo. In this document it is assumed that the reader is familiar with that package. In the documentation of PlaGeo is also a section about the use of C++, which explains some aspects of that language that are indispensable when one wants to use libraries implemented in C++. The user of the RS2 library needs no thorough knowledge of C++. Because C is a subset of C++, the user can write an ordinary C-program, using only some C++-syntax for calling library routines. This program can be translated by the C++ compiler. This documentation supposes the reader has some knowledge of C, not of C++.

2.1 Compilation

Programs using the RS2 library should be compiled with a C++ compiler that supports templates. Suppose that a source file `file.C` uses RS2 routines. In that case `file.C` should contain the line

```
#include <rs2.h>
```

This include-file can be found in the directory `/usr/staff/include`. The library itself resides in `/usr/staff/lib`.[1] Compiling and linking should be done as follows:

```
CC -I/usr/staff/include -c file.C
CC file.o -I/usr/staff/include -L/usr/staff/lib -lrs2 -lplageo -lm
```

If the Silicon Graphics drawing facilities are used, the last line should be:

```
CC file.o -I/usr/staff/include -L/usr/staff/lib -lrs2 -lplageo -lgl_s -lm
```

[1] The location of the files describes the situation at Utrecht University.

Chapter 1

Introduction

The Robot Support Library for Two Dimensional Robots, RS2 for short, is meant to provide assistance for the implementation of motion planning algorithms for planar robots. It is based on PlaGeo, a library for basic geometric computations in the plane. RS2 itself is used as a basis for the MotView library, a library for visualising robot motions.

The RS2 library offers a way to represent robots and configurations (or placements) of robots. Robots consist of vertices, line segments and polygons, possibly connected by means of revolute or prismatic joints. A configurations describes the position of a robot and of its joints.

Furthermore there are routines to draw robots and to write and read robots and configurations to or from file.

The RS2 library, as well as PlaGeo and MotView, can be obtained with anonymous ftp from the ftp-server `archive.cs.ruu.nl` below directory pub/SGI.

Bugs, Extensions, etc.

Although RS2 has gone through extensive testing some bugs may still be present. Your comments would be greatly appreciated. Please report any bug reports or comments to me using the email address **geert@cs.ruu.nl**. Also, if you make additions that could be of more general interest, please mail them to me so that I might add them to a future release.

Notice

This software is public domain. It may freely be copied and used in non-commercial products, assuming proper credit is given to the author, but it should not be resold. If you want to use the software for commercial products, contact the author.

This software is provided "as is" without warranty of any kind, either expressed or implied. The entire risk as to the quality and performance of the software is with you. Should the software prove defective, you assume the cost of all necessary servicing, repair or correction.

Acknowledgements

The work on this package was partially supported by the Dutch Organization for Scientific Research (N.W.O) and by the ESPRIT III BRA Project 6546 (PROMotion).

Contents

1 Introduction **2**

2 Use **3**
2.1 Compilation 3

3 Representation of Robots **4**
3.1 Robots 4
3.2 Robot Configurations 9
3.3 Drawing 10
3.4 File IO 11

RS2

A Support Library for Robot Geometry
Based on PlaGeo

version 1.4

Geert-Jan Giezeman