Universität zu Lübeck

Institute for Robotics and Cognitive Systems

Master Thesis

An approach to solving object displacement problems

written by

Maximilian Mühlfeld (580070)

Supervisor:

Prof. Dr.-Ing. Achim Schweikard

Lübeck, June 16, 2014

Assertion

I assure that the following work is done independently with the use of only stated resources.

Lübeck, June 16, 2014

Kurzfassung

The scope of this thesis is the implementation and testing of algoritms to solve object displacement problems.
To achieve this the search space is divided into multiple cells to reduce its size. Furthermore instead of calculating the whole space at once, a local approach is used to only calculate the parts needed for the next search step. This needs a wide range of basic geometric and algebraic algorithms for object representation, collision detection and object translation/rotation.
On the resulting search space a simple and exchangable graph search algorithm is applied, which will give us a path in the object space from start configuration to the desired target configuration.

Abstract

The scope of this thesis is the implementation and testing of algoritms to solve object displacement problems.
To achieve this the search space is divided into multiple cells to reduce its size. Furthermore instead of calculating the whole space at once, a local approach is used to only calculate the parts needed for the next search step. This needs a wide range of basic geometric and algebraic algorithms for object representation, collision detection and object translation/rotation.
On the resulting search space a simple and exchangable graph search algorithm is applied, which will give us a path in the object space from start configuration to the desired target configuration.

Contents

1 Introduction

1.1 Motivation

1.1.1 Path planning in robotics

In robotics the path planning of a robot can be programmed as a fixed list of movements to be executed. This has multiple drawbacks as for every change in the enviroment the robot needs to manually be reprogrammed.
But what if we equip this robot with a camera that detects the shape of objects in the robots enviroment. This would give us a set of objects at certain position, a robot arm in a starting position and a target where the robots endeffector needs to work. If we would be able to solve this puzzle, we could direct the robot in a different way each time without the actual need to access its software.

1.1.2 Geometric riddles in gaming

Solving geometric riddles is a amazingly fun task for a human. This is the reason many games simply consists of such riddles ranking from easy to hard in difficulty. But even the easiest riddle for a human proposes a big challenge for a simple algorithm that searches trough the possible ways of solving it. Even more complicated is the generation of such riddles. Even for the human brain this task can be exceedingly stressful.
Now, if there would be a possibility to check if such a riddle has a solution, there would be the option to generate them randomly and check for feasability. This would release the developers of the need to manually create each riddle. Also the consumers, in this case the players, would have a never ending stream of new and different riddles to solve.

1.2 Idea

As the representation behind these two cases is the same, for simplification purposes we break the riddles down into two dimensional object displacement problems. The following riddles shall provide simple example of the problem:

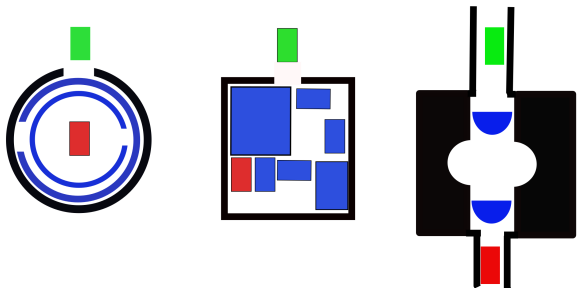

Figure 1.1: *riddle 1 with circles , riddle 2 with boxes , riddle 3 with 2 half-circles*

The riddle is solved, if the red box matches the green one. Blue objects are movable and black ones are stationary. The stationary ones will be referred to as rims hereafter.
As a human the way to solve those is quite obvious. The first riddle is solved by rotating the blue objects, the second through translation. The third needs both ways to be solved.
If we want to solve this with an algorithm, we would need to consider each objects collsision with the other objects and the rim. Also we would need to find a way to express the current configuration consisting of position (x,y) and rotation (ϕ) and the direction the main object needs to take. A simple idea would be to

1. generate all valid configurations per object in regard to the stationary obstacles as a configuration space
2. Create one collision space per object for collision with every other object.
3. Substract the collision space from the configuration space to get valid space in regard to all obstacles for one object.
4. Divide the space in cells and locate the valid ones.
5. Build a graph out of this starting position by adapting the space after each step.
6. Search in that graph to get path from start to target point.

The target point would then be a simple configuration vector holding each position of each

object. A user defined distance function beetween start and target point in that space can then be used as heuristic in the graph search (e.g. $h(start, target) = ||(x_{start} - x_{target}, y_{start} - y_{target})||$).

2 Enviroment and basics

2.1 Programming tools

As the target of this thesis is a mere proof of concept, the programming language of choice is matlab [2]. This is a reasobable decision because in the matlab language a huge part of the needed functionality concerning simple mathematical functions is already implemented and easy to use.
Pros:

- easy to use mathematical function
- fast and easy way to enter data
- good and simple ways of debugging and locating errors

Cons:

- slow computation speed
- needs translation in other language (e.g. c++) for further use

As versioning tool git [1] is used together with www.github.com as an open source storage plattform for the resulting code.

2.2 Basics

For better understanding of the following chapters the central mathematical formulas are repeated here.

2.2.1 basic vector math

If the objects are represented as a list of corner points, vector math comes in handy when describing the borders of the object. Lets say we have a simple object A defined by the following list:

$$A = (0, 0; 2, 0; 2, 2; 0, 2)$$

Each pair x,y defines one corner of A and the points are ordered to travel along the border of A counterclockwise. The lines connecting these points will be called the borders of A. They are calculated by substracting the points from each other such that the vectors point along the border clockwise. This leads to:

$$A_{vec} = (2, 0; 0, 2; -2, 0; 0, -2)$$

Next will be the calculation of the angle beetween two vectors a and b. This is done by taking a reference vector , for example the x-axis r=$(1, 0)$, calculating the angle to that and building the difference in angle. This way we can determine an angle beetween the vectors a and b where the sign of the angle tells us which vector lies "below" the other.
The formula for this is:

$$\phi = anglebeetweenvectors formula$$

2.2.2 convex hull

Next is the calculation of the convex hull of an object for another object. Lets take object A from before and another object B defined by:

$$B = (0, 0; 1, 0; 1, 2; 0, 2)$$

Furthermore we define the center of these objects to be $M_A = (1, 1)$ and $M_B = (0.5, 1)$ The convex hull for A to B is calculated after the formula

$$hull = formula$$

Basically this means we add the points of A shifted with $-M_A$ and mirrored at M_A to each point of B shifted with $-M_B$ and then reshifted with M_B to get a list C_{temp} of $4 \cdot 4 = 16$ points.

$$C_{temp} = \begin{matrix} (& -1, & -1; & 1, & -1; & 1, & 1; & -1, & 1; \\ & 0, & -1; & 2, & -1; & 2, & 1; & 0, & 1; \\ & 0, & 1; & 2, & 1; & 2, & 3; & 0, & 3; \\ & -1, & 1; & 1, & 1; & 1, & 3; & -1, & 3) \end{matrix}$$

Each line represents one point of B with the points of A added. By choosing the outer points from C_{temp} and put them in C, such that no point from C_{temp} is still outside C we will get an object C which defines the space around B which the object A can not enter or they will collide.

$$C = (-1, -1; 2, -1; 2, 3; -1, 3)$$

2.2.3 linear programming

Following the convex hull, as a way to actually compute if a collision took place or not, we use linear programming. For that we represent the convex hull of object A to object B $Hull_{A.B}$ with its vector representation.
Now we take the center of A and check if its inside the convex hull.

$$A(1,:) + A_{vec}(1,:)$$

List of Figures

List of Tables

bibliography

[1] Chacon, S. *Pro Git*, July 2009.

[2] MathWorks, Inc. *Matlab Language Reference Manual*.