3D Robotic Engine

Stephan Jud
me@steve.ch

Giuseppe Accaputo
ga@giu.me

Rapperswil, June 10, 2009

Abstract

In the field of robotics an efficient and fast planning of collision free routes is essential to control parallel movements.
This document discusses algorithms and strategies for moving liquid handling robots both efficiently and safe in a three dimensional space. An approach for collision detection and resolution in real time is presented as well as a path finding mechanism to traverse typical liquid handling platforms. Additionally, an optimization to achieve higher traveling speeds for robots is introduced.

Acknowledgments

Without the help and support of following people this Bachelor Thesis would not have been possible. We would like to thank...

Joas Leemann ... for many insightful discussions, giving valuable feedback and his time and patience

Hansjörg Huser ... for accepting this Bachelor Thesis and for his help

Louis-Sepp Willimann ... for his time and extraordinary useful hints about splines

Rainer Kerkmann ... for accepting the Thesis and provide office space

Our families ... for the support and the good food

Our girlfriends ... for love and for not asking questions why we were working until late at night

Contents

Part I

Management Summary

Initial Situation

Tecan Schweiz AG produces liquid handling platforms used for medicinal and biological laboratory tests. Such a liquid handling platform can be configured with multiple robotic arms and equipment. Using the full capabilities provided by the platform is essential to minimize idle time and optimize throughput. However, the existing solution shows room for improvement.

The goal of this Bachelor Thesis was to define new controlling algorithms for the liquid handling platform which are able to plan and execute parallel motion sequences.

The defined component had to support any type of robotic arms but still take advantage of their intrinsic properties. Additionally, the integration of the component into the current control software of Tecan Schweiz AG had to be possible with little effort.

The project was started by an individual initiative of the team. It has been chosen as Bachelor Thesis because of the interesting domain and the foreseeable required investigations.

Procedure

During the project the Rational Unified Process (RUP) has been used successfully to organize and plan iterations which were partitioned to two weeks each:

Inception

Start	End	Iteration
16.02.09	27.02.09	Inception 1

At the beginning of the Inception phase the required infrastructure like Subversion server and Trac environment has been set up, enabling a solid and convenient working environment for the team. Additionally, a first version of the Software Development Plan describing the iterations in detail has been created. Furthermore, meetings were held with stakeholders to gather information about the different objectives.

Elaboration

Start	End	Iteration
02.03.09	13.03.09	Elaboration 1
16.03.09	27.03.09	Elaboration 2
30.03.09	10.04.09	Elaboration 3
13.04.09	24.04.09	Elaboration 4

During the Elaboration phase relevant algorithms were evaluated regarding the target system. Because it turned out that there were no adequate ones, the development of an environment specific algorithm which is able to meet the demands was initiated.

Soon a first simple test environment was created with a simulator and a textual interface to the developed algorithm. This led to a draft implementation which already included an initial realization of the proposed algorithm.

During this phase another requirement for the algorithm emerged: Computed paths should have smooth corners which would enable high traveling speeds. As a result, different strategies were evaluated which led to the development of an approach using natural cubic splines with the designed algorithms.

Construction

Start	End	Iteration
27.04.09	08.05.09	Construction 1
11.05.09	22.05.09	Construction 2
25.05.09	05.06.09	Construction 3

The Construction phase mainly consisted of programming and testing tasks. Thanks to numerous test cases which have already been defined, a big number of improvements

and enhancements could be realized especially regarding path finding, collision avoidance and resolution.

Transition

Start	End	Iteration
08.06.09	12.06.09	Transition 1

Finally, during the Transition phase the code was extended to successfully pass existing test. Furthermore, all documents have been reviewed and discussed with stakeholders.

Project Management

An agile version of RUP has been successfully applied throughout the project. Since a big part of the Bachelor Thesis consisted mainly of research tasks, investing a high amount of time in evaluation and research in early stages paid itself off. Additionally, keeping track of the progress with Iteration Assessments turned out to be a good instrument to keep all stakeholders up to date.

During the project maintaining an up-to-date state of all the documents was emphasized. This was achieved with recurring documentation reviews.

The Bachelor Thesis has shown that a good and flexible project planning is very important for the succeeding of a project. Using RUP for this project was definitely the right decision.

A maximum of 360 work hours per person (720 work hours total) were available for the project. A total of 726 hours were planned, whereof 33 hours were reserved as a buffer if work packages took more time than expected. A total of 982 hours were invested into the realization. The surplus of 256 hours resulted because the team realized that a lot of work was required to complete certain tasks but decided anyway that it was worth it.

Meetings were held almost every week, either with Joas Leemann or Hansjörg Huser. The purpose of these meetings was to discuss the current project state, work packages and the created artifacts of the corresponding iteration.

People

Hansjörg Huser is the adviser in authority of the Bachelor Thesis. He ensured that formal aspects were respected and helped improving the documentation thanks to his feedback.

Joas Leemann is the adviser provided by the Tecan Schweiz AG and helped the team with many discussions, answers to technical questions and gave lots of feedback to various presented approaches and their implementation.

Rainer Kerkmann is a project manager at Tecan Schweiz AG and was responsible for the approval of the project the team proposed. He also was responsible for the setup of the workspaces provided for the team in offices of Tecan Schweiz AG in Männedorf.

Results

Due to the complexity of the problem it was not possible to find an algorithm that could fulfill the posed requirements. Therefore a multilevel proceeding has been defined that uses an incremental approach to solve the given tasks:

Figure 1: Multilevel proceeding

In a first step the possible conflict states are determined. Then, the shortest collision free path is calculated. Finally, the path is optimized for the robot. This reduction of complexity and size turned out to be essential to achieve fast calculation times.

To ensure smooth path guidance the generated waypoints are interpolated using a natural cubic spline. This enables the robot to follow the computed route with a constantly high velocity.

The developed approaches have been implemented in a prototype and proved themselves to be a realistic and efficient realization to control various types of robotic arms in a liquid handling platform. The actual collision avoidance logic, excluding input validation and data classes could be realized in just 360 lines of code. Counting the lines of all created software results in 3300 lines of code.

Additionally, various tools have been developed to visualize the implemented algorithms. They have been implemented using either .NET 3.5, WPF and Visual Studio 2008 or JavaScript, HTML and CSS.

This means the goals could be achieved. A set of tools is available to support further development of the current implementation. Additionally, a few interesting approaches have been figured out and described so they can be implemented by Tecan Schweiz AG at a later time.

The implemented solution fulfills the real-time requirements of Tecan Schweiz AG.

Parsing	**4** µs/request
Collision Detection	**3** µs/devices
Collision Resolution	**0.8** µs/obstacle
Splining	**3** µs/waypoints
Dispatching	**n/a**

Figure 2: Throughput measurements with five robots in the environment

Perspective

The solution is advanced to the extent so that Tecan Schweiz AG can use it as a foundation for further developments. An integration into the existing framework could be achieved easily.

However, not all proposed enhancements could be implemented. They have been extensively described though and might be realized at a later date.

Part II

Technical Report

Chapter 1

Introduction

1.1 Environment

Tecan Schweiz AG produces liquid handling platforms which can be combined with a wide range of robotic arms. The platform is actively managed by an application which runs on a connected personal computer. The program controls the platform by transmitting control commands to the corresponding devices which then apply the movement.

A reliable and efficient way to control the interaction of such robotic devices is essential to fully utilize the system's mechanical capabilities. Furthermore, minimizing idle time and duration of robotic operations are important aspects to increase overall system throughput.

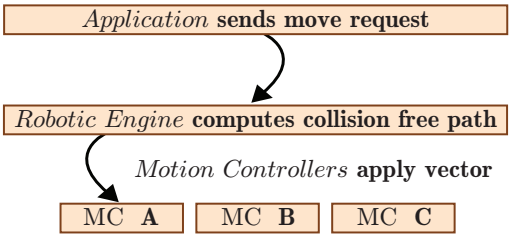
For more information about the project this thesis is written including the exact conceptual formulation, refer to the Software Development Plan on page 139.

1.2 Goal

This Bachelor Thesis shall define the design of a software component which is able to compute and perform moves of robotic devices inside the target system based on move requests received from other components.

This includes a collision avoidance mechanism which automatically resolves collision situations should they be detected. Additionally, the system must comply with performance characteristics given due to limited space and time resources.

The said component shall be designed and realized in a generic way where supported devices do not have path planning logic implemented on their own. Its boundaries are defined as shown below.



The thesis is split into the following three parts:

Research A path finding algorithm has to be chosen and extended so it matches the requirements. Additionally, an efficient collision detection and resolution algorithm has to be developed for the target system which should perform efficiently in a three dimensional space.

Design Once all required algorithms have been defined, a generic engine which fits into the existing software framework shall be designed. It must be device independent and provide a uniform interface to upper layers.

Implementation A move engine shall be implemented according to the previously developed design.

1.3 Scope

The designed system can assume correct behavior of actions performed on the system as well as the correctness of physical properties defined in the device specifications. It must reach prototype quality, demonstrating feasibility and correctness of the developed approaches.

Details regarding exact technical realization of Tecan internal components are omitted in justifications.

Chapter 2

Collision Avoidance

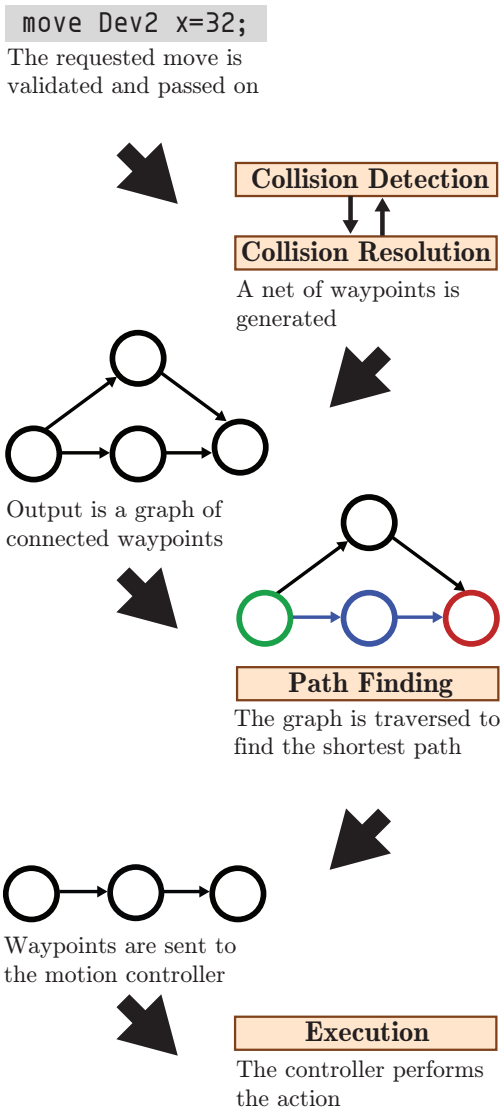
This chapter handles the first of three fragments of the Thesis. It represents the research part. Algorithms and approaches regarding collision avoidance are described and evaluated.

Collision avoidance interprets move requests and calculates a collision free path for the corresponding object. Because the problem of collision avoidance cannot be solved by just one algorithm or strategy[1], it must be split into sub-problems, each requiring a different approach.

Three such sub-problems have been identified and throughout this chapter following nomenclature is used for them.

> *Path Finding* is used to traverse through an environment, making use of *Collision Detection* to determine which ways are clear to move through and finally using *Collision Resolution* to pass through complex regions where interaction with other objects is required.

The main approach is to provide a graph of interconnected abstract waypoints generated according to the physical capabilities of involved components and current environmental situation. After such a graph has been created, it is traversed to determine whether the target position can be reached and if so, which route leads to the least time consumption.



[1] For more information refer to chapter 4

2.1 System Properties

The Bachelor Thesis targets a very specific platform. Highlighting its restrictions and main properties allows the development of a specialized algorithm which performs more efficiently than a generic one due to a more limited solution space. Relevant properties for collision avoidance on liquid handling platforms are listed below.

- All objects in the system are known at runtime.
- All movable objects in the system can be controlled by software.
- When an unmovable object changes its location and/or size, the system will receive a notification about this event.
- Interactions may occur at unforeseeable moments initiated by external actors. However, this kind of events is rather rare.
- The number of independent moveable objects is limited to ten per environment.
- Moveable objects are built in a cubicle shape and do not have round surfaces.
- The interaction space is inside a cube. There may exist interaction points at system boundaries for interoperability with other platforms though.
- The components inside the environment are normally placed close to each other, to minimize traveling time.
- The workflow depends on many exterior influences and is not deterministic. Thus, movements cannot be precomputed.
- There can be an arbitrary number of robot types on the system.
- Moveable objects have following properties
  - Objects are mounted on tracks
  - Objects do collide when mounted on the same track and residing at the same position
  - Axes x,y and z define their range
  - Axis ranges have no gaps
  - Move properties are equal on every point on the axis
  - Objects can move independently and concurrently on every supported axis
  - Objects can have subobjects
  - The range of subobjects is relative to their parents'

2.2 Path Finding

There are various ways to compute routes through a system avoiding obstacles. Relevant algorithms are listed below. For a detailed description refer to chapter 4.

Djikstra's Algorithm A graph traversal algorithm which searches the shortest path between two nodes in a graph.

A* Search A best-first graph search algorithm that finds the path with least cost to a target. It uses heuristics to move through the environment.

Iterative Deepening A* A variant of A* which uses less memory because it does not use lists to keep track of visited nodes.

Fringe Search A search algorithm based on Iterative deepening A* (IDA*) which eliminates some disadvantages and usually performs faster because of an incremental approach.

Potential Fields All objects are expanded with a force comparable to magnetic ones. The device is attracted by the target's force. During the movement it avoids obstacles by being repulsed by them.

2.2.1 3D Path Finding

The listed path finding algorithms perform in a two dimensional space containing both obstacles and free cells or nodes.

As mentioned in section 2.1, path finding must be performed in a three dimensional space. This requirement drastically increases the complexity of the problem as an order of magnitude more calculations are required. Straightly applying the path finding algorithms is not feasible anymore because of the sheer amount of cells this would create.

Additionally, split the environment into cells raises another problem: How should the cell dimensions be determined? If they are too small, many unnecessary checks are required for objects although they could be represented in bigger cells more efficiently. Also, the algorithm is required to maintain a bigger stack of decisions to turn back once a dead end has been detected. If, however, the cell sizes are too big, small items might slip through collision detection, possibly causing collisions.

Binary Cell Splitting

One possible solution is to dynamically adjust the cell size depending on the obstacles inside the area (Figure 2.1). The path algorithm must be extended to take those different sizes into account though.

But also with this approach problems arise when applied to a three dimensional space. Due to the characteristics of the target environment which contains many objects with different sizes, a large amount of cells gets generated which leads to many unnecessarily calculated paths.

Additionally, cell dimensions need to be re-adjusted when an obstacle moves into areas with different cell dimensions. This requires re-dimensioning cells while the search is being performed.

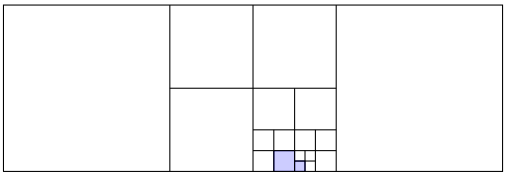


Figure 2.1: Cells are divided until the obstacle (blue) can be represented within the map in the desired precision. The resulting matrix will be used by the path finding algorithm to calculate the shortest path.

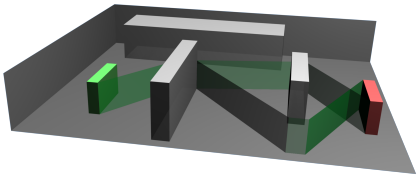


Figure 2.2: Scene with connected waypoints between origin (green) and destination (red).

Obstacle Based Splitting

Another approach is to build a logical map of possible waypoints (Figure 2.2).

By keeping waypoints abstract, the traveled space can be reduced, requiring fewer computations.

Waypoints can be placed at positions the device is likely to pass (between start and target position), at turning points of obstacles. To move a device the required translations between the elements of the optimal waypoint-chain can be calculated and be applied.

Obviously, this results in removing paths from being detected by the path finding algorithm. However, because of the drastically reduced number of calculations the algorithm has to perform, this is preferable as omitted routes can be added in further search iterations.

This approach implies the actual move operations are hidden from the algorithm which traverses the logical map. A “hop count” cost function would lead to wrong results. To correct this, the edges must rep-

resent the cost of the underlying operation and the algorithm must take those different edge costs into account.

Finding the shortest path in that map leads automatically to the shortest travel time for the moving object in the target environment.

Due to this reduction of potential scenarios which need to be evaluated by the path finding algorithm, expanding the solution space to three dimensions does not result in exponentially more computations.

Map Creation Waypoints represent position and time at which a device can pass an obstacle. The time aspect must be included as solely generating waypoints based on the obstacles in the environment would not include dynamic behavior.

An initial map can be calculated from the start to the end position. If it turns out that there is no collision free path, further waypoints must be added to the map in order to avoid the obstacle.

However, obstacles can change their sizes or positions while the device is moving. This dynamic behavior must be taken into account as it influences which waypoints are available.

Thus, waypoints must be generated iteratively keeping track of the time passed since the move started because the distance and even the existence of connected waypoints may change during run time. This means the map must be built up lazily.

As edges already represent the time consumed, the current time can be represented as accumulated costs of already traversed graphs.

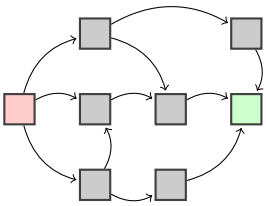
Map States Scenarios which might be encountered by the algorithm while traveling through such a map are listed below[2].

1. When no nodes are connected, the target position cannot be reached:

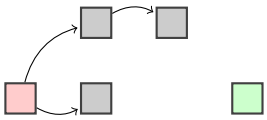
2. When the start node equals the target position, no move is required:

3. If the device can reach its target position by moving straight in its direction without any modifications of its dimensional properties, the algorithm can return a simple graph whose only edge represents all costs of the move:

4. The device cannot move straight to its target position. With help of Collision Resolution the algorithm builds up a map containing logical positions the device can reach. If start and end point are connected in any way, the device can reach its target. However, the costs represented by edges must be considered to ensure the fastest route is chosen:



5. If the start and end point cannot be connected together it means that the target position is not reachable:



While appending graphs to nodes within the map, obstacle for obstacle has to be resolved. This greedy approach forces the algorithm to keep track of decisions made in

[2] Red squares indicate the current, green ones the target position. Gray squares represent nodes which may be traversed

order to resolve dead ends. Also, the consumed time must be maintained as it influences which graphs are added or removed to a node.

A map traversal example can be found in appendix A.1.1 on page 205.

The described approach frees the algorithm from handling dynamic obstacle sizes and tracking other moving objects. Thus, it can be implemented independently, increasing modularity and separation of concerns. The remaining problems are delegated to other components - Collision Detection and Collision Resolution - which will be described later in this chapter.

2.2.2 Conclusion

The path finding algorithms mentioned do all work in a completely unknown environment. Because the main characteristics of the environment in which the robotic arms will interact are fairly clear, some general assumptions can be made reducing path finding computation. Thus, algorithms which take environmental heuristics into account will achieve better results and additionally, can be fine tuned externally.

However, the algorithms listed cannot be applied straightly to the problem because of the four-dimensional solution space. A preprocessing component is required which reduces the collision avoidance problem to an algorithm compatible format. Creating a map of abstract waypoints is the most efficient way to achieve this.

2.3 Collision Detection

Collision Detection is the first essential component for waypoint creation. Given speed, acceleration and direction of a moving object, it determines when it will collide with other moving devices or obstacles.

All active and passive components[3] in the system are known. Therefore, it is possible to predict when collisions will occur.

An efficient way to detect a collision between multiple objects is already available [6]. This section discusses how that algorithm can be applied to the target environment.

2.3.1 Sweep Prune

Basically, Sweep Prune[6] reduces the collision detection problem to a sub-problem for every axis involved. Those sub-problems must be solved independently. The result of a collision check is the sum of all those sub-problems.

For the target environment (section 2.1) all three Cartesian axes x, y, z are relevant for collision detection. Applied to Sweep Prune this means three lists, each describing which parts of an axis are occupied by devices[4] are required.

The sum of boundaries (**B**egin and **E**nd of axis allocation) for every device d on $\mathbf{axis}_a$ must be stored in a list describing the axis' occupation. Begin and end are calculated according to the direction in which the collision detection is performed.

$$\mathbf{alloc_a} = \begin{pmatrix} d1_{aB} & d1_{aE} \\ d2_{aB} & d2_{aE} \\ \vdots & \vdots \\ dn_{aB} & dn_{aE} \end{pmatrix}$$

A collision free move can now be defined as a change of the moving objects in the allocation lists where all other objects stay in the same relative direction of the moving object.

A special case occurs when the direction cannot be determined, i.e. the checked object resides at exactly the same position like the moving one. If so, it must be ensured that the relative position at initial and end time stays the same.

[3] An *active* object is controllable by software whereas a *passive* object is static and cannot be moved by the system. External actors might still change properties of a *passive* object though.

[4] In this document the term *Axis allocation* is used as equivalent.

Example

Let's consider axis x where a move on $d2$ is performed. Its current axis occupation is described as

$$\begin{pmatrix} d1_x \\ d2_x \\ d3_x \\ d4_x \\ d5_x \end{pmatrix}$$

If the occupation after the move is

$$\begin{pmatrix} d1_x \\ d3_x \\ d2_x \\ d4_x \\ d5_x \end{pmatrix}$$

it can be said that a cross over of $d2$ and $d3$, hence at least one intersection occurred. Thus, those two devices collide on axis x. If, however, the occupation list is

$$\begin{pmatrix} d1_x \\ d2_x \\ d3_x \\ d5_x \\ d4_x \end{pmatrix}$$

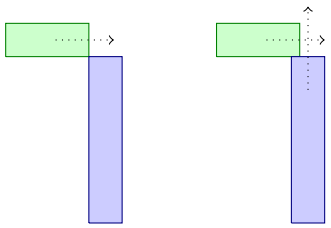
no intersection or crossing occurred. The crossing of $d4$ and $d5$ does not affect $d2$ as no cross over it occurred.

Object Dimensions

In the previous section Sweet Prune has just been described for punctual representations. Applying it to objects requires additional logic.

When just the most advanced point is chosen for collision checks, an intersection might occur anyway if the obstacle changes its position after that point has been passed and the device's endpoint is still residing within the obstacle's area (Figure 2.3).

One approach to solve this problem is to check both, the start and end point of a device at their passing moments. If the latter cannot pass it indicates a moving obstacle. In such a case, the order of checks may be



(a) t_1:Valid (b) t_2:Error

Figure 2.3: The axis allocation indicates that the device is able to pass the obstacle by applying the corresponding axis offset. However, while the obstacle is being passed by the device, the obstacle moves upwards and causes a collision.

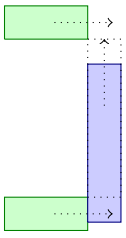


Figure 2.4: By using the device's end as synchronization point for an axis allocation computation, the movement of the obstacle can be taken into account. In this example there are two possible evading solutions.

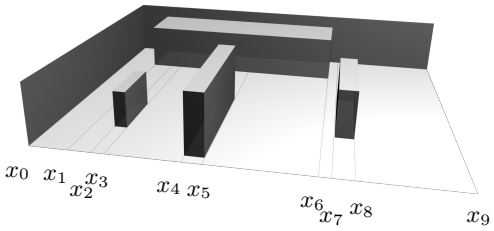
inverted to see whether the obstacle can be passed anyway with a different offset.

Extending Sweep Prune so it takes into account the object dimensions doubles the number of points which must be checked against. Obviously, the start point s_1 has to be checked against all other endpoints e_n and its end point e_1 must be checked against the other obstacles starting points s_n.

List Creation

The Sweep Prune lists can be created iteratively by querying the objects on an axis sorted according to the lowest axis allocation start. The actual values stored may be representing logical or absolute physical positioning values.

For the given environment



the axis allocation for x including all devices on that axis at t_0 is

$$\begin{pmatrix} x_1, x_6 \\ x_2, x_3 \\ x_4, x_5 \\ x_7, x_8 \end{pmatrix}$$

By keeping the lists sorted, the number of checks can be reduced down to the number of queries required by a binary search on that list. After that, every point must be evaluated until e_1 is reached.

2.3.2 Parallel Pruning

The approach of Sweep Prune just works if there is just one device moving at a time. Multiple parallel moves cannot be modeled as the comparison in the allocation lists happens before the move starts and the list gets updated at the end of a move. Collision checks of other devices will result in wrong results because with this race-condition-like behavior they do not use the actual position of the already moving device.

There may be scenarios though, where a collision test would indicate a collision because of overlapping axis allocations, but the move would perform just fine as the axis allocation intersection would not occur in the same time (Figure A.8).

Thus, parallel moves cannot be implemented with the Sweep Prune described.

Therefore, the approach described above has to be extended to determine the axis allocation at a given time.

A function

$$alloc(axis, t\Delta)$$

is introduced, allowing to query the axis allocation for all involved axis at every future moment $t\Delta$.

With such a function parallel movements can be modeled. This would require the caller to verify the method's result for every step a device takes during a move: A procedure which is highly inefficient and impossible to solve properly as the step range to be checked is theoretical infinite small. By increasing the step width however, small objects could slip through collision checks.

This problem comes from using absolute destination values when describing an axis allocation and would not exist if a value would be used which describes the move process instead of defining its final position.

To solve this, a new function is introduced: Every device carries a function

$$fm_a$$

for every axis and subaxis a describing the movement of a device. The function is considered active until the device reached its end point.

To check for collisions, the caller can evaluate possible relevant functions of other devices to determine whether they have an intersecting solution space, and if so, at which

$t\Delta$ this intersection occurs. The evaluation could be realized with an equation system where all fm_a of obstacle candidates and the moving object's fm_a are evaluated to see whether the solution is beyond the desired target position.

With this approach $alloc_a$ needs to be called just once for intersecting function results.

This approach requires the function to be monotonic though. If a device move cannot be described with a monotonic function it must be split into several monotonic ones whose evaluation will be delegated by the origin function fm_a. It must ensure that the right function for a given $t\Delta$ is called.

If the collision checks succeed, it means that the move can be performed and no additional counter measures to avoid a collision are required.

Allowing axis allocation computations at every desired moment undermines the purpose of Sweep Prune which uses sorted allocation lists for speedup because sorted lists would need to be created at every $t\Delta$. However, Sweep Prune checks can still be used for static objects reducing the overall number of function evaluations.

2.4 Collision Resolution

Collision Resolution is the second essential component for waypoint creation. In the previous section the basic principle of detecting collisions between objects has been discussed. Once such a state has been identified, actions must take place to avoid a real collision. There are two options:

Change Speed The speed of one or both moving devices is changed so the axis allocation during the computed collision time does not intersect.

Change Path If an object collision cannot be avoided by just changing the speed of devices, the planned path of one or both devices must be changed in a manner that the axis allocation does not intersect at any time with other devices during the move.

2.4.1 Necessary Data

To determine which one or which combination of those actions will result in minimal execution time, more data is required than just the physical size of objects like in section *Path Finding*. For every axis and subaxis a two sets of new attributes are introduced:

$$kinematic_a$$

containing information about the kinematic capabilities of an axis a

- Range
- Maximum and minimum speed
- Maximum and minimum acceleration factor

and

$$state_a$$

containing information about the state of an active move on axis a

- Move function
- Acceleration function

whose values may change during execution time.

2.4.2 Basic Procedure

With the axis allocation intersection mechanism described in section 2.3, the system can determine exactly at which $t\Delta$ a collision would occur. Once this moment has been computed, the collision resolution mechanism fetches all relevant $state_a$ at $t\Delta$ and determines according to *kinematic* and *state* of every axis involved which actions would be necessary in order to avoid a collision. The solutions computed represent new waypoints.

The actual representations of those attributes may be realized in different ways which influence the algorithmic and calculative complexity of a collision resolution system considerably.

2.4.3 Function Resolution

Representation

One approach is to model the axis allocation for every value in axis range with a function where $\frac{state_a.size}{2}$ is added to each side of a function value (Figure 2.5). This is repeated for every axis where collision resolution must be supported on.

The obstacle which would collide with the moving device can be applied to its functions. The results represent the exact positions on the corresponding axis at which the collision with the obstacle would occur (Figure 2.6).

It might seem obvious that the range will always have the same function graph. This does not count for devices with non proportional axis allocation characteristics. Non linear and complex movements can be modeled[5], may require additional functions though.

Once the intersection points between the function graph and the obstacle have been

[5] Refer to Figure A.6 and Figure A.7 on page 207 for examples.

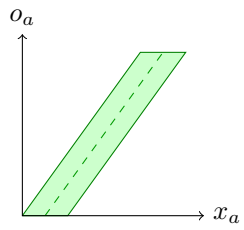


Figure 2.5: Graph of a function which calculates the axis allocation for axis a.
The dashed line represents the actual function graph, the green area is the axis allocation for a parameter within axis range.
o_a represents the supported axis range. Its value is in a normalized form between $[0, 1]$.
x_a represents the environment's axis range.

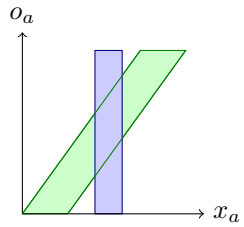


Figure 2.6: A colliding obstacle can be applied to the function to determine which alternative positions a device could take in order to avoid the collision.

calculated (Figure 2.7), the normalized offsets $o_a \in [0, 1]$ is known which could be applied to $axis_a$ in order to bypass the obstacle.

Note that the "relevant state" mentioned above does also include obstacles whose dimension embraces the moving device as they might reduce the solution space of the resolution evaluation.

Calculation

The function is evaluated for every axis on which a collision would occur. The different results comply with the normalized offset a device has to apply in order to avoid the obstacle. With $kinematic_a$ of the corresponding device the cost for every alternative is calculated. If such an alternative can be found, the obstacle can be avoided.

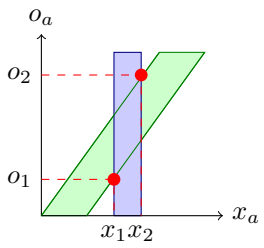


Figure 2.7: Once the colliding obstacle was applied to the function, its intersection points are known to the system. In this example the device on axis a could move within $[0, o_{a1})$ and $(o_{a2}, 1]$, without colliding with the obstacle.

Subaxes

The robotic arms in the target environment are normally composed out of multiple subroboters which are connected to each other (For examples refer to Figure A.9 and Figure A.10 on page 209).

This circumstance causes an increase in complexity of the collision avoidance problem by the number of subdevices attached to a device. Collision resolution can still be performed with the already described function resolving mechanism. However, the device's function Figure 2.5 must be applied within the parents' ranges. This indirection converts the already defined and absolute values into relative ranges.

The new range function can be calculated rather easily. All gradients and offsets of the parent's formula can be added up. The result is a function for the lowest device which can move in the range of its parents. It can be used to determine evading points with the evasion point computation mechanism described above. Because the ranges do not change during run time, the merging process can be done during an initialization phase.

Parent devices must be tested separately for collisions after the merged function indicates a collision free move. This is required because dimensions of parent devices are not included in the computation. This means the collision detection mechanism must be extended in order to take this into account.

The axis allocation of a subaxis can extend the one of its parents or reduce it.

Example

Considering a simple robotic device with a carrier module and an interaction module attached to it (Figure A.9). The carrier module can move within a predefined range on the x axis, the interaction module cannot move on its own.

Because the interaction module needs to be moved during run time, its range function is used for collision detection and resolution. The system takes into account the actual capabilities of the robotic device, the function must be extended with its parent ranges.

First, all range functions of parent modules are be calculated (Figure 2.8). Additionally, the range function of interaction module must be generated (Figure 2.9)

Then those functions must be merged together so the result represents the maximum range the interaction module can reach (Figure 2.10). With help of this function the actual range of the interaction module can be determined (Figure 2.11).

The "mounting offset" (the range in Figure 2.11 between 0.1 and 0.35) of a subdevice is automatically included in the function if it is attached in the middle of the parent device.

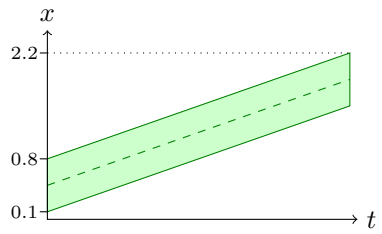


Figure 2.8: The range of the carrier module

Figure 2.9: The range of the interaction module which cannot move on its own

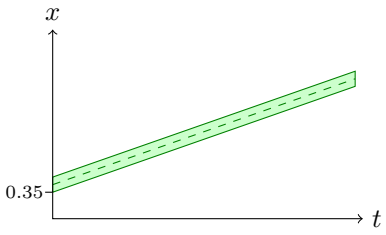


Figure 2.10: Range of the interaction module after merging it with the capabilities of its parent devices. This function may be used for evasion determination.

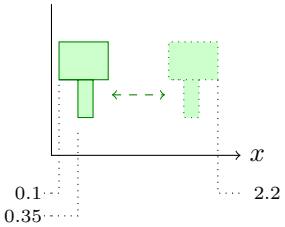


Figure 2.11: The range of the interaction module

2.4.4 Device Collaboration

Sometimes a device cannot advance with its movement because other devices are blocking their way. There are two possibilities – Passive Evading and Active Evadign – in such a situation which are described in the following paragraphs.

Passive Evading

If the blocking object is currently not performing a move, it must be moved away in order to let the device pass. Such an action is basically a prefixed logical waypoint as described in 2.4. If no solution can be found, it means the target position is not reachable.

Active Evading

There may be scenarios where the blocking device is actively participating in a move operation. In such a situation there are two actions possible:

Wait The device waits until the blocking device moves away or reaches passive status so it can be requested to move away.

Collaborate Both devices perform corrective movements so the collision is temporarily resolved.

Depending on the least overall costs, either possibility can be chosen. Normally the second approach tends to cause less run time overhead.

Division of Cost As it would be suboptimal in terms of overall run time when just one device would perform an evasive action, dividing the costs of movement which occur between the involved actors is required.

The required offsets which must be applied can be calculated from the current axis allocation and then distributed to the devices according to their corresponding axis acceleration factors.

Priority Schema There may be move operations which have real time requirements. Involving its devices in scheduling operations would be wrong as it could possibly make the move exceed its deadline.

A "Priority Flag" could be introduced in move representations. This would disable evading mechanism for a specific move.

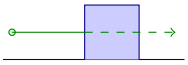


Figure 2.12: Because of an obstacle the device cannot reach its target location without performing additional collision avoidance measures.

Therefore, the more generic solution of introducing a priority mechanism has been chosen.

Every move contains a number describing its current move priority[6]). Evading operations can be performed on devices whose move priority is equal or below the current one. The number can be specified while the move operation is requested, thus giving the caller full control over the prioritizing schema.

2.4.5 Evading Axis Alternation

Collision Resolution loads the relevant state of the colliding objects and computes evading points with help of their range and axis allocation functions.

Once an evading point has been found, the calculated offset is applied to the device and the resolution continues with the new data.

Because security margins are not added separately to those evading points (Safety Clearance), the resulting evading point is basically still in a colliding position with the obstacle (Figure 2.12).

Therefore, the algorithm must not perform collision resolution on that axis anymore.

2.4.6 Safety Clearance

Because of the automatic control motors do not always perform exactly according to the specified move. Not taking those effects into consideration while planning moves would be careless, because driving without safety

[6] Priority weighting: The bigger the number, the higher the priority

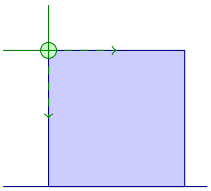


Figure 2.13: Once collision resolution has calculated the nearest evading point it is applied to the device's current position. Collision resolution will detect collision on two axes. The axis on which evasion has just been performed on, must be ignored.

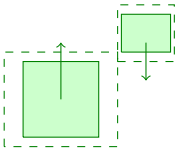


Figure 2.14: Safety clearance can be implemented by simply increasing the bounding box of the involved objects. Depending on the device and their axes, different margins can be chosen according to the device's move precision.

margin would sooner or later result in a collision.

Depending on the axis and device, different margins based on weight and maximum speed must be applied.

Obviously, extending the collision resolution algorithm would be the easiest approach. It can read the margins from the corresponding device and add them to the calculated evading point.

To keep control and logic flows of those central components simple, security margins are realized by increasing the object's size. With this approach, no changes in the collision resolution algorithm are required and security margins can be added while defining the object's dimension (Figure 2.14).

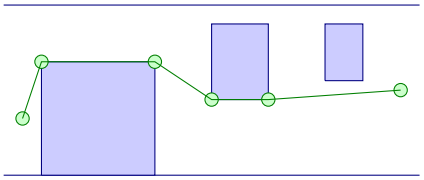


Figure 2.15: After waypoint generation the shortest path is created by connecting points in an optimal way.

2.4.7 Route Smoothing

When waypoints are created, they are placed at obstacle corners. The shortest path will then be created by connecting those points in an ideal manner (Figure 2.15).

For the moving object this usually results in sharp direction changes. This is time-consuming to achieve because the device must slow down its speed in order to keep the device in line. If the speed is not adjusted, the device is likely to overreach the targeted route.

Obviously, those sharp edges are not always necessary. By smoothing them, the device can reduce slowdowns at vertexes leading to a lower traveling time. Thus, the path should be made smooth whenever possible.

There are two approaches how to achieve this.

Sinus Edges

A smooth passing around an edge can be modeled with a sinus curve (Figure 2.16) if supported by the motion controller. The described process below uses a context of three points.

First, the maximum distance in which the new smoothed curve must reside is computed. The maximum distance is $\frac{P_M - P_B}{2}$ or $\frac{P_E - P_M}{2}$ whichever is smaller. It can be thought as a radius r around the middle point.

Then, the point in the middle is moved away from its current location, keeping its relative distances to the neighbor points. A move distance of $\frac{r}{2}$ has shown to give good

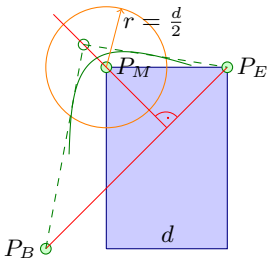


Figure 2.16: The middle point is moved keeping its relative directions to the start and end point. In this example, the original middle point is closer to the end point. Thus the radius is set to $d/2$. The moved point is connected with start and end point. The resulting intersections on the circle with r are used as starting points for the sinus curve.

results.

The moved middle point is connected to the other two points. The two resulting intersections with r are used as start and end point of a $\text{sinus}_{[0;\Pi]}$, whose angles match the opposite angles of the incoming lines.

To smooth a route with multiple waypoints, all points except the ones at start and end must be moved with their corresponding $\frac{r}{2}$. After this has been done for every point, the sinus curves can be drawn tangentially around the edges.

With a dynamic r defining the smoothing area, this approach calculates curves according to the involved objects. Waypoints far away from each other can be connected with long curves allowing higher following speeds. However, the rounded edges are still connected with a straight line.

Cubic Spline Interpolation

A cubic spline is in mathematics a special function $S(x)$ that is defined by piecing together polynomials of third degree p_k, also known as *cubic polynomials*:

$$S(x) = \begin{cases} p_0(x) & x_0 \leq x \geq x_1 \\ p_1(x) & x_1 \leq x \geq x_2 \\ \vdots & \vdots \\ p_{n-1}(x) & x_{n-1} \leq x \geq x_n \end{cases}$$

The goal of the cubic spline interpolation is to find a curve fitting function (*cubic spline*) that goes exactly through defined control points (*interpolation*). As mentioned before, for the definition of a cubic spline a set of defined control points is needed. The control points are simple coordinates in the form of (x_i, y_i). The cubic spline is then constructed by interpolating a cubic polynomial between two consecutive control points (x_i, y_i) and (x_{i+1}, y_{i+1}). Constraints for constructing a cubic spline are listed below.

- Cubic polynomials shall pass through their endpoints x_n and x_{n+1}:

  $$p_n(x_n) = y_n$$

  and

  $$p_n(x_{n+1}) = y_{n+1}$$

- The first and second derivative of each cubic polynomial shall be continuous:

  $$p_n(x_n)' = p_{n+1}(x_n)'$$

  and

  $$p_n(x_n)'' = p_{n+1}(x_n)''$$

  The notation $C^2(\Omega)$ is used to denote this kind of twice-differentiable functions whose second derivative is continuous. In the field of computer graphics this level is called *C2 continuity (of curvature)* and is the highest quality level. Curvature continuity means that two curves p_n and p_{n+1} are tangential at a common endpoint x and have the same radius of curvature at that point. The first equation and the first derivative of both curves ensures that the curves are tangential at x. The second equation assures that the radius of the curvature is the same at the point x, resulting in a smooth curvature and transition.

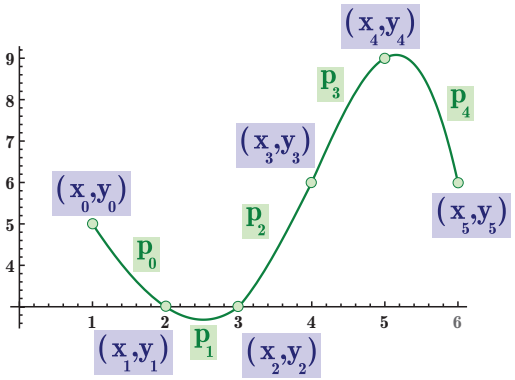


Figure 2.17: A natural cubic spline. The spline consists of the cubic polynomials p_0, p_1, p_2, p_3 and p_4.

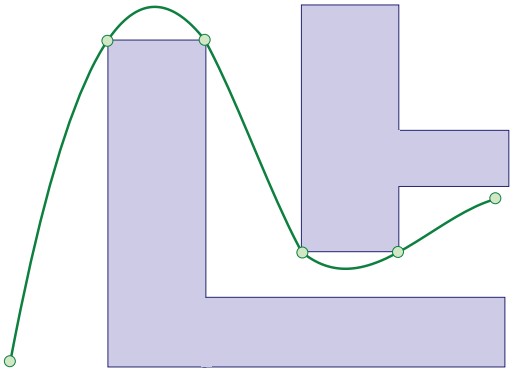


Figure 2.18: Example of a cubic spline applied on the obstacles' waypoints

- The two second derivatives of the cubic polynomials p_0 and p_{n-1} at the endpoints x_0 and x_n shall be 0:

  $$p_0(x_0)'' = 0$$

  and

  $$p_{n-1}(x_n) = 0$$

  Because of the last two equations the cubic spline is called a *natural cubic spline*.

An algorithm to calculate cubic splines based on a given set of coordinates is described in detail on Wikipedia[7].

Ideally, the amount of data which must be transferred is kept as small as possible. However, given a typical distance between two waypoints this results in a far too low resolution of the calculated spline.

This might not seem to be a significant challenge, but the inaccuracy caused by a too coarsely described curve my cause collisions with other obstacles. It also influences the device speed in a negative way.

However, in the following sections alternative charting models are discussed.

Acceleration Table Instead of specifying the desired speeds, passing the belonging acceleration ratio for every axis allows a more abstract description of the route.

After the movement has been split into translations of the involved axis, it is known which speed must be applied at a given time on the axes to achieve the curve.

One approach to compress that data is to further abstract the speed difference between points as shown in Figure 2.19. This means, just if the acceleration changes between two $t\Delta$ a new value pair is required to describe the corresponding curve. This usually leads to a reduced number of necessary pairs to describe a movement.

The acceleration can be computed by extending the original formula with an additional derivation on its results.

However, movements with many smooth parts like the cubic spline do still require a significant amount of points. This is because the derivative of such curves is not any simpler to describe than the original (Figure 2.20).

Thus, this approach does not solve the problem of limited points. For curves there are still many pairs required to model the movement.

Interpolation Formula A different approach reducing the number of points required is to implement the calculation of the cubic polynomials needed to construct the cubic spline on the motion control itself.

[7] `[Algorithm for computing natural cubic splines]`

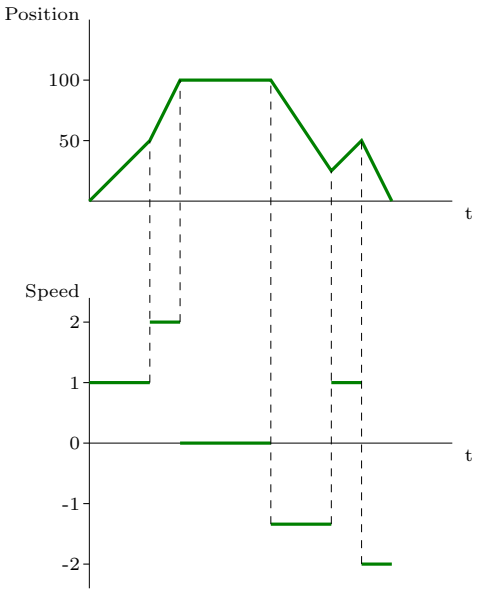
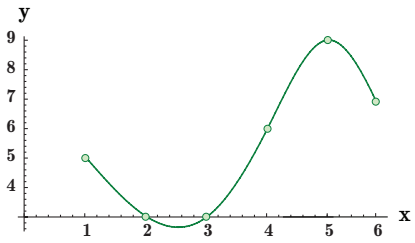
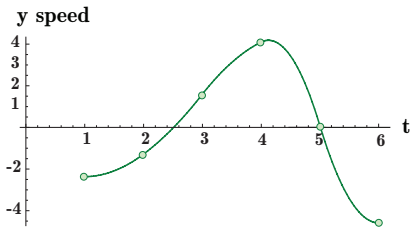


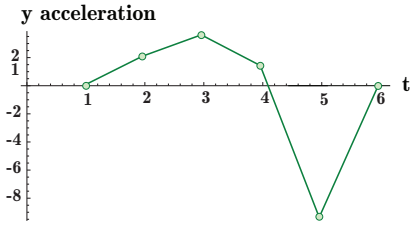
Figure 2.19: By describing the movement with its acceleration alternations, significantly less information is required to describe the move. In the above example, just seven acceleration factors are required to describe the movement, whereas the origin curve requires an update for every position change.



(a) Two dimensional spline



(b) Derivative of y over time



(c) Second derivative of y

Figure 2.20: Derivatives for the y axis from a given spline route.

The actual computation is fairly simple and thus can be easily implemented anywhere:

$$x \to d(x - x_i)^3 + c(x - x_i)^2 + b(x - x_i) + a$$

a can be omitted as it just describes an offset. The computation intensive determination of b, c and d is performed with help of a personal computer. The results and the applicable domain are then forwarded to the corresponding motion control which performs the movement. Following parameters must be included to enable a curve description from a given offset:

Domain start (D_s) The initial value with which the spline function will be called. By using relative offsets, this value can be omitted and replaced with 0.

Domain end (D_e) The maximum parameter with which the spline function will be called

Step width (S_w) Factor controlling move resolution and execution time. The total execution time is $(D_s - D_e)/S_w$

b,c,d Factors which influence the development of the curve

To actually perform the movement, the motion controller just needs to compute the acceleration ratio with help of the given arguments it received and apply it to the corresponding axis (Algorithm 1).

In order to prevent a stalling of movements after the end point has been reached and before the next polynomial has arrived, a queuing mechanism is also required. An option to clear already pending polynomials would allow a delayed updating of already pending movements.

A too small distance between waypoints results in an unexpectedly high curve (Figure 2.21). To prevent this, a minimum distance value with which a cubic spline is applied must be defined. If this value is undercut, another route smoothing approach must be chosen.

Algorithm 1 Enables Cubic Spline support for motion controllers

Require: Axis in a defined state
Ensure: No move pending on axis
Ensure: $x_{inc} > 0$
  $x_{start} \leftarrow 0$
  $oldpos \leftarrow$ current position on axis
  for $i = x_{start}$ to x_{end} **do**
    $i \leftarrow i + x_{inc}$
    $pos \leftarrow dx^3 + cx^2 + bx$
    $accel \leftarrow (pos - oldpos)/x_{inc}$
    if $|accel|$ is too high **then**
      signal error condition
    end if
    $oldpos \leftarrow pos$
    apply $accel$ to axis
    wait x_{inc}
  end for

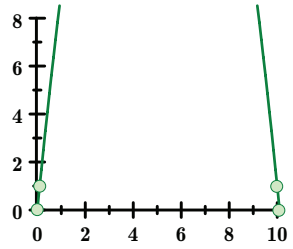


Figure 2.21: Certain waypoints constellations may result in unexpected curves

Because the smoothing algorithms are applied in a post-processing phase, they perform "blindly". This requires an additional check between the computed curve and axis allocation functions to ensure no collision will occur.

Apply Curve to Axes The route must be partitioned into a graph for all involved axes. They can then be sent to the corresponding device which performs that movement. In this partitioning step, a time factor is introduced automatically which might require further adjustments depending on the axis' capabilities.

The axis specific graph which got computed by the cubic spline algorithm may not be directly applicable to the device. Whenever the acceleration exceeds 1 or undercuts -1 it means the corresponding axis will not be capable of executing the movement within the time requirements posed by the other axis (Figure 2.22).

In order to give the axis with a too high acceleration factor time to reach its target state, the other axis must slow down.

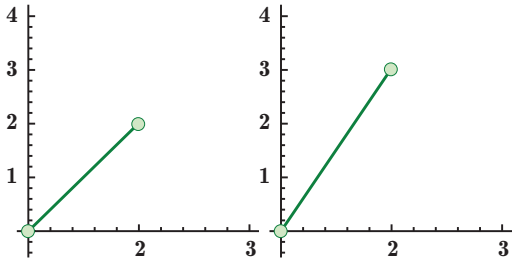
If the absolute value of the acceleration on an axis between two waypoints is higher than 1, the other axis decelerates by a factor $1/|slope|$.

Additionally, when the two axes have a different maximum speed, the according factors must be further adjusted to achieve an unified representation.

After performing those corrections, it is ensured that $|\textbf{slope}| <= 1$. The problem of too sharp curves as shown in Figure 2.21 is automatically resolved if those corrections are applied and the resulting graph serves as basis for computing the splines.

The only remaining problem is that despite the limited slope factor which is given by acceleration limits, the spline may consist of parts where there are slopes factors bigger than 1. By deriving the spline another time, too high factors can be identified and eliminated.

The whole process increases the total duration of the movement.



(a) **slope** $= 1$ (b) **slope** $= \frac{3}{2}$

Figure 2.22: In the left picture the acceleration factor does not exceed $|1|$. This means the graph can be split according to the axis and sent to the device. In the right picture however, the graph cannot be straightly applied because y does not raise fast enough and would not be 3 when x reaches 2. Therefore, the movement on x must be decelerated by $1/$**acceleration** according to the y's maximum acceleration ratio, i.e. the target point must be moved to (3,3).

In order to avoid too many computations because of the late stage of the process, cubic spline support is ideally included into collision resolution. The cubic spline logic should be factored out, for example into a strategy pattern to keep the two concerns separated and exchangeable.

An example how the speed factors are adjusted from a given set of waypoints is shown starting with Figure 2.23. In Figure 2.24 the translations are put into relation with a time factor. In Figure 2.25 the slope leveling is performed and finally in Figure 2.26 the normalized points which do not contain acceleration factors over 1 are splined.

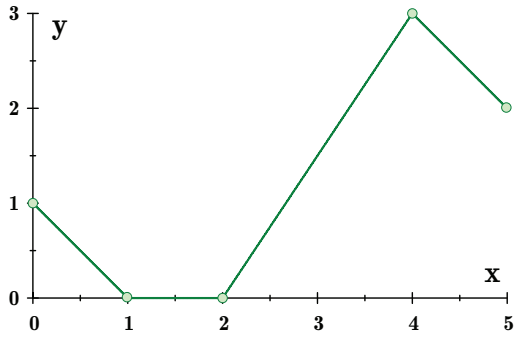
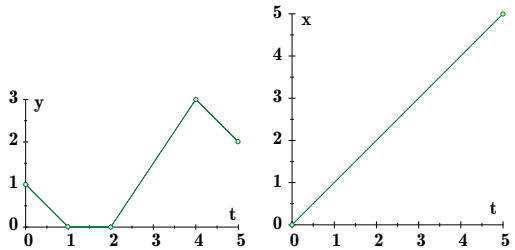


Figure 2.23: A set of connected collision free waypoints in a two dimensional space.



(a) y split (b) x split

Figure 2.24: The route is split into the available axes. As the movement never turns, x is accelerating steadily.

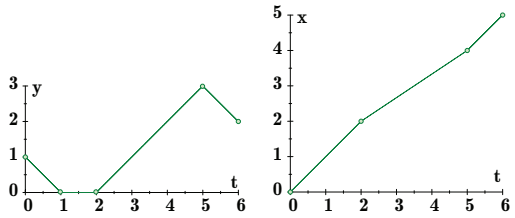
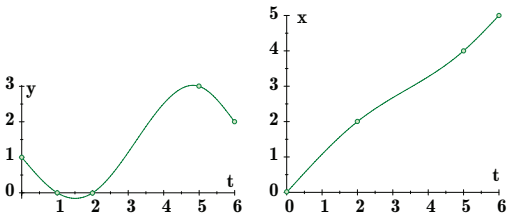


Figure 2.25: Where the development of the y graph exceeds 1 (between 2 and 4 on t), t is adjusted that the maximum factor is 1 again. Additionally, a slow down on the x graph is required as well to keep the movement synchronized.



(a) y spline (b) x spline

Figure 2.26: The normalized graphs are used to generate axis-wise cubic splines routes which are then applied by the motion controller.

Chapter 3

Realization

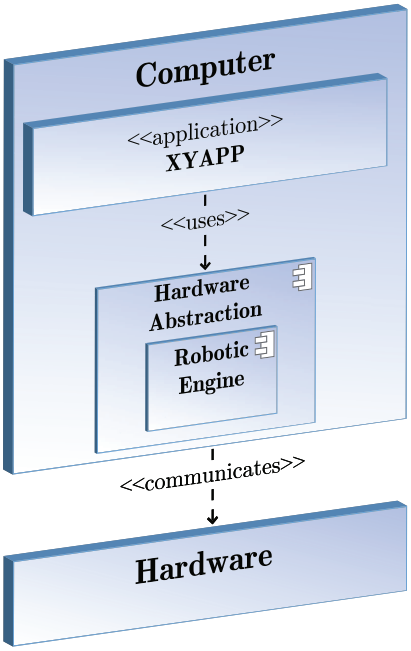


Figure 3.1: Deployment diagram representing the logical architecture

3.1 Design

The robotic engine is a component within the hardware abstraction.

As the component developed serves as basis for further development and must be easy modifiable, it is split into subsystems separating concerns and keeping a certain level of modularity. As shown in Figure 3.1 those subsystems are built according to the structure of the previous chapter. It turned out to give a good modular layout. The engine class itself serves as controlling instance which supervises the logical flow. All subsystems are kept stateless reducing errors caused by side effects and eliminating excessive synchronization needs.

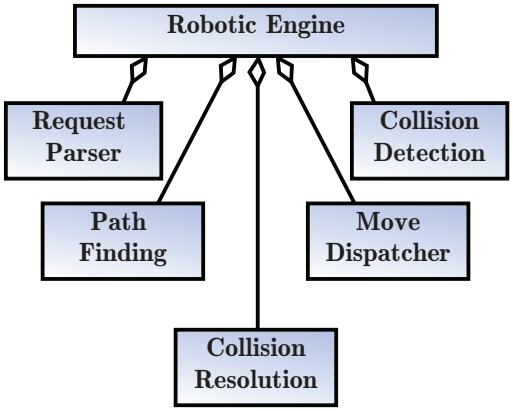


Figure 3.2: Subsystems within the robotic engine.

Request Parser Converts move requests received at the system boundary into an internal representation.

Path Finding Contains the logic to traverse a map of waypoints.

Collision Detection Is able to check if a collision would occur for a given movement.

Collision Resolution Is able to compute evading points in order to avoid a detected collision.

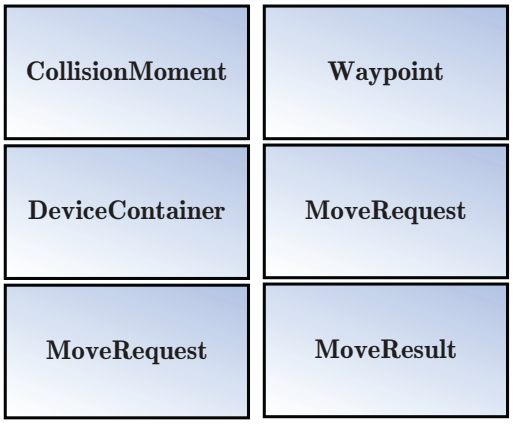


Figure 3.3: The internally used data classes

Move Dispatcher Performs and controls the actual movement asynchronously.

Additionally, the data classes listed in Figure 3.3 are used in interactions between the subsystems.

Device Container Wrapper class for underlying drivers. It serves as storage for move engine related meta data.

Move Request Internal representation of the command received. Instances of this class are generated by the Request Parser subsystem.

Waypoint Describes a device's state at a given time.

Translation Describes a move of a device.

Collision Location Describes the state at which a collision would occur including relevant objects and their properties.

Move Result Serves as synchronization object for pending moves.

In the following sections the mentioned subsystems and data classes are described in more detail. Non-obvious structures and patterns are explained.

3.1.1 Engine Interface

This interface serves as system boundary. It is kept rather minimal and device independent. Its simple definition allows the calling component to control the movement without knowledge of environmental and device specific details.

Devices Property

In this read-only collection the devices controllable by the move engine are listed. Their identifiers can be used in move requests.

Objects Property

In this read-only collection the objects which are known to the move engine but take in a passive role and thus cannot be actively controlled are listed. Their identifiers can be passed to the move method as parameters.

Move Method

Basically, there are move commands and objects where those commands can be applied to. The move command can be configured with any configuration data describing how the object should be modified.

Move Return Value The move method has asynchronous calling semantics. Therefore, a mechanism is required to synchronize for move completion once the call has been accepted. This is achieved by returning an instance of `MoveResult`. The client is able to observe and, if necessary, abort the specified move command with that instance.

This approach is known as future pattern [7]

3.1.2 Move Engine Procedure

This component controls the move engine's flow.

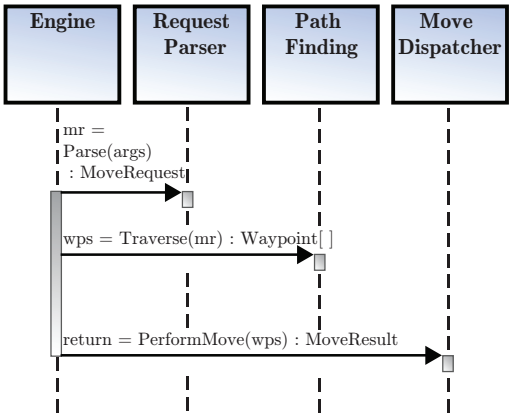


Figure 3.4: Flow diagram of a move request

Initialization

After creation, it waits until the `Devices` list has been initialized and its `Init` method is called[1].

In that initialization routine available devices will be wrapped in `DeviceContainer` instances and converted for easy computation in later steps. Then, the subsystems are created and initialized with the newly created device list.

Move

Once the engine receives a move request, it passes the arguments received to the responsible `RequestParser`. If a `MoveRequest` instance could be created it will be passed on to the Path Finding subsystem. If a valid path could be found, the Path Finding subsystem returns a list of waypoints which must be followed. That list will be passed on to the Move Dispatcher subsytem which makes the corresponding devices move (Figure 3.4).

If an error occurs during this process it will be signaled in form of an exception.

The engine receives a standardized representation of the move command from the move engine controller and tries to calculate the according move vector.

[1]This is a compatibility hook for the existing framework

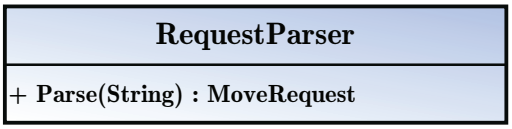


Figure 3.5: The Request Parser class

3.1.3 Request Parser

This subsystem is responsible to convert the commands received into an internal representation.

The move method has just one parameter which allows the caller to describe the desired move. It must be possible to specify a device from the `Devices` property with an arbitrary list of parameters. As the range of possible parameters is not known to the caller because they are dependent by concrete device implementations, no proper data validation can be performed. Therefore, a generic move command syntax in a tree-like structure must be provided. This allows to access all available devices without explicitly state them.

Several representations have been identified as possible candidates:

- Composite object structure
- Plain String
- XML document
- XML document, DTD verified

Because of its simple creation and easy marshaling across system boundaries, the string format has been chosen as data representation for move commands. Other formats could still be implemented later.

The format required to convert the string into the internal Domain Specific Language (DSL) can be described with following grammar[2]:

[2]E:Expression, C:Command, R:Request, O:Object, P:Parameter, A:Argument, U:Unit, L:Length, V:Volume, F:Factor(Acceleration), D:Digit, S:String

$E \rightarrow E;E|CR$
$C \rightarrow move|take|free$
$R \rightarrow OP$
$O \rightarrow S/D$
$P \rightarrow A|PA$
$A \rightarrow S = DU|S+ = DU|S- = DU$
$U \rightarrow L|V|F$
$L \rightarrow um|mm|cm|dm|m$
$V \rightarrow ul|ml|cl|l$
$F \rightarrow L\hat{}2$
$D \rightarrow [0-9]|DD$
$S \rightarrow [a-z, A-Z]|SS$

```
move dev/1 x+=10cm
move dev/2 y=100cm z=10mm
move dev/2 y=1cm accel=2cm^2
```

Listing 3.1: Examples of valid move commands

Access to the corresponding devices is required in order to interpret the given values correctly. This is why the Request Parser subsystem must be initialized with a list of available devices.

Once the parameters have been parsed successfully a `MoveRequest` instance is created and returned.

3.1.4 Path Finding

In order to keep the traversal implementation exchangeable, it is put into its own subsystem and the usage of Collision Detection and Collision Resolution as described in the previous chapter is abstracted away. A simple node class (Figure 3.6) was built converting standard map operations into appropriate calls to other subsystems. The usage of the Path Finding subsystem could also be abstracted. The path selection strategy can be changed easier like this.

3.1.5 Collision Detection

Given a target position and device, this subsystem determines whether a collision would occur during the move and if so, where that collision would occur. With the result Path Finding may create a waypoint and use it for further path probing.

It could be implemented as proposed in section 2.3 on page 27.

On every request the allocation lists are generated for the desired $t\Delta$ and evaluated. To detect a device cross following logic is used:

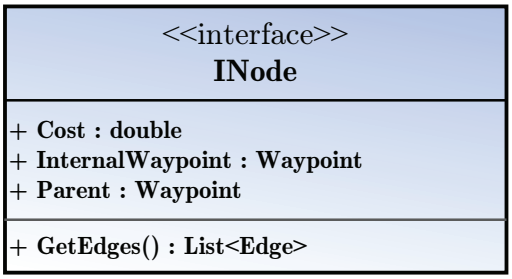


Figure 3.6: Internal Node interface of the `PathFinding` class.

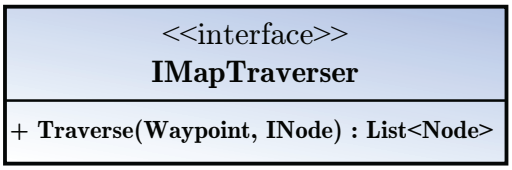


Figure 3.7: Main interface of the `PathFinding` class.

Algorithm 2 Collision detection algorithm

Ensure: Exactly one device in request
  $r \leftarrow$ request
  $m \leftarrow$ moving device
  for obstacle o in obstacle list **do**
    for axis a in x,y,z **do**
      $b \leftarrow o_a$.func(0)
      $e \leftarrow o_a$.func($r.t\Delta_a$)
      $d_b \leftarrow b - m_a$.func(0)
      $d_e \leftarrow e - m_a$.func($r.t\Delta_a$)
      if $sgn(d_b) = sgn(d_e)$ **then**
        break { No collision with o on a}
      end if
    end for
  end for

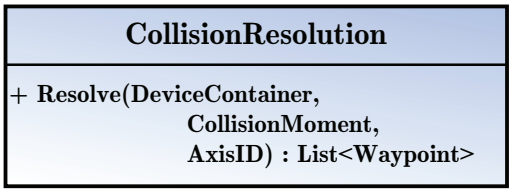


Figure 3.8: Diagram of the Collision Resolution class.

A possible optimization which would reduce the number of checks is to solely check the axis allocation once if the queried obstacle is a passive one.

As described in 2.3, the collision checking process can be aborted once it has been stated that no collision will occur on the axis observed. This reduces the number of required axis allocation queries to 1.5 on average per device.

This subsystem has one public method, `GetSortedCollisionMoments`. It creates a list of `CollisionMoments` sorted according to their occurrences which would occur if the given device would move from the start to the end node.

The resulting list is then used in other components to compute evading points. It is sorted because the elements are normally processed in a chronological order.

3.1.6 Collision Resolution

The Collision Resolution subsystem computes evasion points which must be reached until a Collision Moment in order to avoid a collision. It exposes one method to the other parts of the engine.

The first parameter contains a reference to the device driver representing the arm which should be moved. The second parameter contains the obstacle and time information about the collision. The third parameter identifies the axis on which the collision has been detected. This parameter is required for subsection 2.4.5 described on page 34.

After loading the axis allocation and dimension functions from the moving device and the involved obstacle, it resolves those functions to compute evading points.

It returns a list of waypoints which would avoid the obstacle referenced in Collision Moment. If the list is empty, no evading points could be found.

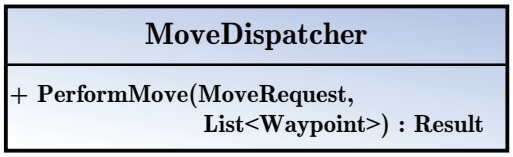


Figure 3.9: Diagram of the Move Dispatcher class.

3.1.7 Move Dispatcher

Access to the motion controller is required to be realized asynchronously. This requires a dedicated subsystem where the waypoints can be processed.

However, a move might also require multiple waypoints which must be passed. If the underlying motion controller support queuing of polynomials, all generated waypoints can be passed at once. But if there are restrictions regarding the number of asynchronously sent waypoints, a detached process is required which works off the waypoints.

The dispatching has been factored out into a separate class in order to be able to switch the underlying interface easily. This simplifies any upcoming changes caused by modifications of the underlying calling semantics.

The subsystem itself (Figure 3.9) exposes one method which requires the parameters listed below.

- Move request of the original command
- Set of waypoints which must be followed to move the device specified in the move request successfully through the environment

3.1.8 Function Representation

For Collision Detection, Collision Resolution (described in the previous chapter) and axis capabilities, functions are used as internal representation. In order to resolve functions efficiently they must be provided in a standardized way. The simplest way is to put the arguments into a data representation which allows native access as otherwise an expression evaluator must be written. That would introduce unnecessary parsing overhead.

An exemplary representation for

$$x \rightarrow 2a + b$$

might be implemented as

```
public struct Func
{
 double A
 double B
}
```

Supported functions are monotonous linear functions as

$$x \rightarrow ax + b$$

which are represented as

```
public struct LinearFunction
{
 double A
 double B
 double X
}
```

and quadratic functions like

$$x \rightarrow ax^2 + b$$

whose internal representation is

```
public struct QuadraticFunction
{
 double A
 double B
 double X
}
```

Those representations allow highly efficient evaluation of functions because native data types can be used. It allows the move engine to determine at which point a collision will take place as the two concerning functions can be transformed to determine that value.

The content of both data structures is the same, they must be implemented in different data types though as they require different computations during evaluation.

However, restricting support to just two types of functions means that movements can just be expressed as linear or quadratic functions. Once this is not sufficient anymore, the decision of using the representations above has to be reconsidered as the usage of an expression solver would be more appropriate.

Example

$$\mathrm{f}(x) = 0.5x + 5$$
$$\mathrm{g}(x) = -3x + 10$$

The engine is able to solve the equation by arranging the values in the data types accordingly:

```
pX = (f.B+g.B)/(f.A+g.A)
```

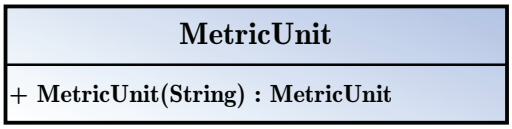
However, precision loss might occur with numbers which contain fractals which cannot be represented in the systems floating point data types. This means the precision of a predicted collision point is bound to the used data type.

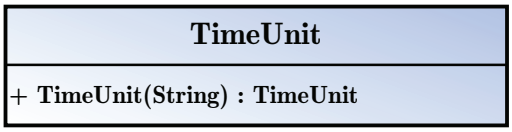
3.1.9 Motion Controller

There are several types of motion controller within the system. Each has to provide a standard set of kinematic information so the move engine can use that information to plan collision free moves.

Additionally, they are also responsible to convert the standardized representation of a vector into their own compatible format.

Because the engine may not know the different controllers in advance, they are abstracted with help of an interface which exposes the capabilities of the corresponding device.





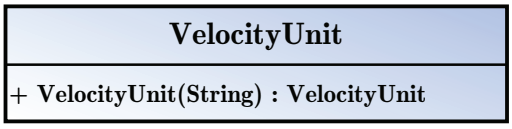


Figure 3.10: Unit classes

3.1.10 Unit Handling

Creators of move requests might want to choose appropriate units when defining their moves, driver implementations depending on their axis precision.

Requiring a format with a standardized unit leads to situations where values must be specified in an unnatural way like 1000000 µm or 0.001 cm. This is a potential source of errors because it is not immediately evident what length is specified. Additionally, already existing code may break once the internal data format is changed.

To decouple internal and external units and provide a natural way for value specification, unit classes are introduced (Figure 3.10). All components which interact with the robotic engine can use the format which is best for them or even implement their own unit class.

At the system boundary, they are converted into a unified floating point representation which is used for the internal computation (Figure 3.11).

3.1.11 Move Request

A `MoveRequest` instance is used as internal representation for received requests. It is created by the Request Parser subsystem

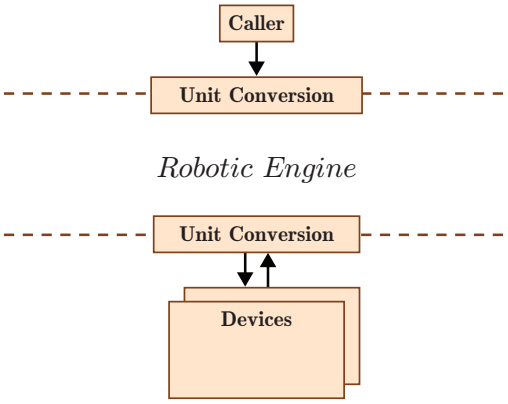


Figure 3.11: At the system boundaries unit types are transformed into a standardized internal form.

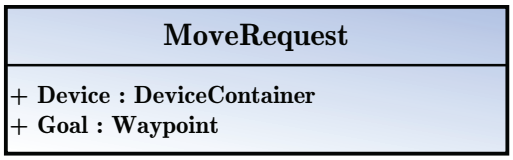


Figure 3.12: Diagram of the Move request class

and used in other subsystems to perform collision avoidance and perform movements.

3.1.12 Waypoint

A waypoint is implemented as defined in section 2.2.1. It represents the state of an object at a given moment.

In order to simplyfy access, the index operator is overloaded to enable programmatically access to its members by specifying the corresponding axis.

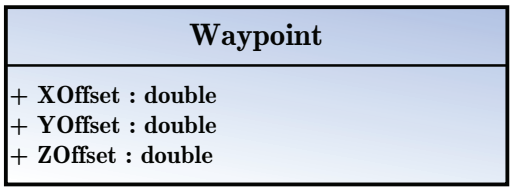


Figure 3.13: Flow diagram of a move request concerning the Move Engine

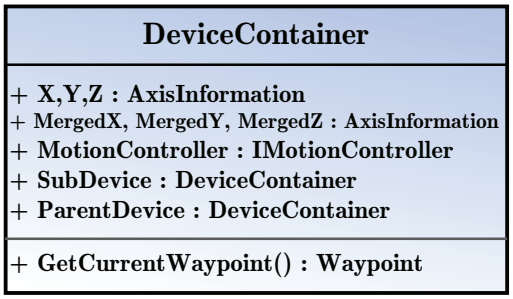


Figure 3.14: Diagram of the Device Container class

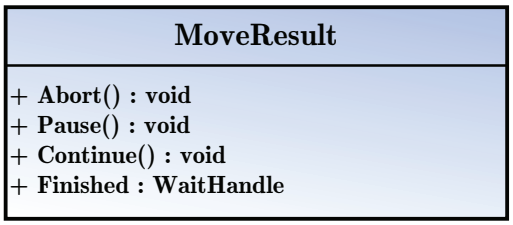


Figure 3.15: Diagram of the Move Result class

3.1.13 Device Container

The framework's motion controller drivers cannot be modified. Therefore, a wrapper class is required which enables the engine to associate internal meta data to them.

Additionally, subdevices are not modeled in the underlying layer and can be represented with help of the container.

The functions starting with "Merged" represent the merged functions presented in section 2.4. They are required to transparently support subdevices in the detection and resolution subsystems. The merged functions will be built in the container's constructor.

The `GetCurrentWaypoint` method uses the controller's function to build a `Waypoint` instance which can be used in other subsystems.

3.1.14 Move Result

Move Updates

Once a request has been accepted by the Move Engine, the caller returns a synchronization object which can be used by the caller to determine when the command has finished.

There may be cases, where the caller is required to change the move while it is already performing. Following update mechanisms are supported:

Abort The move is aborted. The device stops and stays at its current position

Pause The move is paused. The device stops but keeps the desired position in memory.

Continue A route from the position of the currently paused device to its past target position is computed and applied.

Methods supporting the listed move updates are exposed in the synchronization object because it allows to identify the update request to the pending move.

Synchronization

The defined grammar allows the specification of multiple move requests at once. The waypoint generation will take place before any movement is performed.

However, there are cases where a command needs to update settings first (e.g. acceleration) before performing its move. If a command requires many such settings the commands execution is delayed for several milliseconds.

To synchronize a movement, the parameter $isInGroup$ can be set to `true`. This delays the move until all devices have configured their devices.

3.1.15 Natural Cubic Spline

The algorithm could be implemented as described in section 2.4.7. It has a runtime behavior of $O(6n)$ and was implemented as independant module in the engine (Figure 3.16) and in the Spline Tool. It takes a set of waypoints and returns a list of the corresponding polynomials.

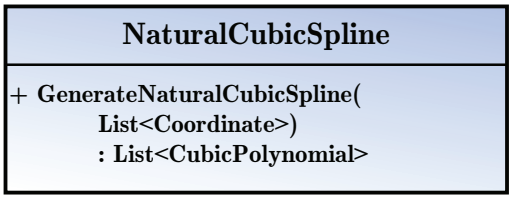


Figure 3.16: The Naturual Cubic Spline class

3.1.16 Extensions

Action Oriented

In the system presented, the involved devices must always be specified while creating a move request. This is cumbersome and because the device identifiers are not static, persisted move requests might not be applicable anymore once they get executed.

One approach to free move requests from device identifiers is to further abstract them. By just specifying the desired result no concrete device identifier is required anymore. The engine can determine on its own which devices must be used. A request like `move tube_1, storage_8` could be parsed by the engine as:

1. Query the specified object to determine how it can be moved.
2. `tube_1`'s class marks itself as moveable by a gripper
3. Currently there is one device with gripper functionality available, `dev_3`
4. Start movement
   (a) Move `dev_3` to `tube_1`'s position
   (b) Apply the gripper functionality
   (c) Move `dev_3` to `storage_8`'s position
   (d) Free the gripper

This would require an extension in the parser subsystem as well as in the device drivers which must be enriched with meta data. Additionally, a set of processes must be available in the engine which are applied depending on the move requirements of the corresponding device.

3.2 Implementation

The engine was required to be implemented in C#, .NET Framework 3.5. The presented concepts and designs have been partially realized. Subsystems have not been included in the engine when it would require major changes and they are planned to be used in a different context anyway.

3.2.1 Axis Logic Redundancy

Often a computation must be performed for several axes which require parameter from different sources. In order to avoid repeating slightly varying code sequences, accessors which allow programmatic axis selection have been implemented in all major components:

```
AxisID id = this.GetID();
moment[id] = this.Resolve(
  device[id].Func,
  device[id].Dim);
```

This certainly increases the complexity but is done anyway to avoid redundant code blocks.

3.2.2 Time Handling

Collision Detection works by resolving functions which are active at a given moment. Because the real motion controller is not actually accessed and can serve as time emitter, a separate notion of time has been implemented: A counter variable has been introduced which increases while moves are performing. The calculation of evasion points is still realized with time differences but once the computation has completed, the resulting time offsets are added to the current counter.

Additionally, although logically correct, negative time values are avoided when creating translations.

3.2.3 Active Evading

An active evading mechanism, where inactive devices move away from moving ones could not be implemented because of insufficient resources.

3.2.4 Dynamic Dimension

No detailed description of devices was available. Therefore, all dimension properties handled as static values are always evaluated with "0" and never with the time counter.

3.2.5 Performance

Where not explicitely stated, the tests were executed with a release build of the program on a 2.5GHz Core 2 Duo with 2GB RAM Laptop

Axis Distinction

Due to the generic nature of the engine there are many places where the program must determine which axis is affected.

There are several ways to accomplish the comparison. Depending on the data type used, different performance is achieved.

String Based Distinction Axes are identified by a string identifier.

```
if( axis == "x" )
  ...
```

int Based Distinction Axes are identified by a predefined integer constant.

```
public const int X=1;

if( axis == Class1.X )
  ...
```

enum Based Distinction Axes are identified by an enum. Although the enum uses integers internally, it may behave differently.

```
public enum Axes
{ X,Y,Z }

if( axis == Axes.X )
  ...
```

Test The test is implemented as listed below where *comp* refers to the respective comparison method.

```
int x=0,y=0,z=0;

for(int i=0; i<10^8;i++)
 if( <compX> ) x++;
 else if( <compY> ) y++;
 else if( <compZ> ) z++;
```

This is executed for all three implementations.

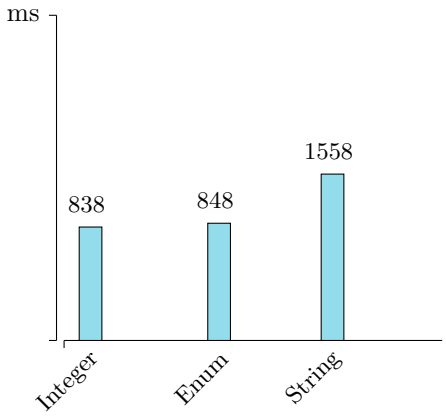


Figure 3.17: Execution times of different comparison methods after 100′000′000 iterations.

Conclusion The comparison overhead of all three methods is negligible. However, due to additional type safety the enum based comparison method will be used.

Delegate vs Method

During Collision Avoidance many function values of involved objects need to be calculated from a given argument. The factor and offset must be exposed anyway in order

to be able to solve function systems. The evaluation can be realized in different ways.

Lambda Expression A C# Lambda Expression which is easier to read in code. It is represented in a type which leads to better control.

```
public struct Function
{
 double A,B;

 Func<double,double> Evaluate
  = x => A*x+B;
}
```

Method A classical method is harder to read. Belated changes are harder to implement.

```
public struct Function
{
 double A,B;

 double Evaluate( double x )
 {
   return A*x+B;
 }
}
```

Test During the test the argument is updated making it impossible for the environment to cache any results.

```
for(int i=0; i<2*10^9; i++)
  func.Evaluate( i );
```

This is executed for both implementations with different data types.

Evaluating lambda expressions takes significantly longer than its method equivalent. Using `float` as data types also increases computation time.

Conclusion Function representations will be implemented with normal methods. The `double` data type is used as `float` representations have a big negative impact on computation time.

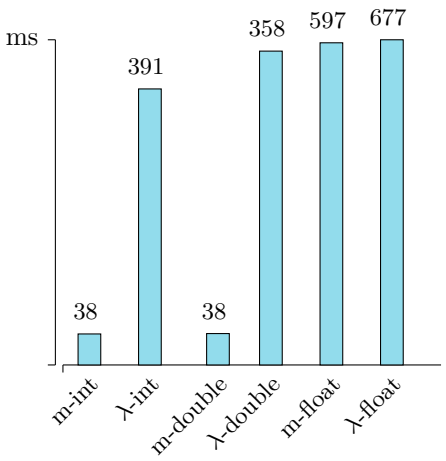


Figure 3.18: Execution times of method and delegate evaluations after 100′000′000 iterations.

Table 3.1: Configuration A

Parameter	Value
Number of coordinates	100
Maximal x value	100
Maximal y value	100
Number of iterations	1000

Natural Cubic Spline Algorithm

The algorithm for calculating a natural cubic spline is put to the test. The algorithm has been implemented in two different ways. Both versions differ in the memory allocation. The first version, which is used throughout the code and is the *safe* implementation of the algorithm allocates the arrays on the heap using the `new` keyword. The second version which has been implemented allocates the arrays on the stack by using the unsafe `stackalloc` keyword.

Configurations The configuration of the following tests are described in Table 3.1, Table 3.2 and Table 3.3.

Conclusion There are just minor differences between the execution times with the

Table 3.2: Configuration B

Parameter	Value
Number of coordinates	1000
Maximal x value	100
Maximal y value	100
Number of iterations	1000

Table 3.3: Configuration C

Parameter	Value
Number of coordinates	10000
Maximal x value	100
Maximal y value	100
Number of iterations	1000

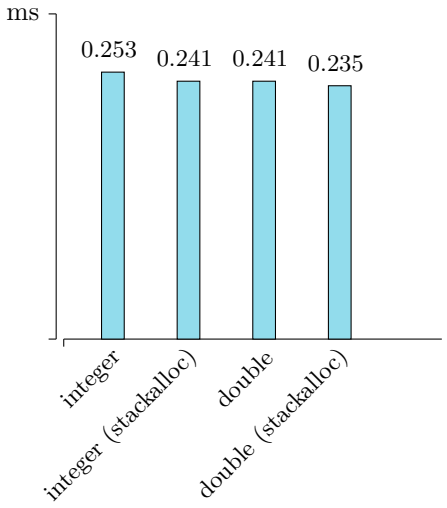


Figure 3.19: Execution times of the Natural Cubic Spline Algorithm with 100 coordinates (for details see Table 3.1) as input data.

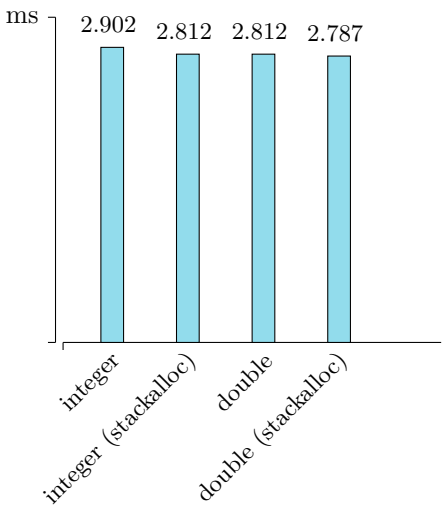


Figure 3.20: Execution times of the Natural Cubic Spline Algorithm with 1000 coordinates (for details see Table 3.2) as input data.

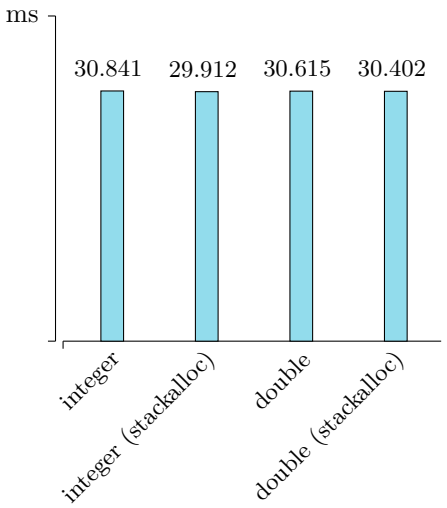


Figure 3.21: Execution times of the Natural Cubic Spline Algorithm with 10000 coordinates (for details see Table 3.3) as input data.

Parsing	**4** µs/request
Collision Detection	**3** µs/devices
Collision Resolution	**0.8** µs/obstacle
Splining	**3** µs/waypoints
Dispatching	**n/a**

Figure 3.22: Throughput measurements on a 2005 Intel Pentium D 2.8 GHz

different configurations. `stackalloc` is not used to calculate natural cubic splines and `double` is used as data type.

3.2.6 Run Time

The elements in Figure 3.22 affect the run time performance.

The different processing steps fulfill the posed requirements. As not many waypoints get generated in a typical robotic environment, fast computation of collision free paths can be expected. The dispatching overhead will depend on the concrete implementation.

Parsing

The parse subsystem must map the given identifier to the internal device driver and resolve the properties specified. In an environment with five devices, the command "move ID=abc x=1cm y+=0.3m" takes 4 µsto be converted into a `MoveRequest` class.

Collision Detection

In an environment with five devices, compute possible collision moments with other devices takes 3 µs.

Collision Resolution

Given a collision moment and an obstacle, evasion points can be calculated within 0.8 µs.

Splining

It takes a different amount of time depending on the number of coordinates given. A set of ten waypoints can be computed in 3 µs.

Move Dispatching

The time required to dispatch a set of waypoints depends on the actual implementation.

3.2.7 Limitations

Acceleration Ratio

The acceleration ratios of the involved devices are taken into account while the path is computed. However, during a move the speed cannot be changed until a waypoint is reached. Supporting this behavior would require a recomputation

3.2.8 Problems

Natural Cubic Splines Algorithm

While evaluating the natural cubic splines algorithm on Wikipedia[3], it turned out that the listed algorithm contained errors in the definition. The algorithm has then been corrected by the team and the new solution submitted to Wikipedia[4][5]

[3] Spline (mathematics)
[4] Corrections made to the algorithm by the team
[5] Notes on the discussion page explaining the changes made by the team

Chapter 4

Algorithm Analysis

In this chapter generic path finding algorithms are described and evaluated in consideration of the target system. Basically, path finding algorithms can be classified into following three categories.

Uninformed Search Uninformed search algorithms can be realized in a generic way. This makes them applicable for a wide range of problems. Their downside is the large amount of time required to perform simple searches in large search spaces.

Graph Search A graph search algorithm uses graph traversal to find a given node in a graph.

Informed Search Informed graph search algorithms use a problem specific heuristic as guiding help. Hence, good heuristics boost up the performance of an informed search and normally lets them outperform uninformed searches. [3].

4.1 Dijktra's Algorithm

Dijkstra's Algorithm is a graph search algorithm that can be used on weighted graphs. It computes a vector representing the shortest path.

Search Area

The search area is defined as a directed and weighted graph where edges have non-negative path costs.

Node Types

Unvisited Nodes Nodes which either have not been checked yet or are not considered for further investigation in the actual iteration due to a large distance value.

Visited Nodes Nodes whose neighbors have been checked and it is sure that the distance value is minimal and will not be overwritten.

Node Information

The only information a node holds is a distance value which equals the cost to reach that node. That value will be continuously updated by the algorithm.

Before the algorithm starts to search for the shortest path, it sets the distance value of the starting node to 0 and all others to $\infty+$.

Path Scoring

For calculating the path from a node A to another node B, the distance value of A and the cost for traversing the connecting edge are required. The actual distance value of B is updated only if the sum of the distance value of A and the path cost for traversing the connecting edge is less than the distance value of B:

$$dist_B = min(dist_B, dist_A + edgecost_{A \to B})$$

with $dist_B$ as the distance value of B.

Approach

Dijkstra's algorithm advances according to following steps:

1. Set the starting node's distance value to 0 and all other to $\infty+$.
2. Mark all nodes as unvisited and set the starting node as *current node*.
3. Calculate the distance of the current node's unvisited neighbors[1].
4. Mark the current node as visited if all neighbors have been checked and the distance value of every neighbor has been calculated.
5. Set the unvisited node with lowest distance value as next *current node*.
6. Repeat steps 3 to 5 until the goal node is marked as visited.

Example Search

Environment

During the exemplary search a simple graph consisting of four nodes is used (Figure 4.1)

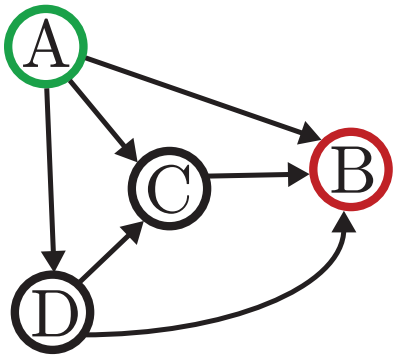


Figure 4.1: The predefined search graph with node *A* as the starting node and *B* as goal node.

[1] A neighbor node is a node that can be reached directly from the current node.

Caption

Following captions are used during the example:

- Search area (graph):
  - Visited blue (blue)
  - Starting node (green)
  - Goal node (red)
- Table:
  - Visited node (bold)

Procedure

For the example the node *A* is selected as starting node and *B* as goal node. The algorithm starts with setting the distance values of every node in the graph (0 for the starting node, $\infty+$ for all other nodes). The next step is to calculate the distance values of A's neighbors. The distance value for the neighbor node B is 4, since the edge cost for A $\rightarrow$ B is 4 and the actual distance value of B, $\infty+$, is bigger than 4. For the other neighbor nodes of A, C and D the following distance values are calculated:

Step	A	B	C	D
0	**0**	$\infty+$	$\infty+$	$\infty+$
1	**0**	20 via A	10 via A	7 via D

The next step is to move forward by selecting the node with the lowest distance value from the list of unvisited nodes, that is node *D*. The algorithm checks the neighbor nodes of D which results in following distance values.

Step	A	B	C	D
0	**0**	$\infty+$	$\infty+$	$\infty+$
1	**0**	20 via A	10 via A	7 via A
2	**0**	*17* via D	10 via A	**7 via D**

Due to the simple procedure of the algorithm, the next steps are shown incrementally in the following tables.

Step	A	B	C	D
0	**0**	∞+	∞+	∞+
1	**0**	20 via A	10 via A	7 via A
2	**0**	17 via D	10 via A	**7 via D**
3	**0**	*12* via C	**10 via A**	**7 via D**

Step	A	B	C	D
0	**0**	∞+	∞+	∞+
1	**0**	20 via A	10 via A	7 via A
2	**0**	17 via D	10 via A	**7 via D**
3	**0**	12 via C	**10 via A**	**7 via D**
4	**0**	**12 via C**	**10 via A**	**7 via D**

The algorithm added the goal node B to the list of visited nodes in the last step, signalizing that a shortest path to B has been found. The next step consists of reconstructing the shortest path to reach B. This is done by following back the path by going from *via* to *via* until the starting node A is reached. In the 4th step the *via* in the column for the node B points to C. In the 3rd step the *via* located in the column for node C points to the starting node A, so the shortest path to reach the goal node B from the starting node A is: **A** → **C** → **B**.

Conclusion

Dijkstra's algorithm is an efficient algorithm because of its low memory usage and simplicity. However, it is not suitable for the problems of this project. Mainly because the target environment and its dynamic behavior cannot be modeled in a graph since basically the number of platform states is unlimited.

4.2 A* Search Algorithm

A* search algorithm (A*) is an informed search that uses best-first search and heuristic values to find a shortest path. This section describes A* [2] and contains an example demonstrating its functionality.

Search Area

The first step of the algorithm is to simplify the search area. By dividing the search area into squares a two dimensional matrix gets created. Squares may have the status: *walkable* or *not walkable*. Examples of not walkable squares are walls, water and other illegal terrain.

Because the search area can be divided in shapes like triangles or rectangles, the term *node* is used to refer to those items in.

Node Types

The nodes of a search area can be classified with following three types:

Unknown Nodes Nodes that are not discovered yet. It is still unknown how to reach them.

Known Nodes Nodes to which a (maybe suboptimal) path is known. They are saved in the *open list* and marked with an F score.

Checked Nodes Nodes to which the shortest path is known. These nodes are saved in the *closed list* and need no further investigation.

Two lists, *open list* and *closed list* are used to track the two node classes mentioned last. *Open list* contains nodes that were discovered but have not been checked yet. *Closed list* contains nodes which were transferred from *open list* once their shortest path got computed.

Initially, *open list* contains only the starting node, *closed list* is empty.

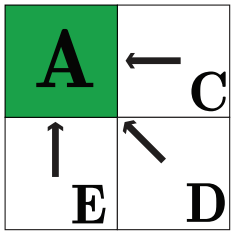


Figure 4.2: A is the starting node. D can reach A with a diagonal move, whereby C and E can reach A with an orthogonal move.

Node Information

All information a node holds is a pointer to the parent node and its F score. In a square grid, there are two types of pointers: *diagonal* and *orthogonal (horizontal or vertical)* ones. This distinction is important, because a diagonal pointer is weighted differently than an orthogonal one while calculating the shortest path.

Path Scoring

Following equation is used to compute the F score:

$$F = G + H$$

G is the cost to move from the starting node A to a given node on the grid. It does not have to be an adjacent node.

To calculate the shortest path, two different values for each direction (diagonal or orthogonal) are required. These two numbers can be floating point numbers, but for simplicity and for faster calculations it is recommended to use integers.

The cost of a diagonal move is weighted with 14 and the cost of an orthogonal move with 10. To calculate G along a path, the G cost of the parent has to be incremented by 10 if the parent node can be reached with an orthogonal move or by 14 if it can be reached with a diagonal move. For example refer to Figure 4.2: A is the parent node of C. Since A is the starting node and C can reach A with an orthogonal move, the G cost of C is $G_C = G_A + G_{orthogonal} = 0 + 10 = 10$.

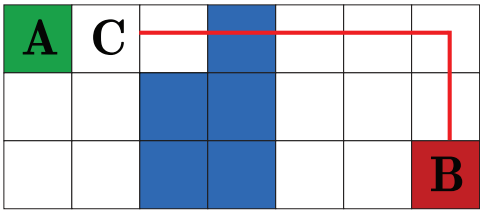


Figure 4.3: An example on how to calculate the H cost using the Manhattan method. To calculate the H cost for node C the number of orthogonal moves (red line) that are needed to be taken to reach the final node B have to be counted and multiplied by 10 (cost for an orthogonal move). The blue squares are obstacles and they are ignored by the Manhattan method. To reach the final node B from C seven orthogonal moves are needed (four horizontal moves and three vertical moves), so that $H_c = 7 * 10 = 70$.

H is the cost to move from a given square to the final destination. H is a heuristic valued, which can be calculated by using different methods. Here, one heuristic, the Manhattan method[2] is used (Figure 4.3).

The Manhattan method calculates the total number of orthogonal moves which are necessary to reach the target square from the current square and multiplies the total number with the selected value of 10 for orthogonal moves. The method ignores possible diagonal movements and obstacles

Finally, F will be used to select the node with the lowest score from *open list* to continue the search for the shortest path.

Approach

1. At the beginning, the starting node is added to *open list*

2. The node in *open list* with the lowest F score is set as *current node*)

3. The current node is moved to *closed list*

[2]The Manhattan method is also known as the Manhattan distance

4. For every adjacent:
   (a) Ignore if already in *closed list*.
   (b) Add to *open list* if not already there yet and set the current as its parent node.H, G and F are computed.
   (c) If already in *open list*, check if the path from the current node to that node is better. This can be realized by adding the cost to reach the node (either orthogonal or diagonal) to the current node's G cost. If G_{new} is lower than G_{node}, G_{node} is updated with it and the current node is set as new parent node. If the G cost of the adjacent node is higher than the sum of the current node and the cost for reaching the adjacent node (either 10 or 14), the parent node of the adjacent node is set as current node. The G cost is set to $G_{current} + G_{cost}$. Since G cost changes, the F value of the adjacent node must be updated accordingly.
5. The procedure can be stopped when
   (a) The target node has been added to *closed list* which means a path has been found
   (b) *open list* is empty, which indicates that no path could be found
6. The shortest path is computed by tracking backwards through the parent nodes until the starting node is reached.

Example Search

An often used word in this section is *Inspection Block*, which represents a node N and its adjacent nodes which must be inspected.

Environment

The search area used for this example search is shown in Figure 4.4.

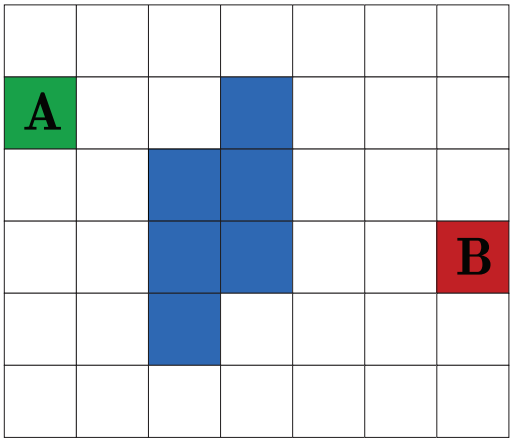


Figure 4.4: A is the starting square (green-colored), B is the final square (red-colored). The blue-colored squares are obstacles.

Caption

Following patterns are used:

Node in the open list Square with an orange border

Node in the closed list Square with a red border

Unknown node Square with a black border

Obstacle Square in blue

Starting node Square in green

Final node Square in red

Procedure

The start node A is added to the *closed list* because *open list* is empty anyway; it has the lowest F score. Then, all adjacent nodes are added to *open list* because they are no obstacles. A is set as parent node of every adjacent (Figure 4.5)

To calculate G, H and F for the adjacent nodes, the G cost must be calculated first. As mentioned in 4.2, an orthogonal move costs 10 and a diagonal one 14. The resulting G costs for the adjacent nodes are shown in Figure 4.6 as well as the H cost

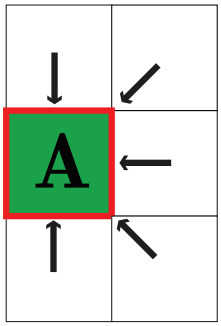


Figure 4.5: The first step of finding the best path. Start node A is in the *closed list* (red border). The adjacent nodes point to A which means A is their parent node.

which is computed according to the Manhattan method. The F values are shown in Figure 4.8.

Moving Forward

Now that the adjacent node's G, H and F cost are known, the search goes on by selecting the node with the lowest F score which is the node at the bottom right with $F = 74$. It is moved to the closed list and called *N1*. Its block (Figure 4.9) is inspected:

First the two unknown nodes are added to the open list and the G, H and F values are calculated. For the first unknown node G is 14 because a diagonal move is necessary to reach node N1. H 70, because seven orthogonal moves (three horizontal moves plus four vertical moves) are necessary to reach the final node B. The F score is simply the sum of the previously computer values, 84. The costs of the second unknown node are G=10, H=60, F=70.

After the costs have been calculated, their corresponding nodes are added to the open list.

The adjacent node right under our current node N1 is checked as next. The G cost of that node is 14 and $G_{N1} = 10$. The cost of an orthogonal move must be added to G_{N1} since an orthogonal move is necessary to reach that node. This results in $G_{new} = G_{N1} + G_{orthogonal} = 10 + 10 = 20$ and since

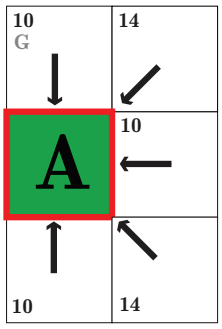


Figure 4.6: G cost of each adjacent node (upper left value of each node). The G cost of the nodes is calculated by adding 10 for an orthogonal move or 14 for a diagonal move to the G cost of the parent node. Due to the fact that the G cost of the parent node - which in this case is also the starting node - is 0, so the G cost of an adjacent node is either 10 or 14.

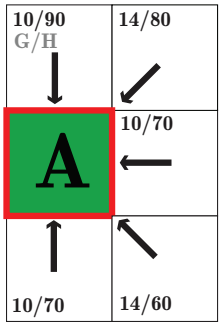


Figure 4.7: G and H cost of each adjacent node (upper left value of each node). The H cost of the nodes is calculated by using the Manhattan method (refer to Figure 4.3). For example, the H cost of the most right adjacent node is 70, because seven orthogonal moves (four horizontal moves plus three vertical moves) are necessary to reach the final node B. $H = 7 * 10 = 70$.

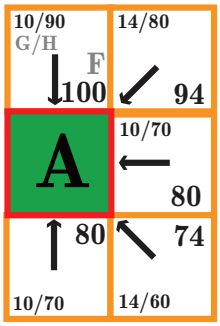


Figure 4.8: F core of each adjacent node. After calculating the F score by adding the G cost + H cost the adjacent nodes are added to the open list (orange border).

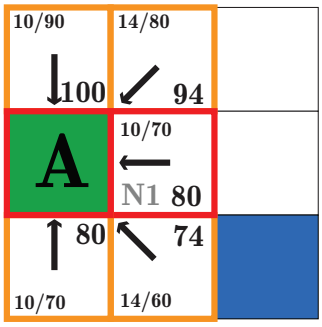


Figure 4.9: The new inspection block of the node N1 is shown. The inspection block contains four nodes from the open list, two nodes from the closed list, two unknown nodes and one obstacle. Our actual position is the node N1 on the right of the starting node A.

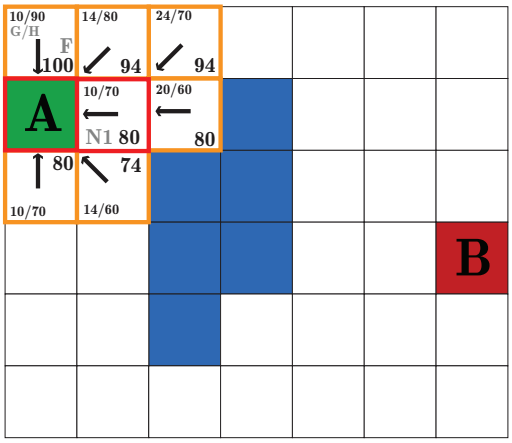


Figure 4.10: Search area after the second step **point to section**.

this is not a better path ($(G_{node} < G_{new})$) no costs need to be updated and N1 will not be set as new parent node of the adjacent node. For the other three adjacent nodes, no costs need to be because no path improvement can be achieved.

Updating an Established Path

To show how a path is updated, the search is moved forward and three steps are skipped - adding *N2*, *N3* and *N4* to the closed list.

N4's inspection block (Figure 4.12) contains four nodes which can be ignored because they are already in the closed list.

The remaining node is inspected by pointing it to N4 instead of node A: N4's G cost is 10 and since an orthogonal move is necessary to reach that node, 10 must be added to its G cost ($G_{new} = G_{N4} + G_{orthogonal} = 10 + 10 = 20$). Comparing the current G value (28) with G_{new} (20) shows that a better result is achieved by setting node N4 as new parent node (Figure 4.13). Now that the node's G cost has changed, the F score needs to be updated ($F_{node} = G_{new} + H_{node} = 20 + 60 = 80$).

Obstacles

Moving a step forward the algorithm gets to choose between two nodes in the open list

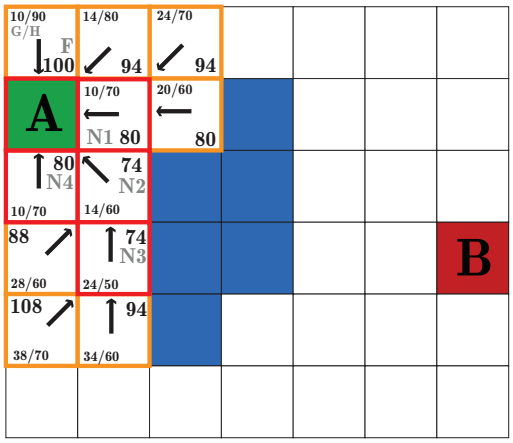


Figure 4.11: Search area after the fifth step, before checking the adjacent nodes of node N4.

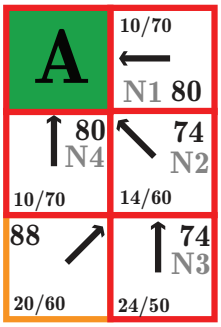


Figure 4.12: The inspection block for node N4 consisting of 4 nodes already contained in the closed list and one node contained in the open list, for which the path has to be checked.

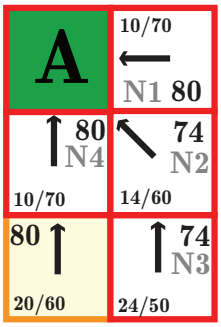


Figure 4.13: The inspection block for node N4 after applying the score and path updates to the highlighted node below N4.

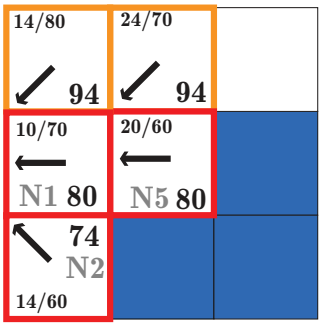


Figure 4.14: The inspection block for node N5 is shown in the image.

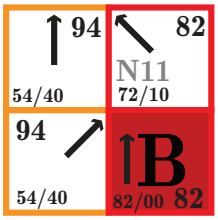


Figure 4.15: The final node B has been added to the closed list. That means that the best path has been found and the search algorithm can terminate its search.

that have a F score of 80 which is the lowest F score. The rightmost node is selected and named *N5* (Figure 4.14)

The unknown node in the upper right can be ignored because it cannot be reached with a diagonal move from N5 because a cut across an obstacle would be required to accomplish this. The node is ignored since it is impossible to reach the adjacent node directly. Checking the remaining nodes in the inspection block results in no changes which means the search can continue.

Reaching the Target

The search process is repeated until node B is put into the closed list which means the algorithm found the best path (Figure 4.15).

The shortest path can then be computed by moving from parent node to parent node beginning at B until node A is reached. (Figure 4.16).

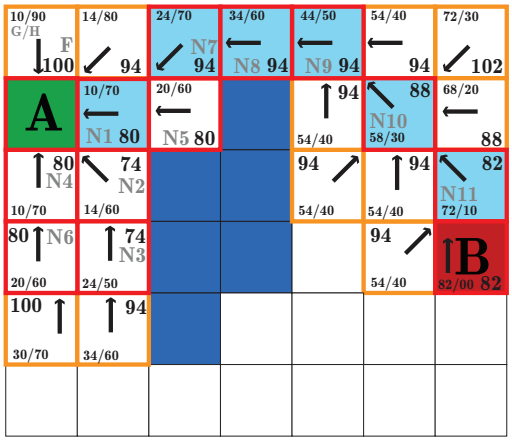


Figure 4.16: The best path is highlighted with blue and can be determined by following the arrows back from the final node B to the starting node A.

Conclusion

A* is an intelligent algorithm which is capable to handle a given environment. The big amount of memory it uses to keep track of nodes is not a big issue since it would run on a modern desktop machine.

However, modeling a three dimensional representation of the platform model creates a huge amount of cells which must be traversed. Estimations have shown that it will not be possible to find a path within the time requirement given. Additionally, the map must be constantly updated as other objects may change their positions on every iteration, creating a fourth dimension of the already big model.

4.3 Iterative Deepening A*

The IDA* is a variant of the A*, which uses less memory thanks to its methodology of not using lists to track visited nodes. Since IDA* is a variant of A*, IDA* differs only in a few points from the A*, but shares its advantages.

IDA* Compared to the A* Search Algorithm

Before giving a detailed description of IDA*, a comparison with the A* is made to show the differences.

Starting with the simplest difference, IDA* does a depth-first search whereas A* does a best-first search. IDA* itself is a variant of the Iterative deepening depth-first search (IDDFS) but other than IDDFS, IDA* uses the F score (heuristic value) as a threshold and not a specified depth limit.

Doing a depth-first search has the advantage of having storage linear to the length of the shortest path. Along with this, IDA* does not use a closed or an open list like A* does. Hence its memory usage is lower. Even if this sounds like an advantage, it is also a disadvantage due to the fact that IDA* ends up re-visiting nodes many times since it does not track nodes that have been visited.

IDA* uses a left-right traversal of the *search frontier*, whereas A* maintains the search frontier in the open list in a sorted order. This is an advantage of the A* as it knows where to continue the search by selecting the next node from the open list in a best-first manner. IDA* instead has to start iterating from the beginning to reconstruct its search frontier, avoiding A*'s memory problem of maintaining the open list but slowing down the search process. There is a variation of the IDA* called the Memory-enhanced IDA* (ME-IDA*) which makes the usage of a transposition table. The transposition table, typically implemented as a hash table, stores search results. ME-IDA* uses the transposition table to query it, checking if further search in the sub-tree of the queried node is necessary or unnecessary.

Search Area

For describing IDA* and especially showing the usage of the depth-first search a simple tree is used (see Figure 4.17)

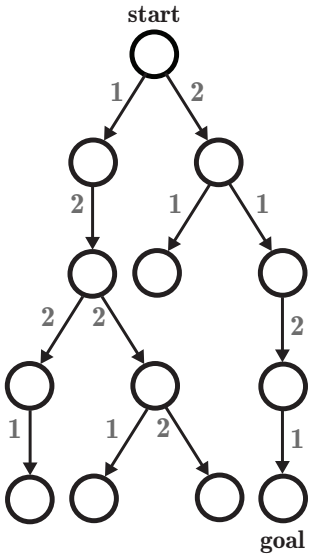


Figure 4.17: The predefined search area for the IDA* example search. The start node and goal node are marked.

Each edge in the tree is labeled with a path cost of 1 or 2. These values are needed for calculating the G cost of a specific node.

Node Types

The nodes in the search tree are classified into two types by IDA*:

Visited Nodes Nodes that have been visited but the algorithm has proven that no solution is possible with the actual set threshold (described later) and stopped its search there.

Expanded Nodes Nodes that the algorithm has checked successfully and have an F score lower than the threshold.

Path Scoring

Like the A*, IDA* makes use of the F score, which is calculated in the same way as A* does:

$$F = G + H$$

G is the cost of the path from the start node to the current node. As mentioned above, each edge is labeled with a path cost.

H is a heuristic estimate defined by the number of moves required to reach the goal of the actual tree. An actual move has a cost of 1.

IDA* defines beside the named two values another important value:

$$F_{limit}$$

which is a threshold and is used to stop the search, without searching further in the subtree once a node is found whose F score is bigger than the defined threshold ($F_{node} > F_{limit}$). The threshold is incremented by 1 if no goal node has been found in the actual iteration and a new search is performed, starting from the start node and rebuilding the search frontier from scratch. Since IDA* does not track visited nodes a new search means to repeat the previous iterations and continue the search. Regardless of this the threshold F_{limit} is a benefit. It avoids that IDA* searches to deep in the tree. This saves a lot of time in execution, signalizing early that a deeper search is unprofitable and to continue the search elsewhere.

Search Algorithm

This is a description of the steps IDA* makes to find a shortest path to the goal node. An example search is presented in the subsection *Example Search* of section 4.3.

1. Define the threshold F_{limit}
2. Do an IDDFS with threshold F_{limit} as limit
   (a) Expand a node if $F_{node} \leq F_{limit}$
   (b) Do not expand a node and mark it as visited if $F_{node} > F_{limit}$.
   (c) Stop the search if goal node has been found. Else continue the search on the subtree of the last node with an alternative path if a node has been found with $F_{node} >$

F_{limit} (standard behavior of IDDFS).

3. If Fringe Search (FS) has finished its search (using IDDFS) and has not found the goal node with the actual threshold F_{limit}, increase F_{limit} by 1 and restart IDDFS from the beginning (start node) with the new threshold.

4. If the goal node has been found, save the best path by backtracking it.

Example Search

This section shows how IDA* works by using a predefined search area defined Figure 4.17 to show its path finding process step by step. The path costs for each step are already set in each edge of the tree. The heuristic value H is calculated by summing up the moves needed to reach the bottom of the tree with a constant move cost of 1. The G cost is defined as the cost to get from the start node to the current node, calculated by adding the costs set at the (traversed) edges in the tree.

Caption For The Search Area The following distinct marks are used to mark special nodes:

- Node with black background: Expanded node
- Node with gray background: Visited node
- Node with white background: Unvisited node

The Beginning The threshold is set to $F_{limit} = 4$. The search starts by first marking the start node as expanded. If a node is marked as expanded it means that its F score is less than the threshold F_{limit} and the search can continue in its subtree (due to the fact that it has been *expanded*). As a next step the node in the left subtree of the start node is examined. The G cost of the current node is $G_{node} = 1$, since only one edge with a cost of 1 has to be traversed to get from the start node to the current node. The H cost is $H_{node} = 3$ since three moves (each weighed with 1) are needed to reach the bottom of the tree (or the actual part of the tree). The F score for this node is $F_{node} = G_{node} + H_{node} = 1 + 3 = 4$ and since $F_{node} \leq F_{limit}$ the node is marked as expanded and the search is continued in the current node's subtree. The next node is selected for examination. For this node the following values are calculated:

- $G_{node} = 3$
- $H_{node} = 2$
- $F_{node} = 5$

Due to the fact that its F score is greater than the threshold F_{limit} the node is marked as visited, but the search is not continued in its subtree. A characteristic of the depth-first search is (since IDA* is a variation of the IDDFS, and IDDFS uses a depth-first search), that if no goal node is found in the subtree with $F_{node} \leq F_{limit}$ the algorithm goes back to the node from which an alternative path is possible. In the current situation this is the start node, so the search continues from there, checking the node in its right subtree. The following values are calculated for the current node in the right subtree of the start node:

- $G_{node} = 2$
- $H_{node} = 3$
- $F_{node} = 5$

The F cost of this node excesses the threshold F_{limit} too. Since the algorithm cannot find an alternative path by revisiting the start node, IDA* finishes its first iteration and increases the threshold F_{limit} by 1, so that $F_{limit} = 5$. The tree in Figure 4.18 shows the expanded and visited nodes in the first iteration.

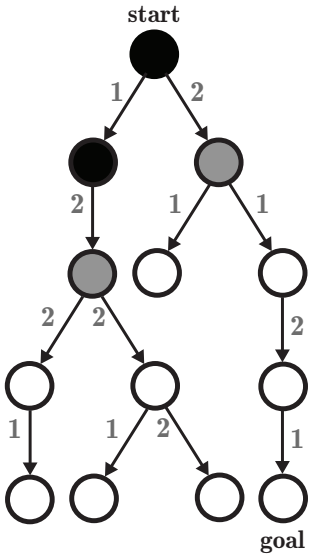


Figure 4.18: The tree after the first iteration with a threshold set to $F_{limit} = 4$. Two nodes have been expanded and two nodes have been marked as visited.

Second Iteration Now that the first iteration has ended without any feasible results, the search restarts from the start node with a new threshold $F_{limit} = 5$. Since the algorithm will mark the same nodes as expanded like the iteration before, the search continues in the tree shown in Figure 4.18[3]. That is the search will continue in the start node's left subtree, since depth-first search would first take this path when restarting the search process. The visited node in the left subtree of the start node is checked first. For this node the following values are calculated:

- $G_{node} = 3$
- $H_{node} = 2$
- $F_{node} = 5$

[3]This is not a characteristic of IDA*, since it does not held track of the nodes that are part of the search frontier. This is only a simplification for not having to repeat the steps made in the first iteration in this paragraph.

This node is marked as expanded and since the node is a branch, its left subtree is selected for further inspection. The whole search process is repeated until all nodes with an F score less than the threshold F_{limit} have been checked. The new tree is shown in Figure 4.19.

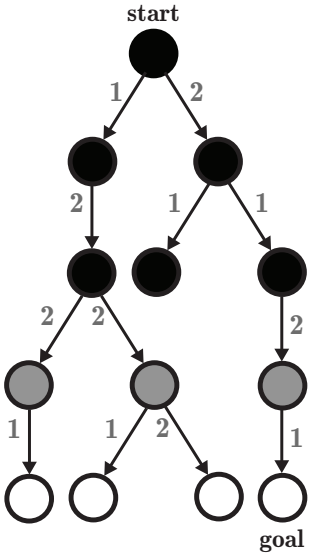


Figure 4.19: The tree after the second iteration with a threshold set to $F_{limit} = 5$. Six nodes have been expanded and three nodes have been marked as visited.

Third Iteration: Finding The Goal Node The tree shown in Figure 4.20 is the final tree with the goal node marked as visited, signalizing that the search was successful. A total of seventeen nodes have been expanded and twenty six nodes have been visited. The start node has been expanded for three times (pro iteration once).

Conclusion

Although the IDA* has the advantage of using less memory than the A*, it suffers from the same problem: it is not possible to model the target environment using nodes.

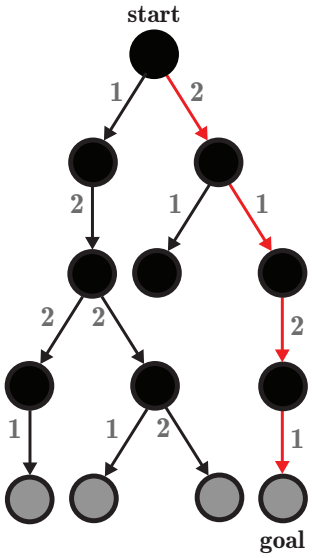


Figure 4.20: The tree after the third and final iteration with a threshold set to $F_{limit} = 6$. The goal node has been found and the shortest path has been marked with.

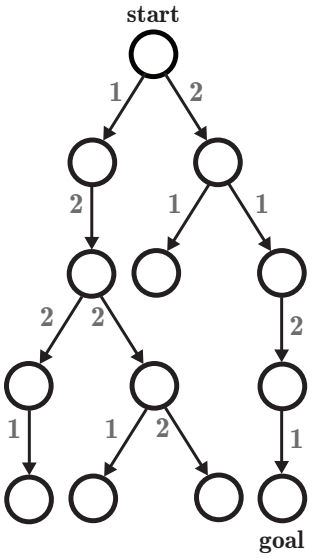


Figure 4.21: The predefined search area for the FS example search. The start node and goal node are each marked.

4.4 Fringe Search

The FS [5] is a search algorithm that adapts most of the characteristics of the IDA* (e.g. usage of a threshold), but makes improvements in its execution due to the fact that it iterates over the search frontier (*fringe*) and continues its search from there, without re-visiting all nodes from the start in each iteration.

It is shown that FS runs roughly 10-40% faster than a highly-optimized A*, searching for a shortest path on a grid.

Search Area

The predefined search area for FS is exactly the same as the search area defined for the IDA*. For the sake of convenience the same tree is shown also in this section (see Figure 4.21).

Node Types

The node types are the same as described in *Node Types* (IDA*) on page 66, namely *visited nodes* and *expanded nodes*. Although IDA* does not use any lists to track nodes, FS uses two lists for this purpose. One list is called *now* and is used for the actual iteration to track nodes that have to be checked with the set threshold and eventually are expanded. The other list is called *later*, is used for the next iteration and consists of nodes that have to be checked due to the fact that the nodes have a higher F score as the set threshold in the actual iteration. The *new* list initially contains only the start node (also called the root node) and the *later* list is empty. Both lists concatenated can be considered as an open list like the one A* uses, but in FS's case the list is not sorted and this is a performance advantage due to the fact that no execution time is used to sort the list.

Path Scoring

The calculations for the path scoring are exactly the same as described in *Path Scoring* (IDA*) on page 66.

Search Algorithm

The following is a short step by step description of the FS algorithm is presented below. An example search can be found after the algorithm description.

1. Define the threshold F_{limit}
2. Put the start node in the *now* list
3. Do a search using IDDFS with threshold F_{limit} as limit
   (a) Until the *now* list is not empty, check the node at the head of the *now* list
   - If $F_{headnode} > F_{limit}$ remove the head node from the *now* list and add it to the *later* list
   - If $F_{headnode} \leq F_{limit}$ the children of the head node have to be checked. Add the head node's child nodes to the front of the *now* list and discard the head node from there.

   (b) Stop the search if goal node has been found. Else continue the search on the subtree of the last node with an alternative path if a node has been found with $F_{node} > F_{limit}$ (standard behavior of IDDFS).
4. If FS has finished its search (or the iteration) and has not found the goal node with the actual threshold F_{limit}, increase F_{limit} by 1 and restart IDDFS from the beginning (start node) with the new threshold.
   - Copy the nodes of the *later* list to the *now* list and set the *later* list to empty.
5. If the goal node has been found, save the best path by backtracking it.

Example Search

In this section a step by step example search using the FS is shown. The predefined search area used for this example is shown in Figure 4.21. Just like in the IDA* example search, the path costs for each step are already set in each edge of the tree. The heuristic value H is calculated by summing up the moves needed to reach the bottom of the tree with a constant move cost of 1. The G cost is defined as the cost to get from the start node to the current node, calculated by adding the path costs marked at the (traversed) edges in the tree.

Caption for the Search Area For the nodes in the predefined search (see Figure 4.21) the following marks are used:

- Node with black background:
  Expanded node positioned in the *new* list
- Node with gray border:
  Visited node positioned in the *later* list
- Node with white background:
  Unvisited node

First Iteration The search process begins with a threshold set to $F_{limit} = 4$. The *new* list contains one node, namely the start node and the *later* list is empty at the beginning. The search starts by inspecting the *head* node of the *new* list. Since this is the start node its children are added to the *now* list and the head node is discarded from there. The search moves on by inspecting the new head node. Since it is not specified in which order the nodes have to be inserted into the *new* list, it is assumed that the left child of the root node is the new head node. The following values are calculated for the head node:

- $G_{headnode} = 1$

- $H_{headnode} = 3$
- $F_{headnode} = 4$

Since the F score of the head node is less than the actual threshold $F_{limit} = 4$ the child of the head node can be added to front of the *new* list and the head node is discarded. The values of the new head node are defined as follows:

- $G_{headnode} = 3$
- $H_{headnode} = 2$
- $F_{headnode} = 5$

The head node's F score excesses the threshold F_{limit} and so it is moved to the *later* list. The new head node of the *new* list is inspected and the mentioned steps are repeated until the *new* list is empty and all nodes with an F score below the threshold are checked. The tree after the first iteration is shown in Figure 4.22.

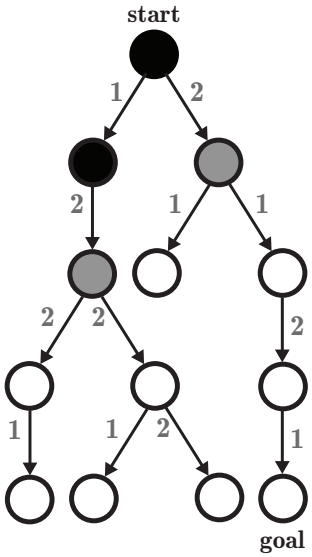


Figure 4.22: The tree after the first iteration with a threshold set to $F_{limit} = 4$. Two nodes have been expanded and two nodes have been marked as visited.

Second Iteration Before starting with the second iteration, the threshold is incremented by 1, resulting in $F_{limit} = 5$. Also, the nodes in the *later* list are moved to the *new* list, so that the *later* list is empty. The *new* list now contains two nodes from which the FS algorithm will continue its search. Again, the search starts by checking the actual head node. That is the node in the left subtree of the root node marked as visited (gray background) in Figure 4.22. The costs for this node are the same as calculated in the first iteration:

- $G_{headnode} = 3$
- $H_{headnode} = 2$
- $F_{headnode} = 5$

Since the threshold is increased, the node's F score now fits in the set threshold $F_{limit} = 5$. Due to this fact its child nodes are added to the front of the *now* list and the head node is removed from that list. The next steps are skipped and the tree, after the second iteration is finished, is shown in Figure 4.23.

Third Iteration For the third iteration the step by step description is skipped, since it is exactly the same routine as described in the first and second iteration. The only thing that changes is the threshold, incremented by 1, so that $F_{limit} = 6$. Instead the result tree, including the shortest path, is shown in Figure 4.24.

The FS algorithm expanded a total of eight nodes and visited seventeen nodes. This is a big improvement compared to the nodes IDA* expanded and visited in the same tree (IDA* expanded seventeen nodes and visited twenty six nodes).

Conclusion

FS is the definitely the most advanced search algorithm of the A* family, since it has an advantage compared to the IDA*: it mantains its search frontier in a list, avoiding to revisit after every iteration previous nodes

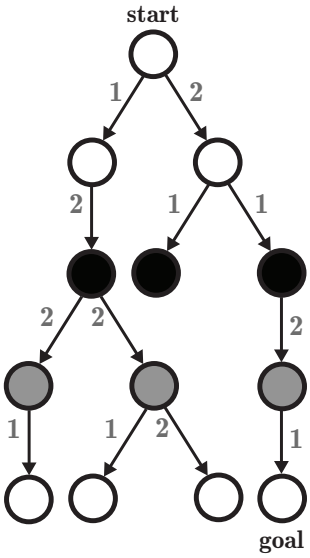


Figure 4.23: The tree after the second iteration with a threshold set to $F_{limit} = 5$. Three nodes have been expanded and three nodes have been marked as visited.

that already have been checked. The problem FS has is that it cannot be used to model the target environment, due to the complexity of the latter.

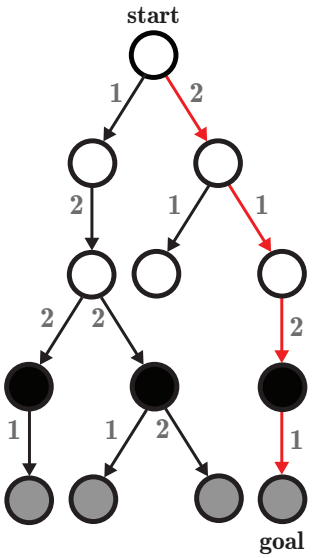


Figure 4.24: The tree after the third and final iteration with a threshold set to $F_{limit} = 6$. The goal node has been found and the shortest path has been marked with red.

4.5 Potential Field

Objects are expanded with a force comparable to magnetic ones. The device is attracted by the target's force. During the movement it avoids obstacles by being repulsed by them.

This means the moving device solely reacts to the current environment and has no predefined move strategy. Forces based on obstacles influence the moving object while it travels through the environment. This requires the moving device to continuously interpret the current state and perform according to it. Thus no central logic is required.

However, the moving object might get stuck when forces compensate each other.

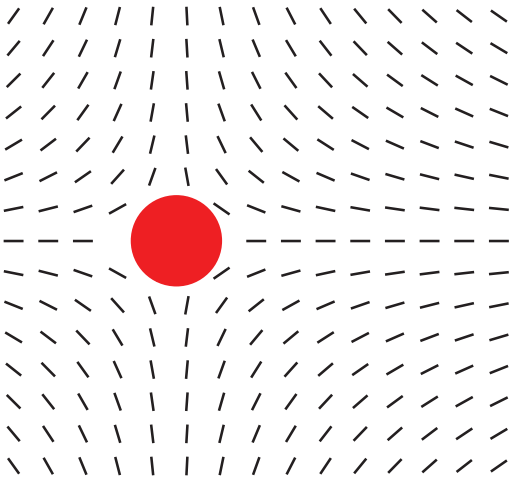


Figure 4.25: An obstacle (red) and the potential field that surrounds it. The forces repulse the moving device (not shown) from the obstacle.

Conclusion

Although the approach seems really interesting, it is not feasible because objects can change their sizes. Also, dynamic behavior can not be modeled.

4.6 Conclusion

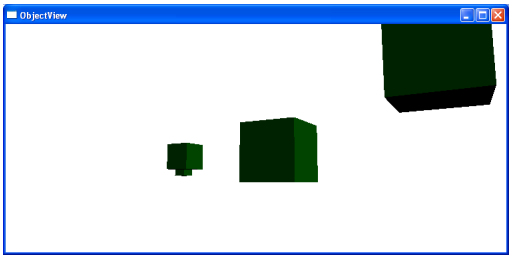
During the Algorithm Analysis many interesting algorithms have been found and evaluated. However, because of the target environment's high complexity, all approaches turned out to be not applicable.

Chapter 5

Object Viewer Tool

5.1 Introduction

The developed 3D Engine interacts with objects that have different representations. To test algorithms and their behavior in specific situations a viewer component is required which allows inspecting the current state of the engine by displaying the positions and sizes of the managed objects.



5.2 Functionality

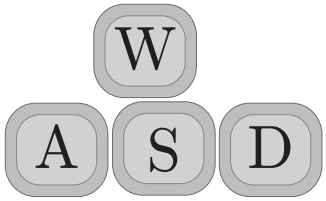
The viewer periodically loads the internal state of the engine, generates objects and displays them. The continuous reloads might seem inefficient but it has been decided that the computation overhead is negligible, especially as the viewer is no time critical component.

Because the engine was developed in C# the viewer is also implemented on .NET minimizing marshaling overhead. The engine's native classes are converted into WPF compatible 3D objects and displayed using the rendering capabilities of .NET.

5.3 Navigation

The object viewer implements standardized controls used in many physic engines and computer games. It is based solely on keys.

5.3.1 Move Commands



These commands change the camera's position.

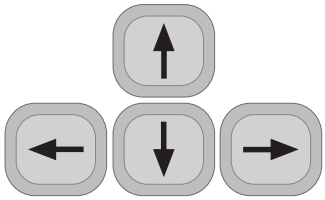
W Moves the camera forward to the looking direction

S Moves the camera backward keeping the looking direction

A Moves the camera and the looking direction to the left

D Moves the camera and the looking direction to the right

5.3.2 Turn Commands



These commands change the direction where the camera is looking at, keeping the camera's position.

↑ Raises the camera's looking direction

↓ Lowers the camera's looking direction

← Turns the camera to the left

→ Turns the camera to the right

5.3.3 Other Commands

Resets the camera's position and looking direction to the initial values

Chapter 6

Move Tool

6.1 Introduction

The developed 3D Engine exposes a generic and textual interface. In order to send requests to the engine in an easy manner, a tool has been developed allowing creation and controlling of arbitrary command.

6.2 Functionality

The Move Tool shows a list of devices known to the robotic engine. Those device names are used to identify concrete robotic arms on the system.

A move command must contain a device identifier and an arbitrary set of parameters which will be interpreted by the engine or the corresponding device driver.

Once a move request has been generated from a query and sent to the engine, it is appended to the list of pending moves and its state is periodically updated.

6.2.1 Define a Query

All queries entered will be allowed and sent to the engine. There is no data validation implemented. However, if input is not according to the query grammar[1] the engine might signal an error.

Examples of valid queries are

- `move dev_1 x=10mm`
- `move dev_1 x=10cm y=5000um`
- `move dev_2 x=10mm`

[1]Please refer to the Technical Report for more information

6.2.2 Auto completion

The query editor provided in the Move Tool implements an auto completion feature. It provides a selection for commonly used commands and parameters. For example, if `move` is entered, a list of devices can be selected from a pop-up menu. Suggestions are also implemented for often used parameters.

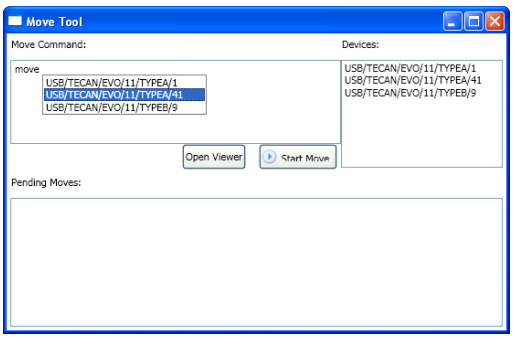


Figure 6.1: After entering the `move`, a list with of devices appears

6.2.3 Controlling Requests

Once a request could be generated from a query and has been accepted by the engine, it is displayed in a list of pending move requests which show the current move status.

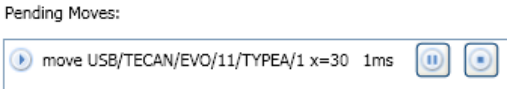


Figure 6.2: Once a move request has been accepted by the engine, it can be controlled with the corresponding buttons

Following operations to control a request are exposed to the user

- Continue Move
- Pause Move
- Abort Move

6.2.4 Object View

To visualize the current state of the engine, the already existing viewer component [2] was integrated into the tool. It can be started by clicking *Show Viewer*.

[2] For more information please refer to the Object Viewer chapter

Chapter 7

Spline Tool

7.1 Introduction

A tool has been developed that calculates the cubic polynomials for a natural cubic spline based on given coordinates. The Spline Tool plots the resulting graph afterwards.

7.2 Functionality

The Spline Tool requires the user to enter at least two coordinates in the form of `x1,y1 x2,y2`. After the required coordinates are entered, the Spline Tool first prints each cubic polynomial of the Spline S_i and subsequently plots the resulting graph.

7.2.1 Define The Coordinates

Two or more coordinates have to be defined in the form of `x1,y1 x2,y2`. After the minimum amount of coordinates has been entered, the calculation of the natural cubic spline can be started by clicking the *Calculate Natural Cubic Spline*-button or by hitting the `ENTER`-key.

It is possible to enter floating point coordinate values. The fractional digits of a floating point number shall be marked with a . (point). Example: `1.32,3 2,3.4123`.

It is possible to enter negative coordinate values. Example `1,2 3,-4`.

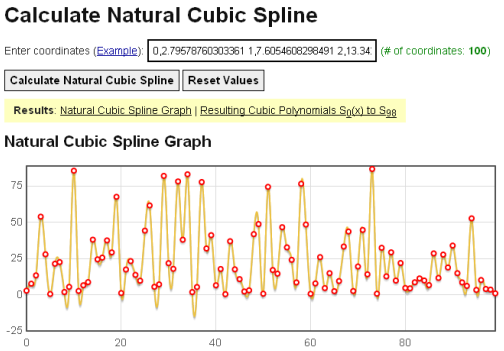


Figure 7.1: The Spline Tool plots the graph of the resulting natural cubic spline and prints the cubic polynomials.

7.2.2 Toggle On and Off Content

Since the goal of the Spline Tool is to provide the needed functionality in its simplest form, the parts *Description*, *Usage* and *Examples* can be toggled on or off. This can be achieved by clicking the yellow button next to each title. If the browser allows the usage of local cookies (refer to section 7.4 on page 80 for a detailed overview of the compatible browsers), depending on the action, the content of each part is either displayed or hidden when the Spline Tool is started. This functionality should permit the shaping of the graphical user interface of the Spline Tool as desired by the user. Since the usage of the Spline Tool is kept very simple and the displayed parts *Description*, *Usage* and

Examples are there if the user needs help or hints, the reason of the toggle functionality is that after the first utilization of the Spline Tool the user does not need them anymore and can hide them permanently.

7.3 Implementation

The Spline Tool has been implemented using HTML, CSS and JavaScript. For the JavaScript part of the Spline Tool, the jQuery[1] framework in version 1.3.2 is in used. For the generation of the graph, the jQuery plugin *flot*[2] is used. The Spline Tool has been implemented using JavaScript because of the simple usage and the high portability. To use the Spline Tool only a supported browser is required which allows JavaScript execution.

7.4 Browser Compatibility

It is recommended to use the Spline Tool either with a current version of the Firefox browser (version 3.0) or an current version of the Opera browser (version 9.64), since both support the full functionality of the Spline Tool.

Problems

Opera Compatibility

During the compatibility check of the Spline Tool on different browsers, a serious issue was found in the Opera browser. The cause of the issue was the usage of a timeout mechanism which was realized using `setTimeout` and `clearTimeout`. `clearTimeout()` was called with no parameter but would required the id received while calling `setTimeout`. After solving the problem, the manual tests for Opera could be run successfully. This problem could not be reproduced in the Firefox browser, since all the exceptions regarding erroneous calls of the `clearTimeout`-function do not have any effect[3].

[1] jQuery homepage
[2] flot (Google Code)

7.5 Heuristic Evaluation

7.5.1 Purpose

Since the Spline Tool allows an easy way to visualize natural cubic splines, it is expected to be used by many people. Therefore, it is important that the Spline Tool can be used easily without requiring training. The purpose of the heuristic evaluation for the Spline Tool is to provide a better user experience. Possible flaws and problems in the user interface design can be identified with an appropriate heuristic evaluation. In this part, a list of evaluation heuristics[4] is presented and how each evaluation heuristic is applied on the graphical user interface of the Spline Tool.

7.5.2 Usability Heuristics

Visibility of system status

> The system should always keep users informed about what is going on, through appropriate feedback within reasonable time.

The Spline Tool shows a notification that informs the user about the state of the calculation. Since the only thing the Spline Tool does is calculating a natural cubic spline and plotting the graph of the latter, no other feedback is needed.

Match between system and the real world

> The system should speak the users' language, with words, phrases and concepts familiar to the user, rather than system-oriented terms.

[3] `window.clearTimeout` reference on the Mozilla Developer Center
[4] 'Ten Usability Heuristics' by Jacob Nielsen

Browser	Version	Compatibility	Notes
Firefox	3.0.10	Full	
Opera	9.64	Full	
Chrome	2.0	Partially	This issue is caused by the execution of the Spline Tool from the local file system. Toggle functionality using local cookies is not supported. Chrome does not allow to set local cookies
I. Explorer	7.0	Partially	This issue is caused by the execution of the Spline Tool from the local filesystem. Without explicitly allowing the Spline Tool to execute within the Internet Explorer, its functionality cannot be used (Internet Explorer security restriction). With the Spline Tool allowed to execute within the Internet Explorer and the latter accepting *all* cookies, the toggle functionality using local cookies is still not supported
Konqueror	4.2.2	Partially	This issue is caused by the execution of the Spline Tool from the local filesystem. The toggle functionality using local cookies is not supported, even if the *Allow all cookies*-setting is selected.

> Follow real-world conventions, making information appear in a natural and logical order.

The Spline Tool uses words, phrases and concepts familiar to the user. Information appears in a natural and logical order.

User control and freedom

> Users often choose system functions by mistake and will need a clearly marked "emergency exit" to leave the unwanted state without having to go through an extended dialogue. Support undo and redo.

The graphical user interface of the Spline Tool allows the user to reset the entered values by hitting the *Reset Values*-button. The browser, in which the Spline Tool is executed, allows to undo inputs made via the input text field.

Consistency and standards

> Users should not have to wonder whether different words, situations, or actions mean the same thing. Follow platform conventions.

The Spline Tool is held very simple and only has one form. The form component consists only of an input text field and two labeled buttons, each triggering the described functionality. Thus, the user can handle the Spline Tool without difficulty and the offered graphical user interface does not lead to confusion on the side of the user.

Error prevention

> Even better than good error messages is a careful design which prevents a problem from occurring in the first place. Either eliminate error-prone conditions or check for them and present users with a confirmation option

> before they commit to the action.

A description regarding the usage of the Spline Tool is provided and is the first paragraph shown. In the usage description it is also specified what input is supported by the Spline Tool. If the user enters invalid input an informative error message shows up that informs the user about the error which occurred and which steps are needed to correct it.

Recognition rather than recall

> Minimize the user's memory load by making objects, actions, and options visible. The user should not have to remember information from one part of the dialogue to another. Instructions for use of the system should be visible or easily retrievable whenever appropriate.

Since the Spline Tool provides its functionality using only a simple form, the user does not have to remember information. Regarding the instruction for the usage of the Spline Tool, in the section *Usage* at the top of the tool a short usage description is placed. The usage description describes how the user can define a correct input and offers in the section *Examples* some predefined examples to be executed on the fly. With the aid of the latter the user can easily see what the Spline Tool does and especially what is needed to get the Spline Tool started.

Flexibility and efficiency of use

> Accelerators – unseen by the novice user – may often speed up the interaction for the expert user such that the system can cater to both inexperienced and experienced users. Allow users to tailor frequent actions.

There is no need of tailoring frequent actions due to the fact that the Spline Tool is held very simple in its functionality.

Aesthetic and minimalist design

> Dialogues should not contain information which is irrelevant or rarely needed. Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility.

The space on the top of the Spline Tool is used for a short description of the tool and a simple usage description with some examples. After the user has used the Spline Tool for a few times, these descriptions are not required anymore. The graphical user interface of the Spline Tool provides the functionality to hide the descriptions permanently by setting a user-specific cookie[5] and checking the cookie every time the user starts the Spline Tool. The user has also the possibility to re-enable the deactivated descriptions. The provided form per se is held very simple, since it consists only of an input text field and two buttons.

Help users recognize, diagnose, and recover from errors

> Error messages should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution.

The error messages shown by the Spline Tool do not contain error codes. As mentioned before in section 7.5.2 on page 82, if the user enters invalid input an informative error message shows up that informs the user about the error that occurred and which steps are needed to correct the error.

Help and documentation

> Even though it is better if the system can be used without

[5] Please refer to section 7.4 for a compatibility overview.

documentation, it may be necessary to provide help and documentation. Any such information should be easy to search, focused on the user's task, list concrete steps to be carried out, and not be too large.

For information regarding this point please refer to section 7.5.2 on page 82

Chapter 8

Conclusion

The discussed algorithms and design allow the realization of an efficient, expandable and easy adaptable move engine. Because of the engine's generic nature, new devices can be supported without requiring an update in the control logic. Also, all existing devices benefit from improvements in the engine.

The developed algorithms take into account unique properties of the target environment and thus deliver better performance than plain algorithms.

By using intelligent movement planning with automatic collision resolution, the move engine enables the calling component to perform parallel and optimized moves without having to specify detailed device parameter and properties or to implement per-device control logic. Additionally, computing truly smooth routes allows higher device speeds and increases throughput.

The logical system boundaries and control flows from already developed components could be preserved, allowing an easy integration in existing systems.

Part III

Testing

Chapter 9

Test Plan

9.1 Algorithm Tests

Due to the many algorithms used it is very complex to set up environments which cause some desired behavior. Because of this, most of the listed tests are realized as automated tests within the NUnit[1] framework.

9.1.1 Logical Map Traversal

The traversal algorithm (Dijkstra's algorithm) gets a list of nodes and is requested to find a path through that list of nodes. Depending on the set of nodes given, different behavior is expected. The following color scheme is used in the graph throughout this chapter:

- **Green border**: Root node
- **Red border**: Goal node
- **Black border**: Normal node
- **Blue border, blue edges**: Shortest path
- **Gray number**: Edge cost

LMT-T01: No Nodes

The traversal algorithm must detect invalid input and abort the execution.

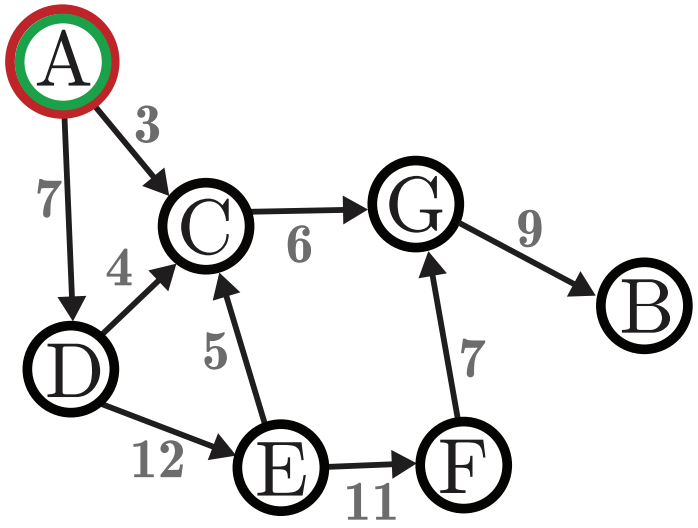
LMT-T02: One Waypoint, Goal Node Does Not Exist

If the one waypoint available does not equal to the defined goal node, the traversal algorithm must return that no path could be found.

LMT-T03: One Node, Root Node Is Goal Node

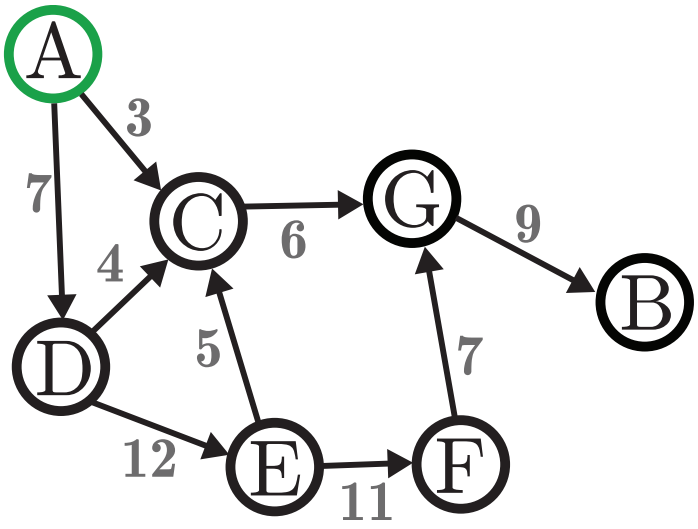
If the one waypoint available equals to the goal node, the traversal algorithm must return that no additional movements are required. Goal node A's distance must amount to 0.

[1] www.nunit.org



LMT-T04: Multiple Nodes, Goal Node Does Not Exist

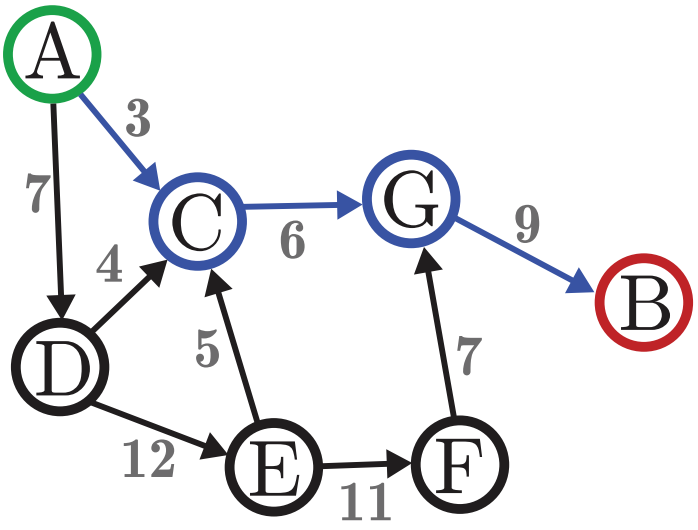
Since the goal node is not part of the graph, the algorithm must return that no path could be found.



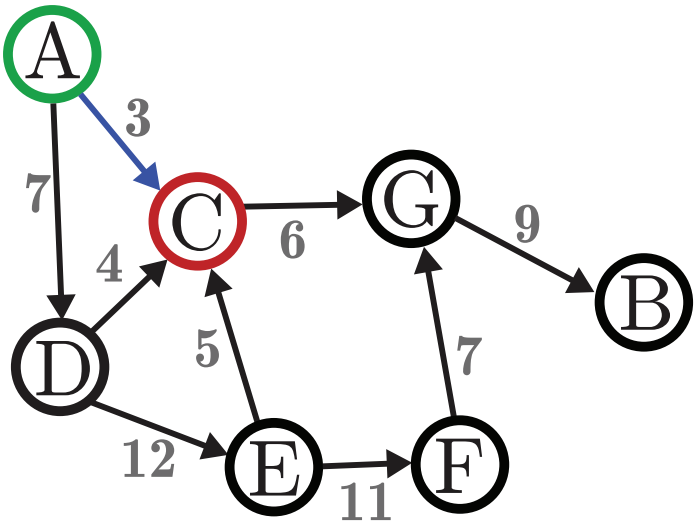
LMT-T05: Multiple Nodes, Goal Node Does Exist

Since the goal node is part of the graph, the algorithm must return the shortest path to the specified goal node. Every node must be set as the goal node, to test if the algorithm finds a shortest path to all the possible goal nodes (excluding to set the root node as goal node, since this test case is described in section 9.1.1). The possible variants of this scenario are presented in the following test cases.

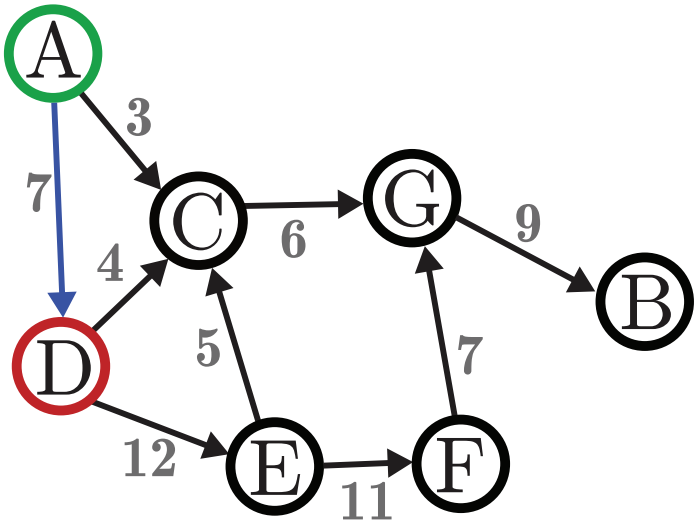
LMT-T05-01: Case: Goal Node = B The shortest path to the goal node B is: $A \rightarrow C \rightarrow G \rightarrow B$. B's distance from the root node must amount to 18 after completion.



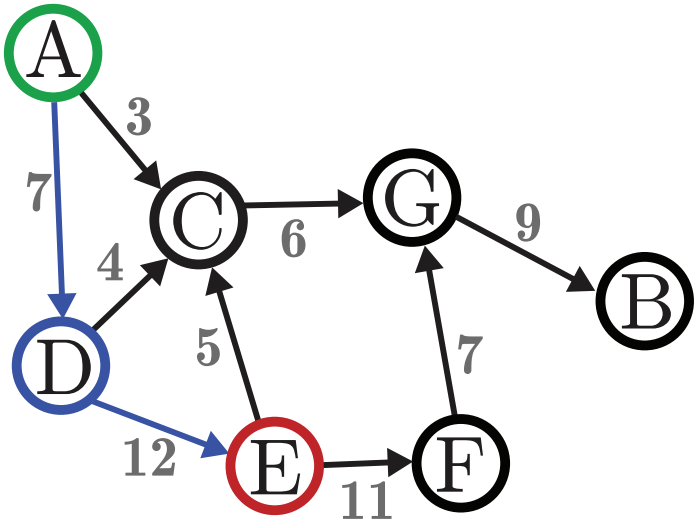
LMT-T05-02: Case: Goal Node = C The shortest path to the goal node C is: $A \rightarrow C$. C's distance from the root node must amount to 3 after completion.



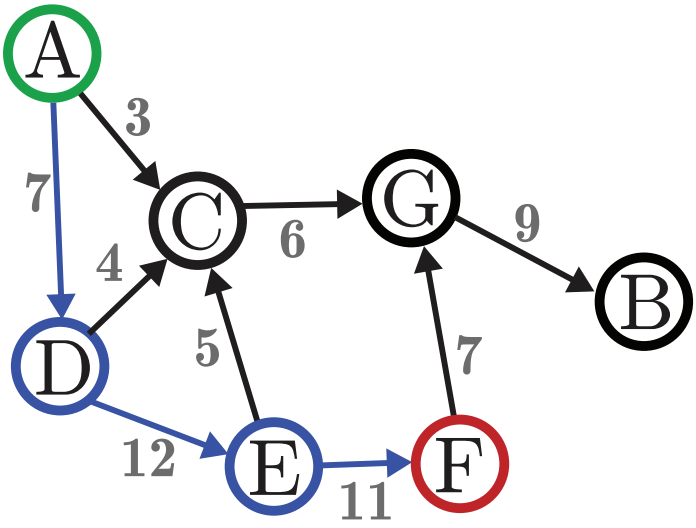
LMT-T05-03: Case: Goal Node = D The shortest path to the goal node D is: $A \rightarrow D$. D's distance from the root node must amount to 7 after completion.



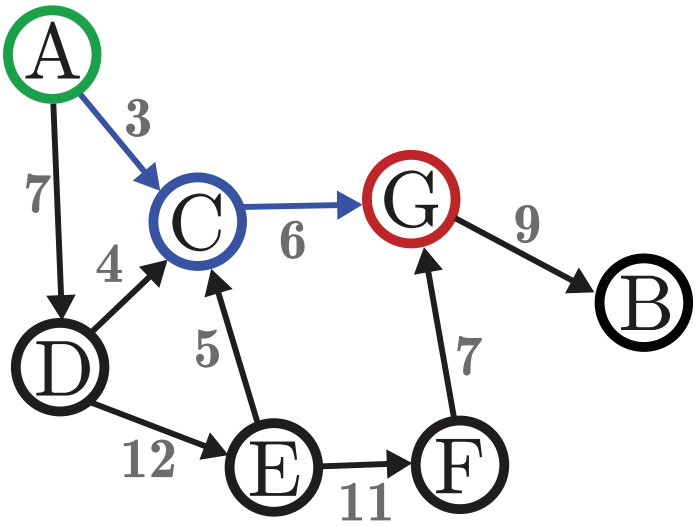
LMT-T05-04: Case: Goal Node = E The shortest path to the goal node E is: $A \rightarrow D \rightarrow E$. E's distance from the root node must amount to 19 after completion.



LMT-T05-05: Case: Goal Node = F The shortest path to the goal node F is: $A \rightarrow D \rightarrow E \rightarrow F$. F's distance from the root node must amount to 30 after completion.

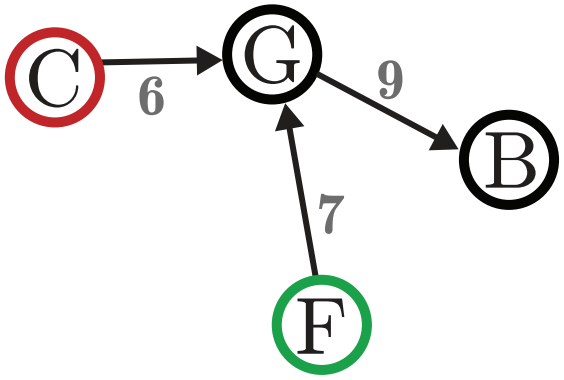


LMT-T05-06: Case: Goal Node = G The shortest path to the goal node G is: $A \rightarrow C \rightarrow G$. G's distance from the root node must amount to 9 after completion.



LMT-T06: Multiple Nodes, Goal Node Does Exist But Is Unreachable

The specified goal node is part of the graph, but cannot be reached from the starting node F. The algorithm must return that no path could be found since F is not able to reach C.

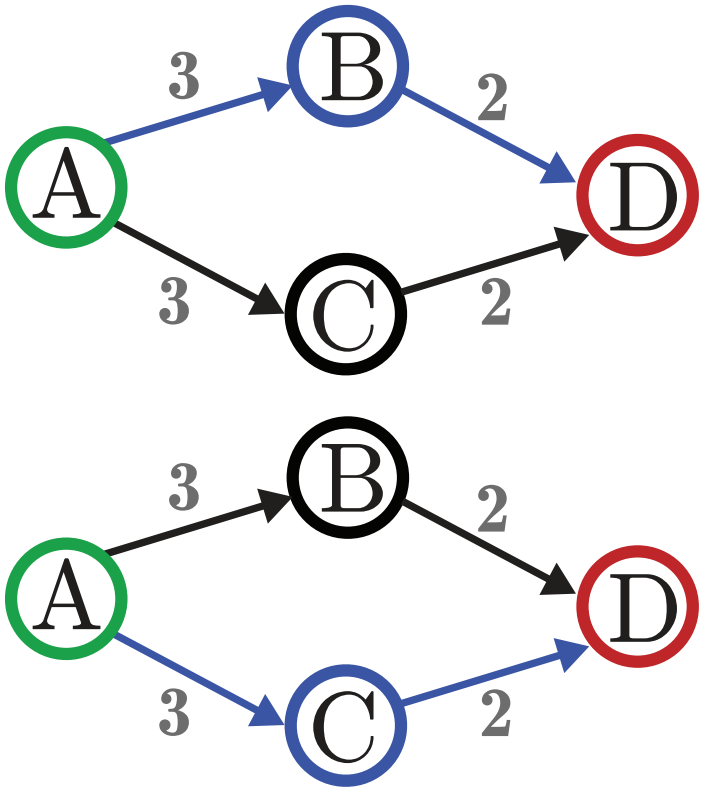


LMT-T07: Multiple Nodes, Goal Node Exists, Two Equal Shortest Paths Exist

Two equal weighted shortest paths lead to the goal node D from the starting node A. The shortest paths for this graph are the following:

- A → B → D
- A → C → D

It does not matter which path is returned as the shortest path by the algorithm, since only one shortest path must be selected. D's distance from the root node must amount to 5 after completion.

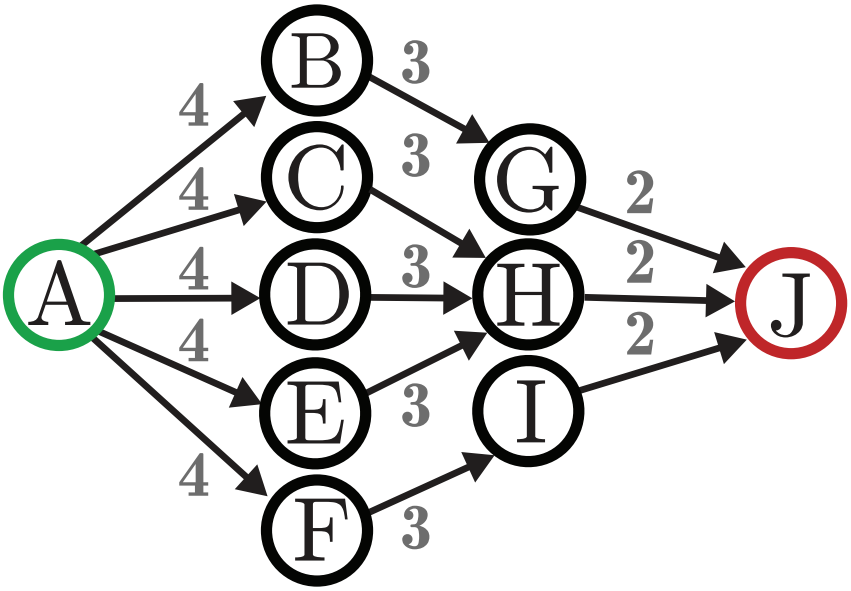


LMT-T08: Multiple Nodes, Goal Node Exists, Multiple Equal Shortest Paths Exist

Multiple equal weighted paths lead to the goal node J from the starting node A. The shortest paths for this graph are the following:

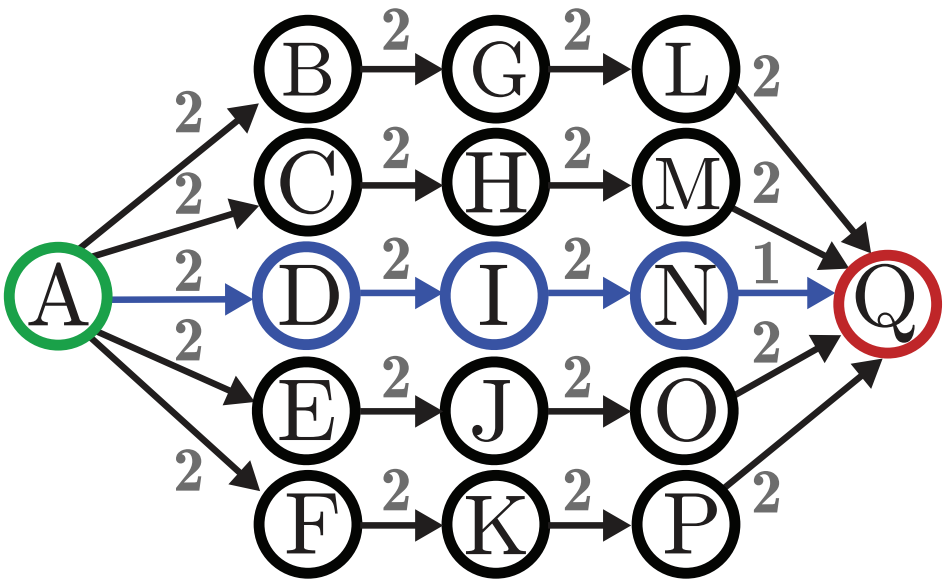
- A → B → G → J
- A → C → H → J
- A → D → H → J
- A → E → H → J
- A → F → I → J

It does not matter which path is returned as the shortest path by the algorithm, since only one shortest path must be selected. J's distance from the root node must amount to 9 after completion.

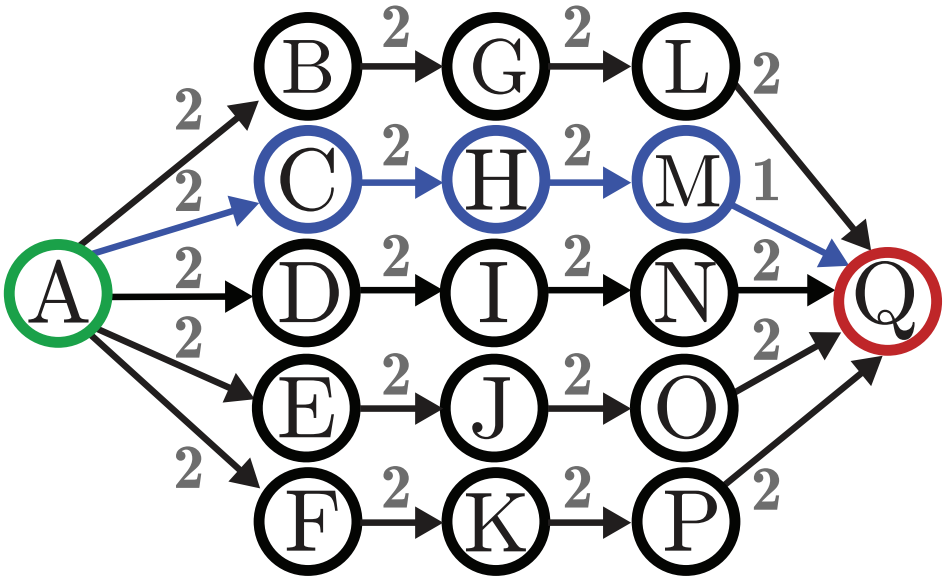


LMT-T09: Multiple Nodes, Goal Node Exists, Multiple Equal Paths Exist, Only One Shortest Path Exists

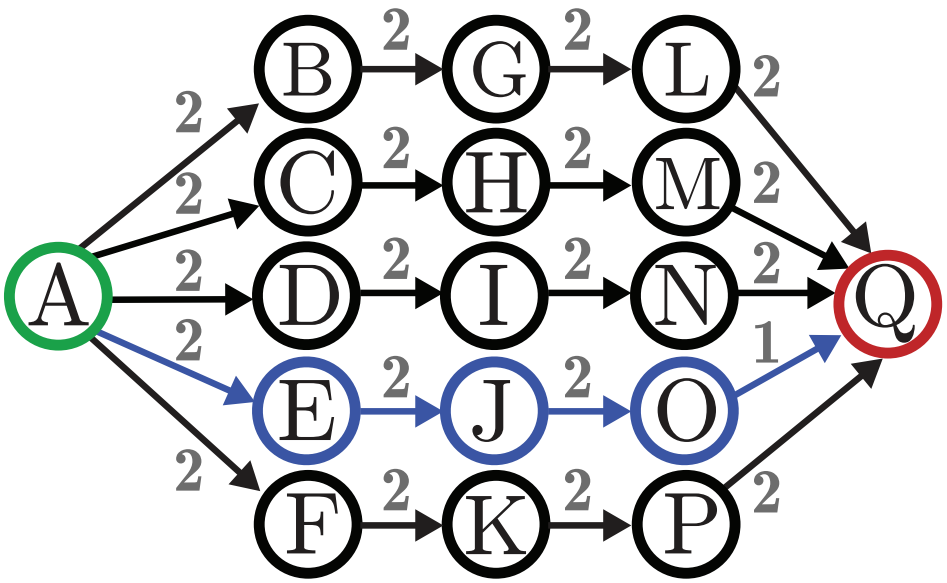
Test Path 1 Multiple equal weighted paths lead to the goal node Q from the starting node A, but in the following scenario only one shortest path exists, which is A → D → I → N → Q. Q's distance from the root node must amount to 7 after completion.



Test Path 2 Multiple equal weighted paths lead to the goal node Q from the starting node A, but in the following scenario only one shortest path exists, which is A → C → H → M → Q. Q's distance from the root node must amount to 7 after completion.

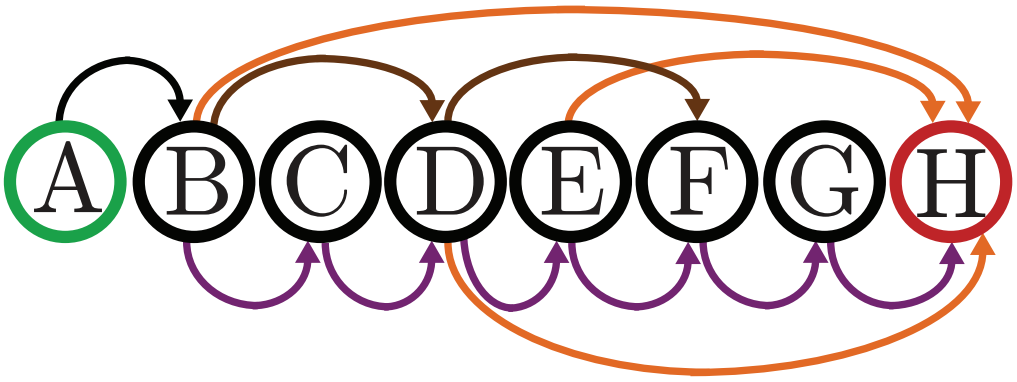


Test Path 3 Multiple equal weighted paths lead to the goal node Q from the starting node A, but in the following scenario only one shortest path exists, which is A → E → J → O → Q. Q's distance from the root node must amount to 7 after completion.



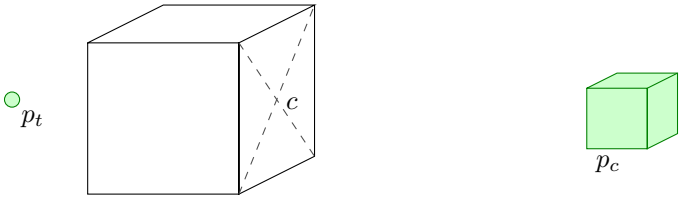
LMT-P-T10: Multiple Nodes, Goal Node Exists, Nodes Are Connected Randomly With Goal Node

Nodes varying from a few to a hundred thousand are connected randomly together. The edge costs are missing since they are generated randomly. The root node is connected to the first node in a list of nodes (black connection). First the brown connections are set up by using a fixed step length (in the graph below the step length is two). Second the purple connections are set up with a random step length and last the orange connections are set up, which point directly to the root node and are set up randomly, too. The purpose of this test case is to test the performance of our implementation of the Dijkstra's algorithm.



9.1.2 Collision Resolution

In all collision resolution there is following given scenario



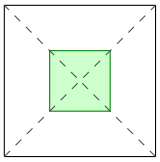
The green object wants to move from p_c around the big obstacle to its target position, the green circle p_t. The tests verify the evading mechanism for different initial positions which might occur. Depending on how the green object is positioned to the obstacle, it must choose a different way to realize the shortest path to its target position.

Note: In this document *Center* refers to the intersection point of the dashed lines marked with c.

Centered

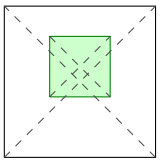
Both balance points stand parallel to each other. In this scenario the device must move north, east, south or west.

Only the essential test cases are implemented as the correctness of the other test cases can be derived from the results of the former.



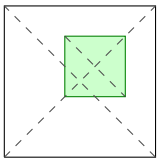
CR-T01: Above Center

The device's balance point is above the center and its boundaries are within the one of the obstacle. The device must move north to avoid the obstacle.



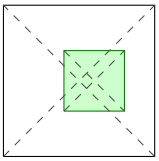
CR-T02: North-West of Center

The device's balance point is equally far away from the north and the west border and still in the obstacle's boundaries. The device could move north or west to avoid the obstacle.



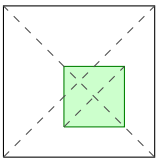
CR-T03: Right Of Center

The device's balance point is right of the center and its boundaries are within the one of the obstacle. The device must move west to avoid the obstacle.



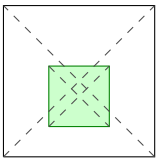
CR-T04: South-East of Center

The device's balance point is equally far away from the south and the east border and still in the obstacle's boundaries. The device could move south or east to avoid the obstacle.



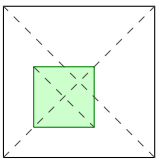
CR-T05: Below Center

The device's balance point is below center and its boundaries are within the one of the obstacle. The device must move south to avoid the obstacle.



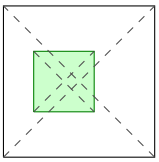
CR-T05: South-West of Center

The device's balance point is equally far away from the south and the west border and still in the obstacle's boundaries. The device could move south or west to avoid the obstacle.



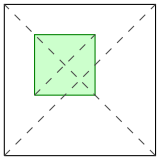
CR-T06: Left Of Center

The device's balance point is left of the center and its boundaries are within the one of the obstacle. The device must move east to avoid the obstacle.



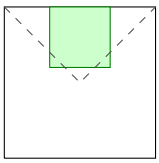
CR-T07: North-East of Center

The device's balance point is equally far away from the north and the east border and still in the obstacle's boundaries. The device could move north and east avoid the obstacle.



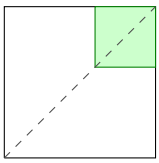
CR-T08: Inside North

The device's balance point is north of the obstacle one's. Therefore, the device must move north.



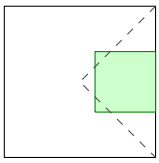
CR-T09: Inside North-East

The device's balance point is equally far away from the north and east border. Therefore, the device could move north or east to pass the obstacle.



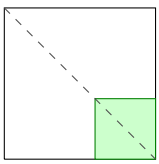
CR-T10: Inside East

The device's balance point is right the center. Therefore, the device must move east to pass the obstacle.



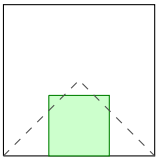
CR-T11: Inside South-East

The device's balance point is equally far away from the south and east border. Therefore, the device could move north or east to pass the obstacle.



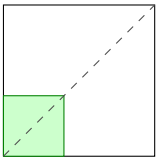
CR-T12: Inside South

The device's balance point is below the center. Therefore, the device must move south to pass the obstacle.



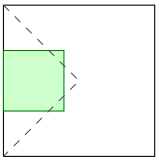
CR-T13: Inside South-West

The device's balance point is equally far away from the south and the west border. Therefore, the device could move south or west to pass the obstacle.



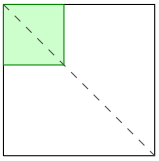
CR-T14: Inside West

The device's balance point is left the center. Therefore, the device must move west to pass the obstacle.



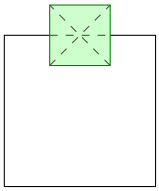
CR-T15: Inside North-West

The device's balance point is equally far away from the north and the west border. Therefore, the device could move north or west to pass the obstacle.



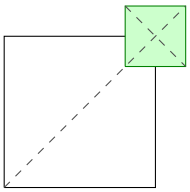
CR-T16: Intersection North

The device's balance point is above the center. Therefore, the device must move north to pass the obstacle.



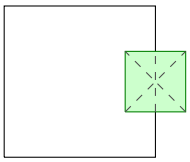
CR-T17: Intersection North-East

The device's balance point is equally far away from the north and the east border. Therefore, the device could move north or east to pass the obstacle.



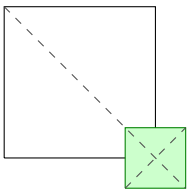
CR-T18: Intersection East

The device's balance point is right of the center. Therefore, the device must move east to pass the obstacle.



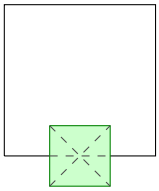
CR-T19: Intersection South-East

The device's balance point is equally far away from the south and the east border. Therefore, the device could move south or east to pass the obstacle.



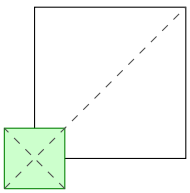
CR-T20: Intersection South

The device's balance point is below the center. Therefore, the device must move south to pass the obstacle.



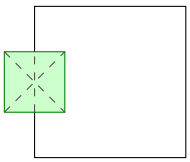
CR-T21: Intersection South-West

The device's balance point is equally far away from the south and the west border. Therefore, the device could move south and east to pass the obstacle.



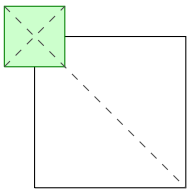
CR-T22: Intersection West

The device's balance point is right of the center. Therefore, the device must move east to pass the obstacle.



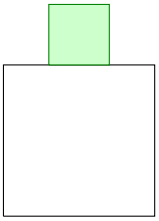
CR-T23: Intersection North-West

The device's balance point is equally far away from the north and the west border. Therefore, the device could move north or west to pass the obstacle.



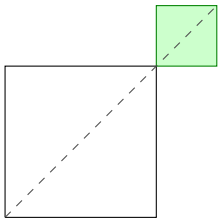
CR-T24: Bordering North

The device's balance point is above the center. Therefore, the device must move north to pass the obstacle.



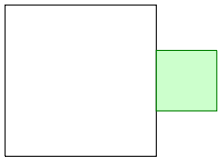
CR-T25: Bordering North-East

The device's balance point is equally far away from the north and the east border. Therefore, the device could move north or east to pass the obstacle.



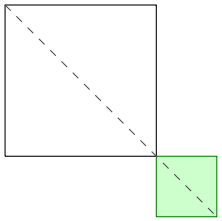
CR-T26: Bordering East

The device's balance point is right of the center. Therefore, the device must move right to pass the obstacle.



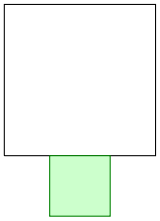
CR-T27: Bordering South-East

The device's balance point is equally far away from the south and the east border. Therefore, the device could move south or east to pass the obstacle.



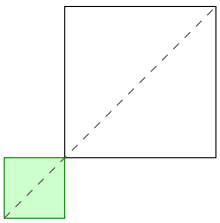
CR-T28: Bordering South

The device's balance point is below the center. Therefore, the device must move south to pass the obstacle.



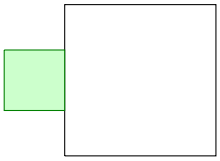
CR-T29: Bordering South-West

The device's balance point is equally far away from the south and the west border. Therefore, the device could move south or west to pass the obstacle.



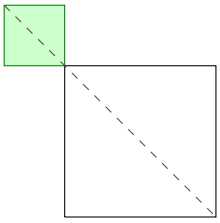
CR-T30: Bordering West

The device's balance point is left of the center. Therefore, the device must move west to pass the obstacle.



CR-T31: Bordering North-West

The device's balance point is equally far away from the north and the west border. Therefore, the device could move north or west to pass the obstacle.



9.1.3 Cubic Spline Interpolation

The algorithm for calculating natural cubic splines is tested by setting fixed coordinates and comparing the resulting parameters with precompiled results[2]. The algorithm is tested with integer coordinates (e.g. (1,2), (3,4) and (5,6)) and with floating point coordinates (e.g. (1.1, 3.412), (2.324, 8.132) and (4.123, 10.132)). Since the Spline Tool is implemented using HTML and JavaScript, no automated tests using the NUnit framework are possible. The test shall be performed manually.

CSI-T01: No Coordinates

If no coordinates are committed, the algorithm must return that it is not possible to calculate a natural cubic spline.

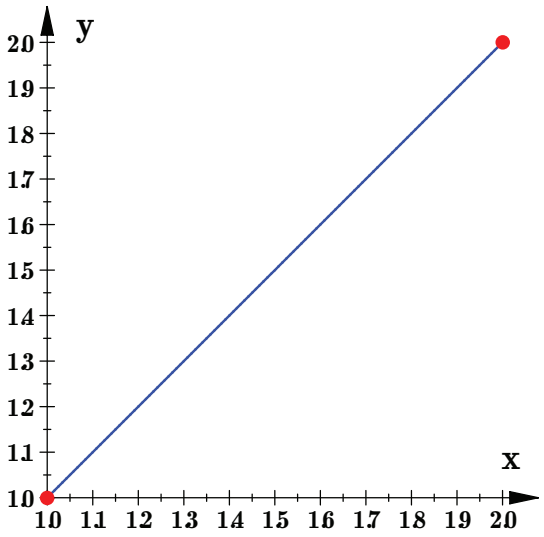
[2]Cubic spline java applet and calculator

CSI-T02: Too Few Coordinates

If less than two coordinates are committed, the algorithm must return that it is not possible to calculate a natural cubic spline.

CSI-T03: Natural Cubic Spline With Integer Coordinates - Minimum Amount Of Coordinates

A natural cubic spline is constructed through the coordinates (1, 1) and (2, 2). The resulting cubic polynomial is $S0(x) = 1 + (x - 1)$ for x $\in [1; 2]$. The plot of the resulting spline is the following (drawn with MuPAD[3]):



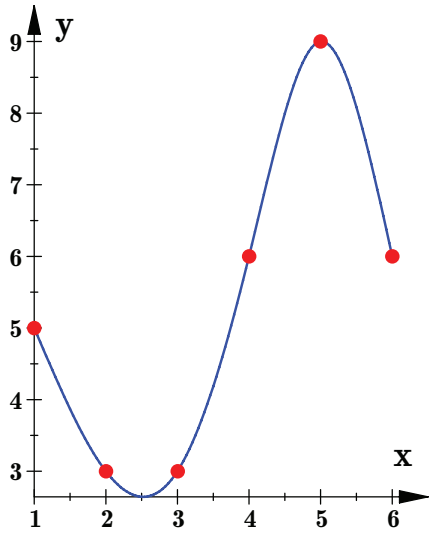
CSI-T04: Natural Cubic Spline With Integer Coordinates - Normal

A natural cubic spline is constructed through the coordinates (1, 5), (2, 3), (3, 3), (4, 6), (5, 9) and (6, 6). The resulting cubic polynomials are:

- $S0(x) = 0.349(x - 1)^3 - 2.349(x - 1) + 5$ for x $\in [1; 2]$
- $S1(x) = 0.254(x - 2)^3 + 1.048(x - 2)^2 - 1.301(x - 2) + 3$ for x $\in [2; 3]$
- $S2(x) = -0.364(x - 3)^3 + 1.809(x - 3)^2 + 1.555(x - 3) + 3$ for x $\in [3; 4]$
- $S3(x) = -1.799(x - 4)^3 + 0.718(x - 4)^2 + 4.081(x - 4) + 6$ for x $\in [4; 5]$
- $S4(x) = 1.56(x - 5)^3 - 4.679(x - 5)^2 + 0.12(x - 5) + 9$ for x $\in [5; 6]$

The plot of the resulting spline is the following (drawn with MuPAD):

[3]MuPAD: Computer algebra system

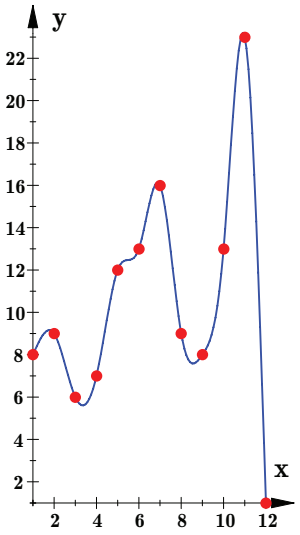


CSI-T05: Natural Cubic Spline With Integer Coordinates - Extensive

A natural cubic spline is constructed through the coordinates (1, 8), (2, 9), (3, 6), (4, 7), (5,12), (6, 13), (7, 16), (8, 9), (9, 8), (10, 13), (11, 23) and (12, 1). The resulting cubic polynomials are:

- $S0(x) = -1.254(x-1)^3 + 2.254(x-1) + 8$ for x $\in [1;2]$
- $S1(x) = 2.272(x-2)^3 - 3.763(x-2)^2 - 1.509(x-2) + 9$ for x $\in [2;3]$
- $S2(x) = 0.166(x-3)^3 + 3.053(x-3)^2 - 2.219(x-3) + 6$ for x $\in [3;4]$
- $S3(x) = -2.936(x-4)^3 + 3.551(x-4)^2 + 4.385(x-4) + 7$ for x $\in [4;5]$
- $S4(x) = 3.578(x-5)^3 - 5.257(x-5)^2 + 2.679(x-5) + 12$ for x $\in [5;6]$
- $S5(x) = -5.377(x-6)^3 + 5.477(x-6)^2 + 2.899(x-6) + 13$ for x $\in [6;7]$
- $S6(x) = 5.928(x-7)^3 - 10.652(x-7)^2 - 2.276(x-7) + 16$ for x $\in [7;8]$
- $S7(x) = -2.335(x-8)^3 + 7.132(x-8)^2 - 5.796(x-8) + 9$ for x $\in [8;9]$
- $S8(x) = 3.414(x-9)^3 + 0.125(x-9)^2 + 1.461(x-9) + 8$ for x $\in [9;10]$
- $S9(x) = -12.319(x-10)^3 + 10.367(x-10)^2 + 11.953(x-10) + 13$ for x $\in [10;11]$
- $S10(x) = 8.864(x-11)^3 - 26.592(x-11)^2 - 4.272(x-11) + 23$ for x $\in [11;12]$

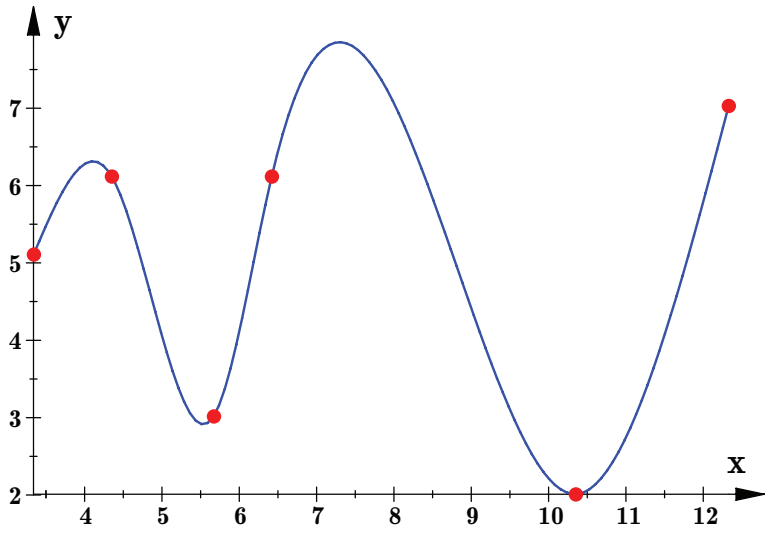
The plot of the resulting spline is the following (drawn with MuPAD):



CSI-T06: Natural Cubic Spline With Floating Point Coordinates

A natural cubic spline is constructed through the floating point coordinates (3.34, 5.11), (4.35, 6.12), (5.67, 3.02), (6.42, 6.12), (10.35, 2.01) and (12.33, 7.03). The resulting cubic polynomials are:

- $S0(x) = -1.32(x - 3.34)^3 + 2.347(x - 3.34) + 5.11$ for x $\in [3.34; 4.35]$
- $S1(x) = 2.655(x - 4.35)^3 - 4(x - 4.35)^2 - 1.694(x - 4.35) + 6.12$ for x $\in [4.35; 5.67]$
- $S2(x) = -4.218(x-5.67)^3+6.512(x-5.67)^2+1.622(x-5.67)+3.02$ for x $\in [5.67; 6.42]$
- $S3(x) = 0.414(x-6.42)^3-2.979(x-6.42)^2+4.272(x-6.42)+6.12$ for x $\in [6.42; 10.35]$
- $S4(x) = -0.32(x-10.35)^3+1.9(x-10.35)^2+0.028(x-10.35)+2.01$ for x $\in [10.35; 12.33]$



CSI-T07: Natural Cubic Spline With Unsorted Floating Coordinates

A natural cubic spline is constructed through floating point coordinates. The coordinates used for this test are the same as the ones used in the *Natural Cubic Spline With Floating Coordinates* test and are passed in an unsorted order. Since the coordinates are the same, but differs in the order they are delivered, the calculated parameters for the natural cubic spline must be the same.

CSI-P-T08: Natural Cubic Spline With Random Floating Point Coordinates

A natural cubic spline is constructed through a great number of *randomly generated* floating point coordinates. The goal of this test is to check the performance of the natural cubic spline algorithm.

CSI-P-T09: Natural Cubic Spline With Random Integer Coordinates

A natural cubic spline is constructed through a great number of *randomly generated* integer coordinates. The goal of this test is to check the performance of the natural cubic spline algorithm.

9.2 Unit Conversion

9.2.1 Metric Tests

The units *km,m,dm,cm,mm* and µm are supported. In this test following conversions are tested.

UC-T01: Basic Conversions

Following conversion are tested:
1km = 1000m
1m = 10dm
1dm = 10cm
1cm = 10mm
1m = 1000µm

UC-T02: km Conversions

Following conversion are tested:
2km = 2000m
0.5km = 500m
10km = 100000dm
0.3km = 30000cm
1.4km = 1400000mm
0.1km = 100000000µm

UC-T03: m Conversions

Following conversion are tested:
2m = 0.002km
10m = 100dm

0.3m = 30cm
1.4m = 1400mm
0.1m = 100000µm

UC-T04: dm **Conversions**

Following conversion are tested:
2dm = 0.0002km
10dm = 1m
0.3dm = 3cm
1.4dm = 140mm
0.1dm = 10000µm

UC-T05: cm **Conversions**

2cm = 0.00002km
10cm = 0.1m
0.3cm = 0.03dm
1.4cm = 14mm
0.1cm = 1000µm

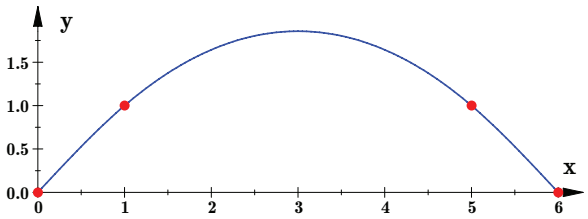
9.3 Spline Tool

9.3.1 Browser Compatibility

The purpose of these tests is to approve the compatibility of the JavaScript implementation of the Spline Tool with each browser in test.

BC-T01: Sample Spline - Integer Coordinates

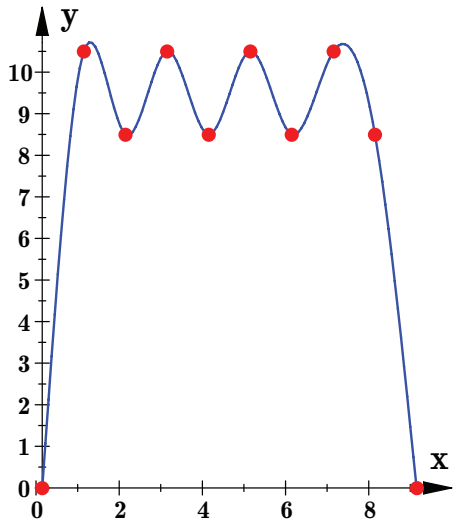
A simple input consisting of four integer coordinates is made by clicking on the *Sample Spline #1* link in the *Examples* part. As a result, the input text field must have set the value `0,0 1,1 5,1 6,0`, the counter on the right side of the input text field must display in bold *4* and finally, the browser must display the graph and the three cubic polynomials $S_0(x)$ to $S_2(x)$. The following is the resulting graph of the natural cubic spline (drawn with MuPAD):



BC-T02: Sample Spline - Floating Point Coordinates

A simple input consisting of ten floating point coordinates is made by clicking on the *Sample Spline #4* link in the *Examples* part. As a result, the input text field must have set the value `0.15,0 1.15,10.5 2.15,8.5 3.15,10.5 4.15,8.5 5.15,10.5 6.15,8.5 7.15,10.5 8.15,8.5`

`9.15,0`, the counter on the right side of the input text field must display in bold *10* and finally it must display the graph and the cubic polynomials $S_0(x)$ to $S_8(x)$. The following is the resulting graph of the natural cubic spline (drawn with MuPAD):



BC-T03: Keylistener

It is possible to hit the `ENTER`-key after entering the coordinates. The Spline Tool will then display according to the input either a result or an error message. This only works if the `ENTER`-key is hit while the focus is on the input text field.

The test consists of entering the coordinates `1,2 3,4` into the input text field and thereafter hitting the `ENTER`-key. The Spline Tool must display in the counter the value *2* in bold and finally it must display the graph and the cubic polynomial $S_0(x)$.

BC-T03: Local Cookie Support

The local cookie support is needed for the ability to hide information that the user does not need anymore. This ability was implemented while conducting an heuristic evaluation. Local cookies are used when the user starts the Spline Tool from the local file system, e.g. `file://C:/SplineTool/index.html`.

The test consists of first hiding each part on the top, namely the *Description*-, the *Usage*- and the *Examples*-part by clicking the yellow toggle-button next to each title. After hiding each part, the site is refreshed and as a positive result the previously hidden parts must not be displayed. If the procedure succeeded up to here, the same procedure must be run with first opening each part and then refreshing the site. After the refresh the previously opened parts must be displayed.

BC-T04: No Coordinates

No coordinates are entered. The Spline Tool must detect invalid input and display an appropriate error message.

BC-T05: Too Few Coordinates

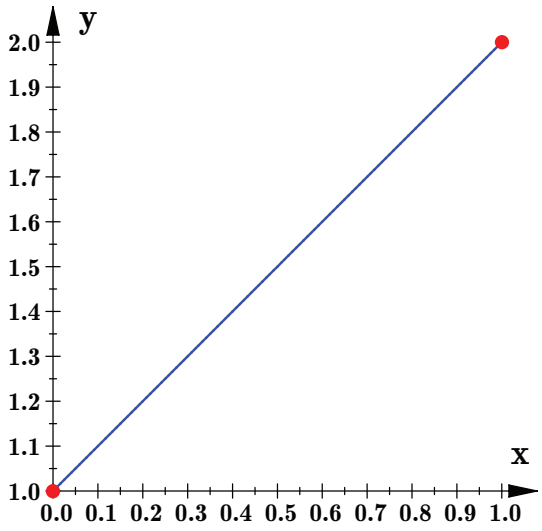
Only one coordinate is entered. Since it is required to enter a minimum of two coordinates, the Spline Tool must detect invalid input and display an appropriate error message.

BC-T06: Multiple Coordiantes With Same X Value

The value `0,1 1,2 3,4 3,5` are entered. Since it is not allowed to enter multiple coordinates with the same x value, the Spline Tool must detect invalid input and display an appropriate error message.

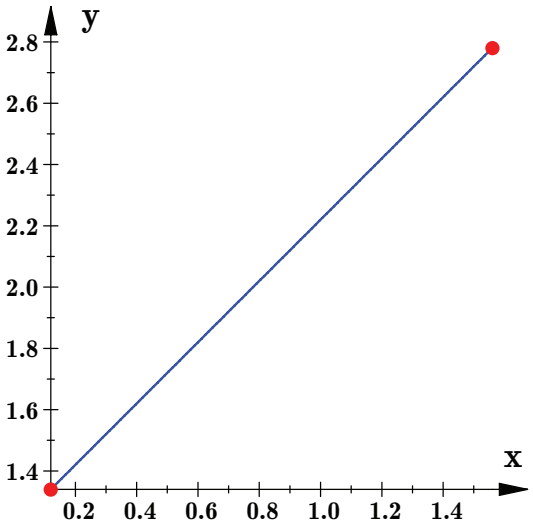
BC-T07: Integer Coordinates

The value `0,1 1,2` are entered manually. The Spline Tool must display the counter with the value *2* in bold, plot the graph of the natural cubic spline and print the cubic polynomial $S_0(x)$. The following is the resulting graph of the natural cubic spline calculated (drawn with MuPAD):



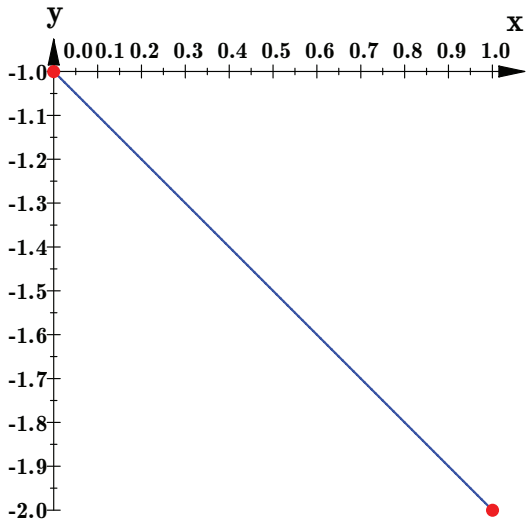
BC-T08: Floating Point Coordinates

The value `0.12,1.34 1.56,2.78` are entered manually. The Spline Tool must display the counter with the value *2* in bold, plot the graph of the natural cubic spline and print the cubic polynomial $S_0(x)$. The following is the resulting graph of the natural cubic spline calculated (drawn with MuPAD):



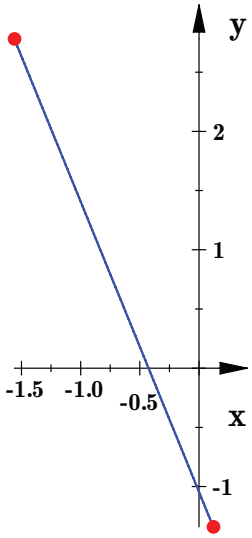
BC-T09: Negative Integer Coordinates

The value `0,-1 1,-2` are entered. The Spline Tool must display the counter with the value *2* in bold, plot the graph of the natural cubic spline and print the cubic polynomial $S_0(x)$. The following is the resulting graph of the natural cubic spline calculated (drawn with MuPAD):



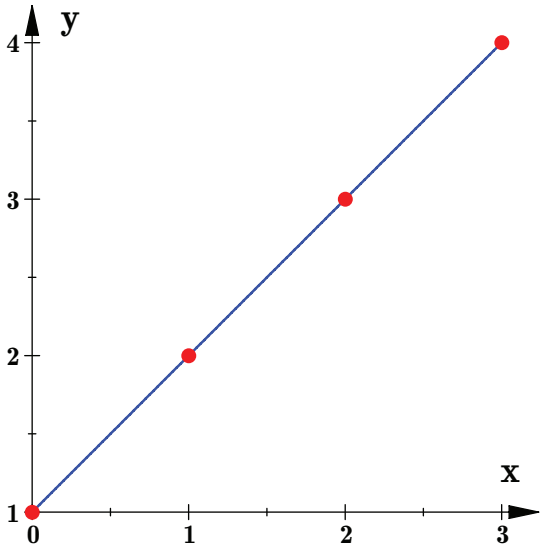
BC-T10: Negative Floating Point Coordinates

The value `-1.56,2.78 0.12,-1.34` are entered. The Spline Tool must display the counter with the value *2* in bold, plot the graph of the natural cubic spline and print the cubic polynomial $S_0(x)$. The following is the resulting graph of the natural cubic spline calculated (drawn with MuPAD):



BC-T11: Unsorted Input

The value `2,3 3,4 1,2 0,1` are entered. The Spline Tool must display the same results as if the value entered was `0,1 1,2 2,3 3,4`. The Spline Tool sorts the entered value consisting of unordered coordinates by the X values in an ascending order. The following is the resulting graph of the natural cubic spline calculated (drawn with MuPAD):



BC-T12: Reset Inputs

With a click on the *Reset Values*-button the Spline Tool must reset the input text value, the counter, the graph of the natural cubic spline and the cubic polynomials.

BC-T13: Example Inputs

Each example input in the *Examples* part is clicked and run. The Spline Tool must fill the input text field with the coordinates, must display the counter and finally it must plot the graph of the natural cubic spline and print the cubic polynomials.

9.4 Parser Tests

Data Validation

The parser must not accept input data which

- Is not set to an instance
- Is an empty string
- Just contains “move”
- Contains an invalid identifier
- Contains no move parameters

9.4.1 P-T01: Absolute Values

If absolute values are specified they must be written to the resulting MoveRequest class.

Input parameters	“x=1 y=10 z=100”
Expected X	1
Expected Y	10
Expected Z	100

9.4.2 P-T02: Positive Relative Values

Existing X	1
Existing Y	2
Existing Z	3
Input parameters	“x+=1 y+=1 z+=1”
Expected X	2
Expected Y	3
Expected Z	4

9.4.3 P-T03: Negative Relative Values

Existing X	1
Existing Y	2
Existing Z	3
Input parameters	“x-=1 y-=2 z-=3”
Expected X	0
Expected Y	0
Expected Z	0

Chapter 10

Test Report 12.05.2009

Date	12.05.2009
Tester	GiAc
Code Revision	r616

For detailed test descriptions refer to the chapter 9 on page 90.

Algorithm Tests

Logical Map Traversal Test Results

Test	Notes	Result
LMT-T01		Sucess
LMT-T02		Success
LMT-T03		Success
LMT-T05-01		Success
LMT-T05-02		Success
LMT-T05-03		Success
LMT-T05-04		Success
LMT-T05-05		Success
LMT-T05-06		Success
LMT-T06		Success
LMT-T07		Success
LMT-T08		Success
LMT-T09		Success
LMT-P-T10		Success

Collision Resolution Test Results

Test	Notes	Result
CR-T01		Success
CR-T02		Success
CR-T03		Success
CR-T04		Success
CR-T05		Success
CR-T06		Success
CR-T07		Success
CR-T08		Success
CR-T09		Success
CR-T10		Success
CR-T11		Success
CR-T12		Success
CR-T13		Success
CR-T14		Success
CR-T15		Success
CR-T16		Success
CR-T17		Success
CR-T18		Success
CR-T19		Success
CR-T20		Success
CR-T21		Success
CR-T22		Success
CR-T23		Success
CR-T24		Success
CR-T25		Success
CR-T26		Success
CR-T27		Success
CR-T28		Success
CR-T29		Success
CR-T30		Success
CR-T31		Success

Cubic Spline Interpolation Test Results

Test	Notes	Result
CSI-T-01		Success
CSI-T-02		Success
CSI-T-03		Success
CSI-T-04		Success
CSI-T-05		Success
CSI-T-06		Success
CSI-T-07		Success
CSI-P-T-08		Success
CSI-P-T-09		Success

Unit Conversion Tests

Test	Notes	Result
UC-T01		Success
UC-T02		Success
UC-T03		Success
UC-T04		Success
UC-T05		Success

Parser Tests

Test	Notes	Result
P-T01		Success
P-T02		Success
P-T03		Success

Chapter 11

Test Report 27.05.2009

Date	27.05.2009
Tester	GiAc
Code Revision	r643

For detailed test descriptions refer to the chapter 9 on page 90.

Algorithm Tests

Logical Map Traversal Test Results

Test	Notes	Result
LMT-T01		Sucess
LMT-T02		Success
LMT-T03		Success
LMT-T05-01		Success
LMT-T05-02		Success
LMT-T05-03		Success
LMT-T05-04		Success
LMT-T05-05		Success
LMT-T05-06		Success
LMT-T06		Success
LMT-T07		Success
LMT-T08		Success
LMT-T09		Success
LMT-P-T10		Success

Collision Resolution Test Results

Test	Notes	Result
CR-T01		Success
CR-T02		Success
CR-T03		Success
CR-T04		Success
CR-T05		Success
CR-T06		Success
CR-T07		Success
CR-T08		Success
CR-T09		Success
CR-T10		Success
CR-T11		Success
CR-T12		Success
CR-T13		Success
CR-T14		Success
CR-T15		Success
CR-T16		Success
CR-T17		Success
CR-T18		Success
CR-T19		Success
CR-T20		Success
CR-T21		Success
CR-T22		Success
CR-T23		Success
CR-T24		Success
CR-T25		Success
CR-T26		Success
CR-T27		Success
CR-T28		Success
CR-T29		Success
CR-T30		Success
CR-T31		Success

Cubic Spline Interpolation Test Results

Test	Notes	Result
CSI-T-01		Success
CSI-T-02		Success
CSI-T-03		Success
CSI-T-04		Success
CSI-T-05		Success
CSI-T-06		Success
CSI-T-07		Success
CSI-P-T-08		Success
CSI-P-T-09		Success

Unit Conversion Tests

Test	Notes	Result
UC-T01		Success
UC-T02		Success
UC-T03		Success
UC-T04		Success
UC-T05		Success

Parser Tests

Test	Notes	Result
P-T01		Success
P-T02		Success
P-T03		Success

Spline Tool Browser Compatibility Tests

Firefox (Version 3.0.10)

Test	Notes	Result
BC-T01		Success
BC-T02		Success
BC-T03		Success
BC-T04		Success
BC-T05		Success
BC-T06		Success
BC-T07		Success
BC-T08		Success
BC-T09		Success
BC-T10		Success
BC-T11		Success
BC-T12		Success
BC-T13		Success

Opera (Version 9.64)

Test	Notes	Result
BC-T01	Unknown error caused by a call of the `SetSampleCoordinates` function. It is possible to enter the coordinates and the counter works. It is not possible to use one of the links in the *Examples* part. Also, the plotting of the graph and the printing of the cubic polynomials does not work	Fail
BC-T02	Refer to BC-T01	Fail
BC-T03	Refer to BC-T01	Fail
BC-T04	Refer to BC-T01	Fail
BC-T05	Refer to BC-T01	Fail
BC-T06	Refer to BC-T01	Fail
BC-T07	Refer to BC-T01	Fail
BC-T08	Refer to BC-T01	Fail
BC-T09	Refer to BC-T01	Fail
BC-T10	Refer to BC-T01	Fail
BC-T11	Refer to BC-T01	Fail
BC-T12	Refer to BC-T01	Fail
BC-T13	Refer to BC-T01	Fail

Chrome (Version 2.0.172.28)

Test	Notes	Result
BC-T01		Success
BC-T02		Success
BC-T03	Local cookies are not allowed in Chrome[1]	Provider problem
BC-T04		Success
BC-T05		Success
BC-T06		Success
BC-T07		Success
BC-T08		Success
BC-T09		Success
BC-T10		Success
BC-T11		Success
BC-T12		Success
BC-T13		Success

[1] Issue 535: Support Cookies on file://

Chapter 12

Test Report 02.06.2009

Date	02.06.2009
Tester	GiAc
Code Revision	r675

For detailed test descriptions refer to the chapter 9 on page 90.

Algorithm Tests

Logical Map Traversal Test Results

Test	Notes	Result
LMT-T01		Sucess
LMT-T02		Success
LMT-T03		Success
LMT-T05-01		Success
LMT-T05-02		Success
LMT-T05-03		Success
LMT-T05-04		Success
LMT-T05-05		Success
LMT-T05-06		Success
LMT-T06		Success
LMT-T07		Success
LMT-T08		Success
LMT-T09		Success
LMT-P-T10		Success

Collision Resolution Test Results

Test	Notes	Result
CR-T01		Success
CR-T02		Success
CR-T03		Success
CR-T04		Success
CR-T05		Success
CR-T06		Success
CR-T07		Success
CR-T08		Success
CR-T09		Success
CR-T10		Success
CR-T11		Success
CR-T12		Success
CR-T13		Success
CR-T14		Success
CR-T15		Success
CR-T16		Success
CR-T17		Success
CR-T18		Success
CR-T19		Success
CR-T20		Success
CR-T21		Success
CR-T22		Success
CR-T23		Success
CR-T24		Success
CR-T25		Success
CR-T26		Success
CR-T27		Success
CR-T28		Success
CR-T29		Success
CR-T30		Success
CR-T31		Success

Cubic Spline Interpolation Test Results

Test	Notes	Result
CSI-T-01		Success
CSI-T-02		Success
CSI-T-03		Success
CSI-T-04		Success
CSI-T-05		Success
CSI-T-06		Success
CSI-T-07		Success
CSI-P-T-08		Success
CSI-P-T-09		Success

Unit Conversion Tests

Test	Notes	Result
UC-T01		Success
UC-T02		Success
UC-T03		Success
UC-T04		Success
UC-T05		Success

Parser Tests

Test	Notes	Result
P-T01		Success
P-T02		Success
P-T03		Success

Spline Tool Browser Compatibility Tests

Firefox (Version 3.0.10)

Test	Notes	Result
BC-T01		Success
BC-T02		Success
BC-T03		Success
BC-T04		Success
BC-T05		Success
BC-T06		Success
BC-T07		Success
BC-T08		Success
BC-T09		Success
BC-T10		Success
BC-T11		Success
BC-T12		Success
BC-T13		Success

Opera (Version 9.64)

Note: For a detailed description on how the problem (see section 11) was solved, please refer to section 7.4 on page 80.

Test	Notes	Result
BC-T01		Success
BC-T02		Success
BC-T03		Success
BC-T04		Success
BC-T05		Success
BC-T06		Success
BC-T07		Success
BC-T08		Success
BC-T09		Success
BC-T10		Success
BC-T11		Success
BC-T12		Success
BC-T13		Success

Chrome (Version 2.0.172.28)

Test	Notes	Result
BC-T01		Success
BC-T02		Success
BC-T03	Local cookies are not allowed in Chrome[1]	Provider problem
BC-T04		Success
BC-T05		Success
BC-T06		Success
BC-T07		Success
BC-T08		Success
BC-T09		Success
BC-T10		Success
BC-T11		Success
BC-T12		Success
BC-T13		Success

Internet Explorer (Version 7.0.5730.13)

Note: If the Spline Tool is started from the local file system, e.g. `file://C:/SplineTool/index.html`, Internet Explorer does restrict the webpage from using scripts that could access and therefore harm the computer. An information bar informs the user about the behavior of the Internet Explorer. Since the Spline Tool is not equipped with malicious code, the Spline Tool should be allowed to execute the code fragments in the Internet Explorer. This can be achieved by clicking on the information bar and selecting *Allow Blocked Content...*.

[1] Issue 535: Support Cookies on file://

Test	Notes	Result
BC-T01		Success
BC-T02		Success
BC-T03	With the setting *Allow All Cookies* enabled, the Internet Explorer still does not support the toggle funtionality using local cookies. Also, the Internet Explorer does not show up an error regarding the JavaScript implementation of the cookie-handling routine. This misbehaviour will not be traced anymore	Fail
BC-T04		Success
BC-T05		Success
BC-T06		Success
BC-T07		Success
BC-T08		Success
BC-T09		Success
BC-T10		Success
BC-T11		Success
BC-T12		Success
BC-T13		Success

Konqueror (Version 4.2.2)

Test	Notes	Result
BC-T01		Success
BC-T02		Success
BC-T03	With the setting *Accept all cookies* enabled, the Konqueror still does not support the toggle-funtionality using cookies. Also, the Konqueror JavaScript Debugger does not show up an error regarding the implementation of the cookie-handling routine. This misbehaviour will not be traced anymore	Fail
BC-T04		Success
BC-T05		Success
BC-T06		Success
BC-T07		Success
BC-T08		Success
BC-T09		Success
BC-T10		Success
BC-T11		Success
BC-T12		Success
BC-T13		Success

Part IV

Project Management

Chapter 13

Software Development Plan

13.1 Changes

Date	Author	Change
22. Feb. 2009	StJu	Initial draft
23. Feb. 2009	StJu	Changed *Week 9+10* from *Construction* to *Elaboration*
26. Feb. 2009	GiAc	Added section *Organization*, *External Interfaces* and *Infrastructure*. Merged *Work Packages* into this document
28. Feb. 2009	GiAc	Added work packages Elaboration 1
05. Mar. 2009	StJu	Changes according to decisions from meeting[1]
13. Mar. 2009	StJu	Changed formatting and updated work packages of *Elaboration 1*: Architecture Document is no standalone document anymore and will be included in the Technical Report
14. Mar. 2009	GiAc	Added first drafts of the work packages for Elaboration 2
15. Mar. 2009	StJu	Planned Elaboration 2
18. Mar. 2009	StJu	Added deliverables of remaining iterations based on the project schedule
19. Mar. 2009	GiAc	Added new work packages to the second and current iteration of the Elaboration phase
27. Mar. 2009	StJu	Planned Elaboration 3
31. Mar. 2009	StJu	Updated work packages for the third iteration of the Elaboration phase
13. Apr. 2009	StJu	Planned Elaboration 4
27. Apr. 2009	StJu	Planned Construction 1
10. May 2009	StJu	Planned Construction 2
24. May 2009	StJu	Planned Construction 3
06. June 2009	StJu	Planned Transition 1
10. June 2009	StJu	Finished document

[1] http://stephanjud.ch/trac/robotic/wiki/OrganizationMinutes

13.2 Abbreviations

Abbreviations used for people in this Bachelor Thesis are described below.

Abbreviation	Name
StJu	Stephan Jud
GiAc	Giuseppe Accaputo
JoLe	Joas Leemann
HaHu	Hansjörg Huser

13.3 Organization

Both team members will be writing documentation, developing code and being responsible for testing.

Name	Email	Task
Stephan Jud	sjud@hsr.ch	Project team member
Giuseppe Accaputo	gaccaput@hsr.ch	Project team member

13.4 External Interfaces

Name	Email	Task
Hansjörg Huser	hhuser@hsr.ch	Adviser
Joas Leemann	joas.leemann@tecan.com	Contact at Tecan Schweiz AG
S. Zettel	- s.zettel@ascentiv.ch	Expert

13.5 Infrastructure

The following tools are used for development and documentation.

Name	Version
Visual Studio 2008	9.0
Microsoft .NET Framework	3.5
LaTeX[2]	3.1415926
Tortoise SVN	3.0
Trac[3] `[https://stephanjud.ch/trac/robotic]`	0.11.2

13.6 Process

An agile RUP has been chosen as software development process. It allows an iterative advancement and its adaptable elements can be applied according to the project's needs. Iterations used during the project are *Inception*, *Elaboration*, *Construction*, and *Transition*.

[2]Document markup language and document preparation system for the TeX typesetting program
[3]Web-based project management tool and interface to Subversion

After every iteration Work Packages which must be completed in the next iterations will be defined. Additionally, an Iteration Assessment will be created once an iteration has completed, presenting the general project situation and state of Work Packages.

The artifacts which are requested by the project stakeholders will be created. Before every iteration those required arifacts will be defined and noted under *Deliverables*.

Due to the large amount of research in this Bachelor Thesis the Elaboration phase will take four iterations.

Meetings, minutes, absences, guidelines, To-dos and questions will be noted on wiki pages within the Trac environment.

Meetings will be held every second week with the project stakeholders. Agenda items will be noted on the corresponding Trac page as well as in emails sent to the attendees before the meeting starts.

Time tracking is realized with a separate Excel spreadsheet.

13.6.1 Iteration Plan

[a]**M1:** Overall architecture and functionality for the 3D engine has been defined.
[b]**M2:** Move related algorithms have been designed. Requirements and test cases have been defined.
[c]**M3:** Move tool has been created, allowing fast debugging and testing.
[d]**M4:** Product delivery and acceptance

13.7 Work Packages

13.7.1 Inception 1

ID	Person	Package	Dep.[4]	Cost[5]
I1.1	StJu	Set up development infrastructure including Subversion and Trac	-	5
I1.2	All	Create Software Development Plan	-	8
I1.3	All	Create Iteration Assessment for first iteration of the Inception phase	-	2
I1.4	All	Gather basic information about target environment	-	3
I1.5	All	Create list of possible relevant algorithms	I1.4	20
I1.6	All	Add milestones to Software Development Plan	I1.2	3
I1.7	All	Define Work Packages for first iteration of the Elaboration phase	-	2
I1.8	All	Start writing Technical Report	I1.1	35
RT.1[6]	All	Meetings	-	6

Deliverables

- Project Schedule
- Work Packages for the first iteration of the Elaboration phase
- Iteration Assessment of the Inception phase

[4]Dependency on other Work Packages
[5]Effort in hours for people specified in column *Person*
[6]Recurring Task (RT): Task whose execution spans over multiple iterations or recurs periodically

13.7.2 Elaboration 1

ID	Person	Package	Dep.	Cost
E1.1	All	Search for more path finding algorithms	-	10
E1.2	All	Evaluate and study gathered algorithms in detail	I1.5	22
E1.3	All	Create the Software Architecture Document	-	5
E1.4	All	Create prototype for the software architecture	E1.4	5
E1.5	All	Define test scenarios where algorithms can be evaluated on	I1.4	5
E1.6	All	Define the most important requirements for the project	-	5
E1.7	All	Define Work Packages for the second iteration of the Elaboration phase	-	2
RT.1	All	Meetings	-	6
RT.2	All	Update Technical Report	I1.8	18
RT.3	All	Update Software Development Plan	I1.2	6

Deliverables

- Iteration Assessment of the first iteration in the Elaboration phase
- Work Packages for the second iteration in the Elaboration phase
- First version of the Requirements Document

13.7.3 Elaboration 2

ID	Person	Package	Dep.	Cost
E2.1	All	Create software architecture prototype including basic control flow	E1.4	26
E2.2	All	Define architectural design	-	10
E2.3	All	Detailed requirements analysis	-	5
E2.4	All	Evaluate relevant algorithms with regard to the target system	E1.2	15
RT.1	All	Meetings	-	6
RT.2	All	Update Technical Report	-	16
RT.3	All	Update Software Development Plan	-	5

Deliverables

- Iteration Assessment of the second iteration in the Elaboration phase
- Work Packages for the third iteration in the Elaboration phase
- Architecture prototype

13.7.4 Elaboration 3

ID	Person	Package	Dep.	Cost
E3.1	All	Create draft implementation of path finding and collision avoidance	-	12
E3.2	All	Define which algorithms shall be used	E2.4	12
E3.3	All	Define and implement test scenarios	E2.3	17
E3.4	All	Finish describing the architectural design of the engine	E2.5	17
RT.1	All	Meetings	-	6
RT.2	All	Update Technical Report	-	10
RT.3	All	Update Software Development Plan	-	5
RT.4	All	Check state of every document and update document if necessary	-	6

Deliverables

- Iteration Assessment of the third iteration in the Elaboration phase
- Work Packages for the fourth iteration in the Elaboration phase
- Test Document
- Requirements Document
- The Technical Report specifies the used algorithms in detail

13.7.5 Elaboration 4

ID	Person	Package	Dep.	Cost
E4.1	All	Implement collision detection / avoidance	E3.1	8
E4.2	GiAc	Evaluate how to compute round paths around obstacles	-	10
E4.3	StJu	Design how to extend collision avoidance with support for dependent subdevices	E3.4	4
E4.4	All	Improve the Viewer Tool	-	10
E4.5	StJu	Evaluate the benefit of using *late move updates*	E3.4	5
E4.6	All	Use buffer to complete unfinished tasks	-	15
RT.1	All	Meetings	-	4
RT.2	All	Update Technical Report	-	10
RT.3	All	Update Software Development Plan	-	10
RT.4	All	Check state of every document and update document if necessary	-	8

Deliverables

- Iteration Assessment of the fourth iteration in the Elaboration phase
- Work Packages for the first iteration in the Construction phase
- Implemented tests
- Working 2D collision avoidance implementation

13.7.6 Construction 1

ID	Person	Package	Dep.	Cost
C1.1	StJu	Implement collision detection / avoidance	E4.1	12
C1.2	GiAc	Implement *waypoints to spline* converter	-	20
C1.3	All	Improve the Viewer Tool	-	8
C1.4	StJu	Extend collision avoidance with support for dependent subdevices	E4.3	10
C1.5	All	Complete project management documents so they can be reviewed	-	15
RT.1	All	Meetings	-	4
RT.2	All	Update Technical Report	-	10
RT.3	All	Update Software Development Plan	-	5

Deliverables

- Iteration Assessment of the first iteration in the Construction phase
- Work Packages for the second iteration in the Construction phase
- Project management documents
- Move Tool

13.7.7 Construction 2

ID	Person	Package	Dep.	Cost
C2.1	StJu	Extend collision avoidance with support for ranges	C1.4	10
C2.2	GiAc	Adapt route smoothing algorithm to 3D waypoints	C1.2	20
C2.3	StJu	Improve the request parser	-	15
C2.4	All	Describe software architecture	-	5
C2.5	All	Prepare engine for realistic environments	-	5
RT.1	All	Meetings	-	4
RT.2	All	Update Technical Report	-	20
RT.3	All	Update Software Development Plan	-	5

Deliverables

- Draft of the technical report
- Iteration Assessment of the second iteration in the Construction phase
- Work Packages for the third iteration in the Construction phase

13.7.8 Construction 3

ID	Person	Package	Dep.	Cost
C3.1	All	Describe software architecture	-	6
C3.2	StJu	Prepare engine for realistic environments	-	6
C3.3	GiAc	Create test report template and perform first test	-	5
C3.4	All	Define “Aims and Objectives”	-	3
C3.5	All	Prepare management summary	-	4
C3.6	All	Add cubic spline support to move engine	-	4
C3.7	GiAc	Gather information about document printing	-	5
C3.8	All	Use buffer to complete unfinished tasks	-	28
RT.1	All	Meetings	-	8
RT.2	All	Update Technical Report	-	10
RT.3	All	Update Software Development Plan	-	5

Deliverables

- Thesis Description
- Iteration Assessment of the third iteration in the Construction phase

13.7.9 Transition 1

ID	Person	Package	Dep.	Cost
T1.1	All	Review documents for release (Layout and spell checking)	-	18
T1.2	All	Review programs for release	-	8
T1.3	All	Write personal reports	-	4
T1.4	All	Finish Management Summary	-	2
T1.5	All	Print documents and create CD	-	4
T1.6	All	Finish A0 poster	-	2
T1.7	All	Update Abstract	-	2
T1.8	All	Hand in Thesis description	-	4
RT.1	All	Meetings	-	4

Deliverables

- All Iteration Assessments including the one of the Transition phase
- Project Management documents
- Technical Report
- Test Documents
- Relevant Wiki Pages as documents
- Tools and 3D Engine
- Tool descriptions
- Management Summary
- Personal Reports
- Thesis Description

Chapter 14

Iteration Assessments

14.1 Inception 1

Iteration Overview

Phase	Inception
Iteration	Iteration 1
From	Week 1 (16.02 - 20.02.2009)
To	Week 2 (23.02 - 27.02.2009)
Span	2 weeks

Attendees

GiAc
StJu

Created Artifacts

Artifact
Software Development Plan
Risk Management
Technical Report

Iteration Objectives Reached

Current Iteration

ID	Person	Package	Priority	State
I1.1	StJu	Set up development infrastructure including Subversion and Trac	1	OK
I1.2	All	Create Software Development Plan	1	OK
I1.3	All	Create Iteration Assessment for first iteration of the Inception phase	1	OK
I1.4	All	Gather basic information about target environment	2	OK
I1.5	All	Create list of possible relevant algorithms	1	OK
I1.6	All	Add milestones to Software Development Plan	1	OK
I1.7	All	Define working packages for first iteration of the Elaboration phase	1	OK
I1.8	All	Start writing Technical Report	1	OK

Next Iteration

ID	Person	Package	Priority
E1.1	All	Continue writing Technical Report	1
E1.2	All	Continue to search for relevant algorithms	1
E1.3	All	Evaluate and study gathered algorithms in detail	1
E1.4	All	Create the Software Architecture Document	2
E1.5	All	Create prototype for software architecture	2
E1.6	All	Define test scenarios where algorithms can be evaluated on	1
E1.7	All	Define the most important requirements for the project	1
E1.8	All	Define work packages for the second iteration of the Elaboration phase	1

Decisions

None

Results Relative to Evaluation Criteria

Evaluation Criteria	Iteration Results
Create a Software Development Plan defining milestones, RUP phases and iteration details.	The Software Development Plan has been created and the milestones have been defined. The iteration length has been set to two weeks. The iteration plan has been defined in the Software Development Plan. The iteration plan consists of one iteration for the Inception phase, four iterations for the Elaboration phase, three iterations for the Construction phase and one iteration for the Transition phase.
Evaluate possible algorithms for path finding and collision avoidance. Create a list of the possible relevant algorithms and describe their functionality.	Possible algorithms have been found and described in the Technical Report.
Gather basic information about the target environment	The target environment was presented in a short introduction, including its functionality and its elements. The simulator software was also shown.

Adherence to Plan

The iteration executed according to plan completing on schedule. All tasks could be completed as planned and there are no work packages left for the next iteration. Much effort has been invested in this early stage into the Technical Report which already shows its basic structure and organization.

The infrastructure has been set up and is ready and essential domain knowledge has been gathered so fast progress is expected in the next iteration.

14.2 Elaboration 1

Iteration Overview

Phase	Elaboration
Iteration	Iteration 1
From	Week 3 (02.03 - 06.03.2009)
To	Week 4 (09.03 - 13.03.2009)
Span	2 weeks

Attendees

GiAc
StJu

Created Artifacts

Artifact
Requirements Analysis
Prototype

Iteration Objectives Reached

Current Iteration

ID	Person	Package	Priority	State
E1.1	All	Search for more (relevant) algorithms	1	OK
E1.2	All	Evaluate and study gathered algorithms in detail	1	OK
E1.3	All	Create the Software Architecture Document	2	OK
E1.4	All	Create prototype for the Software Architecture	2	OK
E1.5	All	Define test scenarios where algorithms can be evaluated on	1	OK
E1.6	All	Define the most important requirements for the project	1	OK
E1.7	All	Define Work Packages for the second iteration of the Elaboration phase	1	OK

Next Iteration

ID	Person	Package	Priority
E2.1	All	Create software architecture prototype including basic control flow	1
E2.2	All	Define architectural design	2
E2.3	All	Detailed requirements analysis	1
E2.4	All	Evaluate relevant algorithms with regard to the target system	1
RT.1	All	Meetings	1
RT.2	All	Update Technical Report	1
RT.3	All	Update Software Development Plan	1

Decisions

There will be no independent Software Architecture Document. Its content will be included in the already existing Technical Report.

Results Relative to Evaluation Criteria

Evaluation Criteria	Iteration Results
Search for more algorithms and evaluate the algorithm's functionality	The search for more algorithms has been continued. New algorithms have been found and evaluated. A description of every algorithm found has been added to the Technical Report, including an exemplary search demonstrating the algorithm's procedure.
Create Software Architecture Document	An initial version of the Software Architecture Document has been written. It has been decided that this document will be moved into the Technical Report (Chapter *Realization*, subsection *Design*).
Create prototype	A prototype has been created, including a graphical user interface for testing purposes. Basic control flow has been implemented demonstrating the involved components.
Define test cases for algorithms	A Test Plan has been created with test cases for collision resolution algorithms.
Define the most important requirements for the project	A first version of the Requirements Document has been created. The most important requirements have been noted and have to be reviewed and discussed with JoLe.

Adherence to Plan

The iteration executed according to plan completing on schedule. All tasks could be completed as planned.

New relevant algorithms have been found. Search and evaluation of new algorithms could be completed as planned.

A first prototype has been created, including a graphical user interface for testing purposes.

14.3 Elaboration 2

Iteration Overview

Phase	Elaboration
Iteration	Iteration 2
From	Week 5 (16.03 - 20.03.2009)
To	Week 6 (23.03 - 27.03.2009)
Span	2 weeks

Attendees

GiAc
StJu

Created Artifacts

Artifact
Software Architecture Document (inside Technical Report)

Iteration Objectives Reached

Current Iteration

ID	Person	Package	Priority	State
E2.1	All	Create software architecture prototype including basic control flow	1	OK
E2.2	All	Define architectural design	2	OK
E2.3	All	Detailed requirements analysis	1	OK
E2.4	All	Evaluate relevant algorithms with regard to the target system	1	OK

Next Iteration

ID	Person	Package	Priority
E3.1	All	Create draft implementation of path finding and collision avoidance	1
E3.2	All	Define which algorithms shall be used	1
E3.3	All	Define and implement test scenarios	1
E3.4	All	Finish describing the architectural design of the engine	1
RT.1	All	Meetings	1
RT.2	All	Update Technical Report	1
RT.3	All	Update Software Development Plan	1
RT.4	All	Check state of every document and update document if necessary	1

Decisions

The Requirements Document does not need to be extended. The currently coarse requirements are sufficient.

Results Relative to Evaluation Criteria

Evaluation Criteria	Iteration Results
Architectural prototype	The logical flow and included components of the engine have been defined. Additionally, a first implementation has been created for feasibility purposes. The algorithms developed from the Technical Report have been partially implemented and tested for practicability. Ambiguities which have got evident during realization have been corrected in the Technical Report.
Requirements analysis	Because the main focus of the project is to provide strategies and algorithms capable of serving as robotic engine, the currently defined requirements have not been extended. The existing mostly non functional requirements do give enough information for creating the system.
Evaluate algorithms	Two new algorithms, *Potential Fields* and *Voronoi Diagram* have been found. They are not likely to be implemented directly but could give valuable approaches which can be applied to the existing components

Adherence to Plan

The iteration executed according to plan completing on schedule. All tasks could be completed as planned.

The existing prototype was extended showing feasibility of approaches defined in the Technical Report document.

14.4 Elaboration 3

Iteration Overview

Phase	Elaboration
Iteration	Iteration 3
From	Week 7 (30.03 - 03.04.2009)
To	Week 8 (06.04 - 10.04.2009)
Span	2 weeks

Attendees

GiAc
StJu

Created Artifacts

Artifact
Test Document
Requirements Document

Iteration Objectives Reached

Current Iteration

ID	Person	Package	Priority	State
E3.1	All	Create draft implementation of path finding and collision avoidance	1	OK
E3.2	All	Define which algorithms shall be used	2	OK
E3.3	All	Define and implement test scenarios	1	OK
E3.4	All	Finish describing the architectural design of the engine	1	OK

Next Iteration

ID	Person	Package	Priority
E4.1	All	Implement collision detection / avoidance	1
E4.2	GiAc	Evaluate how to compute round paths around obstacles	1
E4.3	StJu	Design how to extend collision avoidance with support for dependent subdevices	2
E4.4	All	Improve the Viewer Tool	2
E4.5	StJu	Evaluate the benefit of using *late move updates*	1
E4.6	All	Use buffer to complete unfinished tasks	1
RT.2	All	Update Technical Report	1
RT.3	All	Update Software Development Plan	1
RT.4	All	Check state of every document and update document if necessary	1

Decisions

None

Results Relative to Evaluation Criteria

Evaluation Criteria	Iteration Results
Create a draft implementation of the path finding and collision avoidance	A first draft implementation of the path finding has been completed. For the graph traversal the Dijkstra algorithm has been implemented. Additionally, an initial version of collision avoidance has been implemented.
Define which algorithms shall be used	The basic algorithms needed to perform moves are defined. Detailed evading mechanism and strategies have not been defined yet though.
Define and implement test scenarios	For the graph traversal a total of sixteen test cases plus one performance test have been defined and implemented. For the collision resolution a total of thirty-three test cases have been defined.

Adherence to Plan

The iteration executed according to plan completing on schedule. All tasks could be completed as planned.

An initial version of the algorithms has been realized. The test document has been extended and a first set of test cases have been implemented using NUnit.

14.5 Elaboration 4

Iteration Overview

Phase	Elaboration
Iteration	Iteration 4
From	Week 9 (13.04 - 17.04.2009)
To	Week 10 (20.04 - 24.04.2009)
Span	2 weeks

Attendees

GiAc
StJu

Created Artifacts

Artifact
Project Management documents

Iteration Objectives Reached

Current Iteration

ID	Person	Package	Priority	State
E4.1	All	Implement collision detection / avoidance	1	OK
E4.2	GiAc	Evaluate how to compute round paths around obstacles	1	OK
E4.3	StJu	Design how to extend collision avoidance with support for dependent subdevices	2	OK
E4.4	All	Improve the viewer tool	2	OK
E4.5	StJu	Evaluate the benefit of using *late move updates*	1	OK
E4.6	All	Use buffer to complete unfinished tasks	1	OK

Next Iteration

ID	Person	Package	Priority
C1.1	StJu	Implement collision detection / avoidance	1
C1.2	GiAc	Implement *waypoints to spline* converter	1
C1.3	All	Improve the viewer tool	1
C1.4	StJu	Extend collision avoidance with support for dependent subdevices	1
C1.5	All	Complete project management documents	1
RT.1	All	Meetings	1
RT.2	All	Update Technical Report	1
RT.3	All	Update Software Development Plan	1
RT.4	All	Check state of every document and update document if necessary	1

Decisions

Project Management documents will reviewed by the adviser in the next iteration.

Results Relative to Evaluation Criteria

Evaluation Criteria	Iteration Results
Improve Viewer Tool	The tool is now ready to be used for debugging and testing. An autocompletion mechanism has been added which allows faster query creation. The overall usability also has been improved.
Evaluate round moves	A mathematical solution was found which enables the definition of smooth round moves. It is planned to describe round moves with the help of spline curves.
Use buffer	The documentation has been extensively reviewed and improved. No major changes in documentation are expected in further iterations.
Extend Collision Avoidance	Collision avoidance has been implemented according to the definitions in the Technical Report. The major approaches defined in the report fulfill the expectations. The current code base will be incrementally extended with various features during the next iterations.

Adherence to Plan

The iteration executed according to plan completing on schedule. All tasks could be completed as planned. However, the planned buffer must be used to update and review all documents in this iteration.

All defined unit tests could be implemented. They will be used intensely in the next iteration where the main focus will be improving collision avoidance.

14.6 Construction 1

Iteration Overview

Phase	Construction
Iteration	Iteration 1
From	Week 11 (27.04 - 02.05.2009)
To	Week 12 (04.05 - 08.05.2009)
Span	2 weeks

Attendees

GiAc
StJu

Created Artifacts

None

Iteration Objectives Reached

Current Iteration

ID	Person	Package	Priority	State
C1.1	StJu	Implement collision detection / avoidance	1	OK
C1.2	GiAc	Implement *waypoints to spline* converter	1	OK
C1.3	All	Improve the viewer tool	1	OK
C1.4	StJu	Extend collision avoidance with support for dependent subdevices	1	OK
C1.5	All	Complete project management documents	1	OK

Next Iteration

ID	Person	Package	Priority
C2.1	StJu	Extend collision avoidance with support for ranges	1
C2.2	GiAc	Adapt route smoothing algorithm to 3D waypoints	1
C2.3	StJu	Improve the request parser	2
C2.4	All	Describe software architecture	1
C2.5	All	Prepare engine for realistic environments	1

Decisions

The project management documents are formal complete. Wiki pages will also be included in final document. The technical report will be reviewed in the next iteration.

Results Relative to Evaluation Criteria

Evaluation Criteria	Iteration Results
Support for dependent subdevices	The extensions to support dependent subdevices required have been described and implemented. However, there is still functionality missing to compute realistic movements. Those extensions will be implemented in this iteration.
Complete project management documents	The documents have been carefully reviewed and presented to HaHu. Its current composition and degree of detail is appropriate for this Thesis.
Splines converter	The cubic spline formula has been implemented. There were concerns about how round moves should be described so a device can follow the computed curve in the desired correctness. One approach has been developed and will be presented in the next iteration.

Adherence to Plan

The iteration executed according to plan completing on schedule. All tasks could be completed as planned. Emerging issues like how to describe round moves and collision avoidance related ones could be resolved. They influenced the work packages for the next iteration however.

14.7 Construction 2

Iteration Overview

Phase	Construction
Iteration	Iteration 2
From	Week 13 (11.05 - 15.05.2009)
To	Week 14 (18.05 - 22.05.2009)
Span	2 weeks

Attendees

GiAc
StJu

Created Artifacts

Artifact
Draft of the technical report

Iteration Objectives Reached

Current Iteration

ID	Person	Package	Priority	State
C2.1	StJu	Extend collision avoidance with support for ranges	1	OK
C2.2	GiAc	Adapt route smoothing algorithm to 3D waypoints	1	OK
C2.3	StJu	Improve the request parser	2	OK
C2.4	All	Describe software architecture	1	NOK
C2.5	All	Prepare engine for realistic environments	1	NOK

Next Iteration

ID	Person	Package	Priority
C3.1	All	Describe software architecture	1
C3.2	StJu	Prepare engine for realistic environments	1
C3.3	GiAc	Create test report template and perform first test	1
C3.4	All	Define “Aims and Objectives”	1
C3.5	All	Prepare management summary	1
C3.6	All	Add cubic spline support to move engine	1
C3.6	GiAc	Gather information about document printing	2
C3.7	All	Use buffer to complete unfinished tasks	1

Decisions

Cubic Spline interpolation formula should be implemented that it can serve as base for a new hardware supported smooth curve implementation.

Results Relative to Evaluation Criteria

Evaluation Criteria	Iteration Results
Code Review	The existing code base has been reviewed and edited where necessary.
Ranges support	The robotic engine now supports range constraints. They can be either posed by a device itself or by constraints of the parent devices.
Route Smoothing	The cubic spline algorithm and how to apply it to devices has been extensively researched. For problems like missing hardware support, different axis speeds and acceleration limitations approaches have been developed. Additionally, a dedicated tool has been created which allows fast and easy visualization and verification of splines.
Unit handling	The units used internally for computation are not dependent of the incoming values anymore. A conversion layer has been implemented which frees the different components of having to use the same units.

Adherence to Plan

Not all tasks could be completed. The work on those tasks will be continued in the next iteration. Because the next iteration contains a buffer, no delays are expected because of the additional tasks.

14.8 Construction 3

Iteration Overview

Phase	Construction
Iteration	Iteration 3
From	Week 15 (25.05 - 29.05.2009)
To	Week 16 (02.06 - 05.06.2009)
Span	2 weeks

Attendees

GiAc
StJu

Created Artifacts

Artifact
Thesis Description

Iteration Objectives Reached

Current Iteration

ID	Person	Package	Priority	State
C3.1	All	Describe software architecture	1	OK
C3.2	StJu	Prepare engine for realistic environments	1	OK
C3.3	GiAc	Create test report template and perform first test	1	OK
C3.4	All	Define “Aims and Objectives”	1	OK
C3.5	All	Prepare management summary	1	OK
C3.6	All	Add cubic spline support to move engine	1	OK
C3.6	GiAc	Gather information about document printing	2	OK
C3.7	All	Use buffer to complete unfinished tasks	1	OK

Next Iteration

ID	Person	Package	Priority
T1.1	All	Review documents for release. Including layout and spell checking	1
T1.2	All	Review programs for release	1
T1.3	All	Write personal reports	1
T1.4	All	Finish Management Summary	1
T1.5	All	Print documents and create CD	1
T1.6	All	Finish A0 poster	1
T1.7	All	Update Abstract	1
T1.8	All	Hand in Thesis description	1

Decisions

None

Results Relative to Evaluation Criteria

Evaluation Criteria	Iteration Results
Spline Tool	A tool has been created which allows an easy visualization of cubic spline. The properties of splines could be demonstrated to the stakeholders with help of that tool.
Route Smoothing	Much time has been invested to create a detailed description of how cubic spline curves can be enabled on the target platform.
Technical Report	The technical report has been extensively reviewed. Chapters have been reorganized or have been moved to more appropriate places leading to a more understandable composition.
Architecture	The architectural model has been described. As discussed with the stakeholders, just interesting and non obvious cases were described.
Wiki Handling	A dedicated tool has been created which allows exporting content of Wiki pages into the documents which will be handed in.

Adherence to Plan

The iteration executed according to plan completing on schedule. A detailed plan has been created on the Wiki page describing deadlines for every day of the next iteration.

14.9 Transition 1

Iteration Overview

Phase	Transition
Iteration	Iteration 1
From	Week 17 (08.06.2009)
To	Week 17 (12.06.2009)
Span	2 weeks

Attendees

GiAc
StJu

Created Artifacts

Artifact
All Iteration Assessments including the one of the Transition phase
Project Management documents
Technical Report
Test Documents
Relevant Wiki Pages as documents
Tools and 3D Engine
Tool descriptions
Management Summary
Personal Reports
Thesis Description

Iteration Objectives Reached

Current Iteration

ID	Person	Package	Priority	State
T1.1	All	Review documents for release. Including layout and spell checking	1	OK
T1.2	All	Review programs for release	1	OK
T1.3	All	Write personal reports	1	OK
T1.4	All	Finish Management Summary	1	OK
T1.5	All	Print documents and create CD	1	OK
T1.6	All	Finish A0 poster	1	OK
T1.7	All	Update Abstract	1	OK
T1.8	All	Hand in Thesis description	1	OK

Decisions

None

Results Relative to Evaluation Criteria

Evaluation Criteria	Iteration Results
Review documents for release	All documents have been successfully reviewed by both the team members and the responsible authorities at Tecan Schweiz AG for the approval of the documentation, Rainer Kerkmann and Joas Leemann.
Review programs for release	All the programs have been successfully reviewed by both the team members.
Finish relevant documents	All the relevant documents have been created and printed, and are ready to be handed in.

Adherence to Plan

The iteration executed according to plan completing on schedule. The Bachelor Thesis is ready to be handed in.

Chapter 15

Risk Management

15.1 Changes

Date	Author	Change
22. Feb. 2009	StJu	Initial version
23. Feb. 2009	StJu	Set likelihood of risk *R.1* to *30*
27. Feb. 2009	GiAc	Added new risks (R.9, R.10, R.11)
05. Mar. 2009	StJu	Added fallback actions in risk description. Removed R.9 (Refactoring-Risk) because it is not a real risk
25. Mar. 2009	StJu	A simple simulator has been implemented reducing severities of R.10 and R11 to *40* and *10*
13. Apr. 2009	StJu	• Evaluation of algorithms is mostly completed. Reducing likelihood of risk *R.3* from *80* to *20* • The existing prototypes are capable of handling generic situations. Reducing likelihood of *R.5* from *40* to *20* • Changes in the target system are unlikely to happen. Reducing likelihood of *R.4* from *20* to *10*
10. May 2009	StJu	• Set likelihood of *R.4* and *R.3* from both *20* to *0* because the requirements are clear • Set likelihood of *R.5* from *40* to *20* because the algorithm can handle generic scenes • Set likelihood of *R.8* from *80* to *60* because many test cases have already been implemented
27. May 2009	StJu	Set likelihood of *R.5* from *20* to *0* because the algorithm is generic. Likelihood of *R7* is set from *20* to *0* because the outcome is fairly clear.
09. June 2009	StJu	Set likelihood of *R6,R8,R10* and *R11* to *0* because the presented solution fulfills the requirements.
10. June 2009	StJu	Set likelihood of *R1* and *R2* to *0* because the release is over.

15.2 Risks

Risks which might endanger project's success sorted according to their impact[1]

ID	Risk	Consequence	Countermeasure	Impact
R.8	Error situations are not represented in test cases	Unexpected and erroneous behavior of the software shows up during execution	**Prevention:** Use test-driven development as primary software development technique to avoid errors and plan regular meetings to discuss possible error situations **Fallback:** Plan and enforce early system tests.	$2 * 0 = 0$
R.6	The algorithm does not match performance expectations	The algorithm designed limits system throughput	**Prevention:** Gather information about how the algorithm is used and set up automatic performance checks **Fallback:** Remove complexity in the algorithm by removing checks and other complex parts of the algorithm. Profile the algorithm and optimize costly parts as good as possible.	$2 * 0 = 0$

Continued on next page...

[1] **Impact:** Severity *[0-3]* $*$ Likelihood *[%]*

Continued

ID	Risk	Consequence	Countermeasure	Impact
R.10	The simulator does not match exactly the target environment in its functionality and execution	The algorithm cannot be applied to the target environment because of discrepancies between the simulator and the target environment. The algorithm could cause problems on the target environment.	**Prevention:** Clarify the actual state of the simulator and its functionality. Clarify also if there are differences between the simulator and the target environment. **Fallback:** Test implementation directly on the target environment and/or set up test environment which is able to determine whether a collision occurs.	$1 * 0 = 0$
R.1	Illness of team members	Shortened team and delays in the project schedule	**Prevention:** Periodically discuss project state so partner can take over work. Additionally plan time buffers. **Fallback:** Remove low-priority features	$3 * 0 = 0$
R.2	Data loss	Source code or documents get lost	**Prevention:** Check-in work often and backup server regularly **Fallback:** Restore latest backups and, if needed, rewrite missing documents and source code. If necessary increase construction phase or remove low-priority features	$3 * 0 = 0$

Continued on next page...

Continued

ID	Risk	Consequence	Countermeasure	Impact
R.11	Software bug in the simulator	The algorithms cannot be tested on the simulator and the bug fixing causes delays in the project schedule	**Prevention:** The simulator should be tested and its functionality should be approved during the Elaboration phase so no delays in the Inception phase caused by appearing bugs in the simulator must be expected. **Fallback:** Directly test on the target system. Discuss proceeding with stakeholders.	$2 * 0 = 20$
R.7	Development and requirements show discrepancies	The algorithm developed does not satisfy expectations	**Prevention:** Periodically verify project state with stakeholders **Fallback:** Increase construction phase or remove low-priority features	$2 * 0 = 0$
R.5	The algorithm developed is too focused on the target system	The algorithm developed cannot be reused for future versions of the environment	**Prevention:** The initial algorithm must work with basic geometrical objects. If possible, test the algorithm on other systems to avoid focusing on the target system. **Fallback:** Rewrite algorithm and/or implement different classes of algorithms. Choose one algorithm which should be continued. Write the algorithm to work efficiently on the available target system.	$2 * 0 = 0$

Continued on next page...

Continued

ID	Risk	Consequence	Countermeasure	Impact
R.3	The evaluation of the algorithms takes more time than expected	The Elaboration phase has to be extended causing delays in the project schedule	**Prevention:** Define algorithms that can be considered and examine carefully the algorithms, discarding algorithms that are not suitable **Fallback:** Remove features and/or increase elaboration phase. Discuss proceeding with stakeholders.	$1 * 0 = 0$
R.4	The target system changes during the project	Assumptions and analysis become invalid	**Prevention:** Periodically check state of the target environment **Fallback:** Define with stakeholders which features must be removed or increase construction phase	$2 * 0 = 0$

Chapter 16

Requirements

16.1 Aims and Objectives

A robotic engine capable of planning and executing movements for robotic arms at Tecan Schweiz AG shall be developed. The engine will not be used in a productive environment right after the projects ends. In fact, it will serve as basis for further developments.

Therefore, the requirements are kept rather general and are mainly based on the original thesis description.

> Tecan Schweiz stellt Laborgeräte mit Pipettierrobotern her. Hauptaufgabe der Bachelorarbeit ist es, ein Konzept zu entwickeln, welches einen beliebigen, unter Umständen parallelen Bewegungsablauf der einzelnen Roboter kollisionsfrei planen und ausführen vermag.
>
> In einer ersten Phase sollen verschiedene Algorithmen und Strategien für die Bewegungssteuerung analysiert und auf ihre Zweckmässigkeit überprüft werden.
>
> Danach soll ein Konzept erarbeitet werden, wie solch eine Bewegungssteuerungseinheit implementiert werden kann. Das Konzept sollte:
>
> - Die Steuerungslogik exakt beschreiben
> - Die benötigte Softwarearchitektur definieren
>
> In einem letzten Schritt soll das erarbeitete Konzept als Prototyp implementiert und für das existierende Softwareframework vorbereitet werden.

16.2 Requirements

16.2.1 Functionality

FU1: Path Finding

A set of algorithms shall be defined which are capable of finding the shortest collision free path of a roboter from a start position to an end position.

FU2: Collision Avoidance

An algorithm shall be defined which is able to compute collision-free paths for a roboter within the target system. No generic strategies shall be used. Instead, an evading path depending on the specific collision situation shall be computed making optimal use of the capabilities of all involved components.

FU3: Collision Resolution

An algorithm shall be defined which is capable of solving roboter collisions should they be detected. The resolution mechanism shall consider the costs of evading mechanism on all components involved and choose the one which causes least run time delay.

16.2.2 Reliability

Equal requests must result in equal action given the same state of the environment.

16.2.3 Usability

Because the robotic engine is not directly exposed to the end user, there is no need for a graphical user interface for controlling the engine.

16.2.4 Performance

Calculation Time

The calculation time when computing a move shall not show exponential growth. Small requests which just changes positions for just a few millimeters must cause a negligible computation delay.

Memory Usage

There are no specific limitations regarding memory usage.

16.2.5 Supportability

Expandability

The move engine can be tailored to replace each algorithm by a better algorithm if needed

Generic Algorithms

The engine shall make use of already existing device drivers which expose physical properties. Those values must be interpreted by the engine to plan, coordinate and perform movements of multiple devices. No additional logic may be added to the device drivers.

Configurability

The move engine shall be configurable by other components. Exposed settings shall be:

- The internal calculation unit
- Minimal distance between objects

- Algorithmic behavior
  - Default evading axis

16.2.6 Design Constraints

Framework

The underlying framework, which is responsible for the communication has already been implemented. Its API and the used communication protocol are given and may not be changed.

Environment

The robotic engine must be written in C# running on .NET version 3.5. The application will run on a desktop computer.

Chapter 17

Quality Management

17.1 Code Reviews

Beginning with the Construction phase, a code review is held once in a week in which a systematic examination of the source code is conducted. The goal of the code reviews is to minimize mistakes in the code like bugs and performance issues and to increase the overall quality of the code by applying refactoring patterns where possible.

For the code reviews the *over-the-shoulder* approach is used which consists of a commented guidance through the code by the author. The code metrics generation in Visual Studio 2008 serves as additional guidance for possible problems.

17.2 Document Reviews

Beginning with the Elaboration phase, each committed document is counterchecked by a team member. The goal of this method is to maintain a high quality standard and up-to-date state of documents.

17.3 Bug Tracking

Detected bugs in the code are reported and tracked using the ticketing system provided by Trac[1]. Bugs must be added with a detailed description. Steps how to reproduce must be added as well if applicable. The bug priority is determined by the team members and serves as indicator which bugs are critical and must be resolved with high priority.

With help of a ticketing system it can be ensured that no bugs get forgotten. The list of open bugs helps to keep an overview whether the project state is according to the iteration plan.

17.4 Testing

A set of tests must be described until the last Elaboration phase[2] verifying correct interaction between modules and intended data processing of all major components. Tests in that

[1] For more information refer to the Software Development chapter
[2] See chapter Test Plan

document must be defined by the developers while specifying and describing the components.

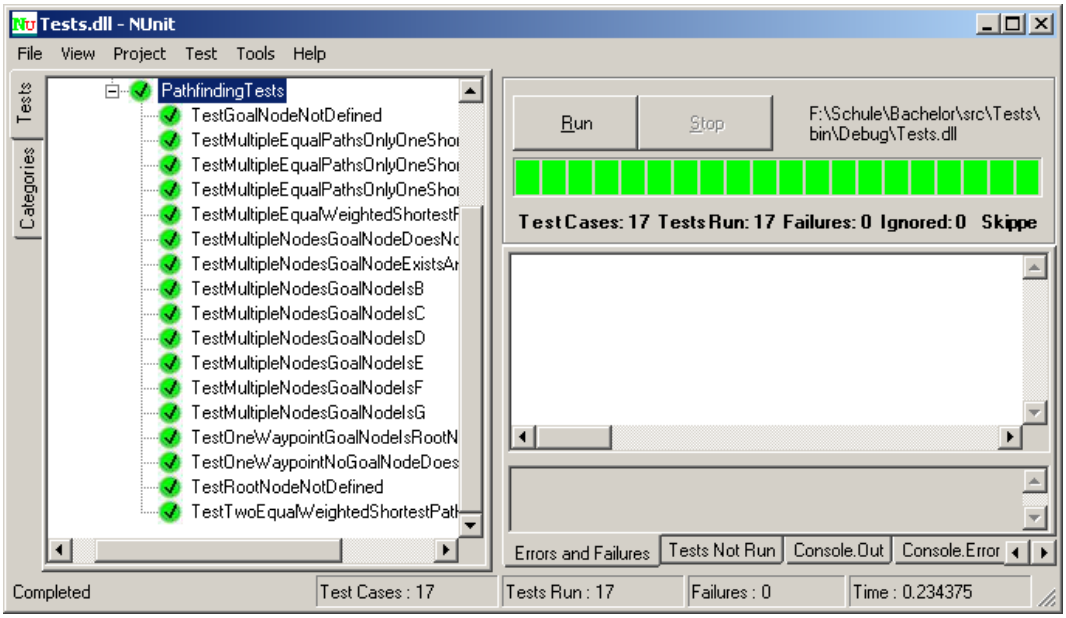
At the end of every Construction and the Transition phase, all cases in the test plan must be tested. The project revision and the corresponding test result must be noted in a separate document[3].

17.5 Tools

17.5.1 NUnit

NUnit is a free utility for unit testing .NET classes. Before a change set is committed, all existing unit tests must be executed and pass. The tests which do not pass because of an ongoing change must be marked with the *Ignore* tag.

Unit tests must be written for core components.



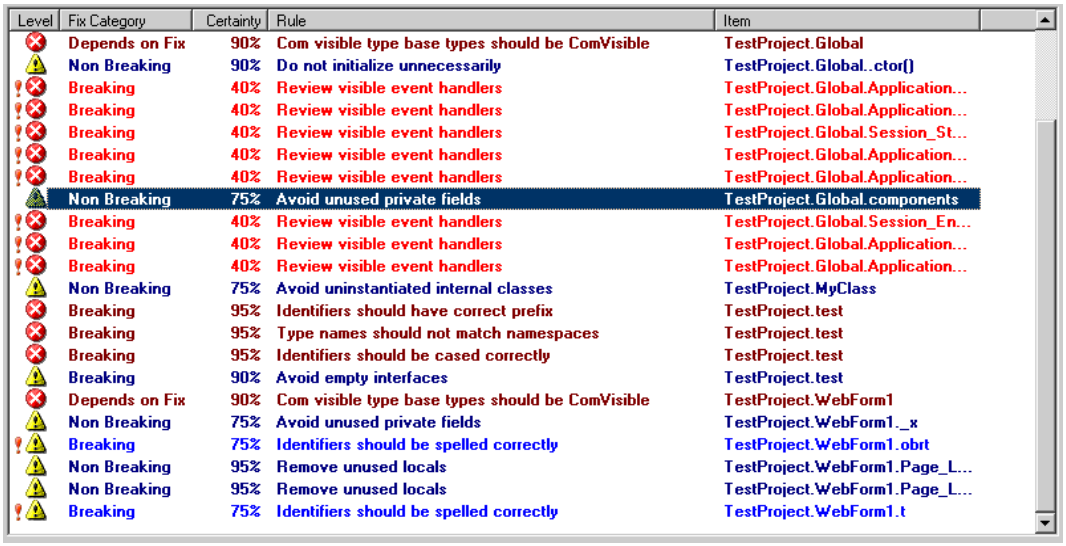
17.5.2 FXCop

FXCop is a free code analysis tool from Microsoft. It analyzes the given assemblies and calculates a set of metrics from it. The results are displayed in a report. Following rules are included in the report:

- Design Rules
- Globalization Rules
- Naming Rules
- Performance Rules

[3] Refer to the Test Reports

- Usage Rules

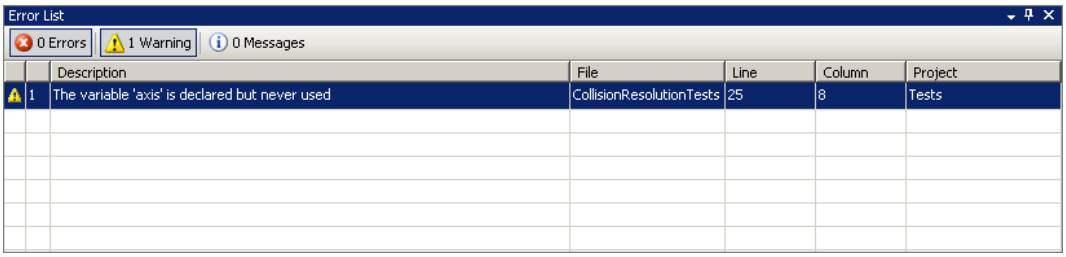


Level	Fix Category	Certainty	Rule	Item
	Depends on Fix	90%	Com visible type base types should be ComVisible	TestProject.Global
	Non Breaking	90%	Do not initialize unnecessarily	TestProject.Global..ctor()
	Breaking	40%	Review visible event handlers	TestProject.Global.Application...
	Breaking	40%	Review visible event handlers	TestProject.Global.Application...
	Breaking	40%	Review visible event handlers	TestProject.Global.Session_St...
	Breaking	40%	Review visible event handlers	TestProject.Global.Application...
	Breaking	40%	Review visible event handlers	TestProject.Global.Application...
	Non Breaking	75%	Avoid unused private fields	TestProject.Global.components
	Breaking	40%	Review visible event handlers	TestProject.Global.Session_En...
	Breaking	40%	Review visible event handlers	TestProject.Global.Application...
	Breaking	40%	Review visible event handlers	TestProject.Global.Application...
	Non Breaking	75%	Avoid uninstantiated internal classes	TestProject.MyClass
	Breaking	95%	Identifiers should have correct prefix	TestProject.test
	Breaking	95%	Type names should not match namespaces	TestProject.test
	Breaking	95%	Identifiers should be cased correctly	TestProject.test
	Breaking	90%	Avoid empty interfaces	TestProject.test
	Depends on Fix	90%	Com visible type base types should be ComVisible	TestProject.WebForm1
	Non Breaking	75%	Avoid unused private fields	TestProject.WebForm1._x
	Breaking	75%	Identifiers should be spelled correctly	TestProject.WebForm1.obrt
	Non Breaking	95%	Remove unused locals	TestProject.WebForm1.Page_L...
	Non Breaking	95%	Remove unused locals	TestProject.WebForm1.Page_L...
	Breaking	75%	Identifiers should be spelled correctly	TestProject.WebForm1.t

Unfortunately, many false positives get found. This makes it impossible to periodically review all warnings carefully. Therefore, in the last Construction phase the developers must review all warnings and decide whether severity and applicability of a warning justifies a change.

17.5.3 CSharp Compiler

Before checking-in, the code must compile successfully and must not cause compiler warnings.



17.6 Coding Guidelines

Standard .NET conventions are used[4][5] for all source code. For the formatting settings a Visual Studio 2008 compatible settings file provided by Tecan Schweiz AG is used.

[4].NET Design Guidelines `http://msdn2.microsoft.com/en-us/library/ms229042.aspx`
[5].NET Naming Guidelines `http://msdn.microsoft.com/en-us/library/ms229045.aspx`

17.7 Guidelines for TEX

For a better and a unique documentation style, the Documentation Guidelines have been defined as follows:

1. Write negation and short forms out, e.g. cannot, does not, it is
2. Write in passive form. Avoid the usage of "we"
3. Write out numeric values in full where applicable, e.g. one, two, three
   (a) This rule is not applicable to mathematical equations
4. When a new term is introduced it should be highlighted in italics only the first time
   (a) Use \emph over \textit (both italic)
5. Acronyms should be introduced as following: "[...]Iterative Deepening Depth-First Search (IDDFS)[...].The IDDFS is used to[...]"
   (a) Define an acronym only if it is used in the following paragraphs
   (b) If an acronym is defined in the abstract it should not be used in the document body. The acronym should be redefined in the document body if the need arises
6. Mark unfinished parts or parts that have to be reviewed in the LaTeX code with %todo (LaTeX comment)
7. Words in a title (sections, subsections, sub subsections, paragraphs) start with a capital letter. This rule does not apply to prepositions (in, at, on, etc.) and to articles (the, a/an)
   (a) Example: *Third Iteration: Finding the Goal Node*
8. Figures should be labeled with a foregoing "img:"
   (a) Example: \label{**img:**a_beautiful_picture}
9. Sections should be labeled with a foregoing "sec:"
   (a) Example: \label{**sec:**a_simple_section}
10. For referring to figures use *see* or *refer to* in the text, followed by the reference to the graphic and surrounded by brackets
    (a) Example: "[...]a sample tree (see Figure 1.2.3.4).[...]"
11. For referring to sections within subsections (in LaTeX marked as \subsubsection) use the following constellation: "As mentioned in \textit{ABC} on page \pageref{section:the_section}"

Chapter 18

Minutes and Meetings

In the following sections the Trac Wiki pages concerning project management are listed.

Organization

- Upcoming
- Minutes
- People
- Aufgabenstellung

Guidelines

- Documentation Guideline

ToDo's

The list is ordered ascendant according to the priority of each task

- A0 poster
- ~~Update screenshot of the Spline Tool (make screenshot of the Spline Tool with the Large Sample Spline)~~
- Start Management Summary (can be done in German I think)
- ~~Improve "Kurze Arbeitsbeschreibung"~~
- ~~Consider different speed factors when splitting movement into different axes, make spline examples (StJu)~~
- Review Tasks in Minutes Wiki if anything is still open
- Describe Voronoi and Potential Field in algo analysis. Add short reference to collision.tex introduction (GiAc)
- Test Report
  - Format test results using colors
  - ~~Add test dates~~
- Applying splines to a three dimensional environment
- Improve Parser
- Add references to target environment in algorithm analysis
- Technical Report
  - ~~Merge Cubic Spline chapter (collision_cubicsplines.tex) with the Collision section (collision.tex)~~
  - Add link to paper describing the Fringe Search
  - ~~Add IDDFS (Iterative Deepening Depth First Search) and FS (Fringe Search) to acronym list~~

Previous Meetings

Friday, 5th June

- Attendees: StJu, GiAc, JoLe, RaKe
- Location: Tecan, Männedorf
- Time: ?
- Purpose
  - Set up development environment
  - Discuss project state with JoLe

Thursday, 4th June

- Attendees: StJu, GiAc, HaHu
- Location: HSR
- Time: 13:00
- Purpose
  - Discuss technical report

Tuesday, 26th May

- Attendees: StJu, GiAc, JoLe
- Location: HSR
- Time: 13:30
- Purpose
  - Discuss project state
  - Present Round Moves Info

Tuesday, 26th May

- Attendees: StJu, GiAc, HaHu
- Location: HSR
- Time: 10:15
- Purpose
  - Discuss project state
  - Round Moves Info
  - Review technical report
  - Prepare Thesis description for cfurrer.

Tuesday, 12th May

- Attendees: StJu, GiAc, JoLe
- Location: Tecan, Männedorf
- Time: 13:30
- Purpose
  - Discuss project state
  - Round Moves Info
  - Review project management documents

Tuesday, 26th May

- Attendees: StJu, GiAc, JoLe, HaHu
- Location: HSR
- Time: 10:15
- Purpose
  - Discuss project state and deliverables
  - Discuss project management document review

Thursday, 7th May

- Attendees: StJu, GiAc, HaHu
- Location: HSR, 6.010
- Time: 16:30
- Purpose
  - Discuss project state and deliverables
  - Discuss document review

Tuesday, 28th April

- Attendees: StJu, GiAc, JoLe
- Location: Tecan
- Time: 13:30
- Purpose
  - Discuss project state and deliverables
  - Late updates of device movements
  - Demonstrate waypoint generation

Thursday, 23rd April

- Attendees: StJu, GiAc, HaHu
- Location: HSR
- Time: 16:00
- Purpose
  - Discuss project state and deliverables
  - Demonstrate waypoint generation

Monday, 20th April

- Attendees: GiAc, Louis-Sepp Willimann
- Location: HSR
- Time: 17:15
- Purpose
  - Discuss possibilities to construct a curve that goes through defined control points (coordinates)

Wednesday, 8th April

- Attendees: StJu, GiAc, JoLe, HaHu
- Location: HSR
- Time: 16:00
- Purpose
  - Discuss project state and deliverables
- Questions
  - How's the state of the documents?
  - Testing

Thursday, 19th March

- Attendees: StJu, GiAc, JoLe
- Location: Tecan
- Time: 17:30
- Purpose
  - Discuss project state and deliverables, Requirements
- Questions
  - NDA?
  - Software Architecture Document in own document?

Thursday, 19th March

- Attendees: StJu, GiAc, HaHu
- Location: HSR, 6.010
- Time: 16:00
- Purpose
  - Discuss project state and deliverables
- Questions
  - Software Architecture Document in own document?

Thursday, 5th March

- Attendees: StJu, GiAc, HaHu
- Location: HSR, 6.010
- Time: 16:00
- Purpose
  - Discuss project schedule, milestones and deliverables
- Notes
  - Send in management docs and other technical documents until Monday, the 2nd of March

Thursday, 4th March

- Attendees: JoLe, StJu
- Location: Tecan, Männedorf
- Time: 13:00
- Purpose
  - Review of management documents after inception phase

Thursday, 19th February

- Attendees: StJu, GiAa, HaHu, JoLe
- Location: Tecan, Männedorf
- Time: 16:15
- Purpose
  - Kickoff Meeting

Minutes

04.06.09, Construction 3 - 3

Attendees

StJu, GiAc, HaHu

Location

HSR, 6.010

Decisions

- Short description has to be updated
- Introduction section has to be updated
- Management Summary should be included in the documentation with an own chapter and therefore should be written in English
- The documentation should be reordered: Management Summary, Technical Report, Project Management, Testing, Appendix

Tasks

- Update short description of the Bachelor Thesis
- Expand the Introduction chapter with a description regarding the non-deterministic property of the Liquid Handling Platform
- Reorder the documentation
- Include Management Summary

26.05.09, Construction 3 - 2

Attendees

StJu, GiAc, JoLe

Location

Tecan, Männedorf

Decisions

- Next meeting will be held on Friday the 5th of June at Tecan Männedorf. The date is not known yet.
- Short abstract like implemented now is preferred

Tasks

- Improve Cubic Spline description
  - Add axis splitting
  - Add examples
- Do not mention hardware related details in Technical Report

26.05.09, Construction 3 - 1

Attendees

StJu, GiAc, HaHu

Location

HSR, 6.010

Decisions

- Next meeting will be held on 4th of June, 11:00 at HSR to review the Technical Report

Tasks

- Send in Technical Report until 1st of June
- Send in Abstract until 27th of May
- Restructure Technical Report
  - Extend introduction
  - Rate algorithms
  - Add system overview, components

12.05.09, Construction 2 - 1

Attendees

StJu, GiAc, JoLe

Location

Tecan, Männedorf

Decisions

- Smooth Algorithm must be exchangeable
- Next meeting will be held on the 26th Mai, 13:30 at Tecan
- In the last week development will take place at Tecan. On Friday the 5th June the infrastructure will be set up

Tasks

- Compare Spline to Sinus-Smoothing: Acceleration per axis (Mathematica)
- Describe how the speed is calculated during a move
- Use fastest route, not shortest one
- Use the same scene for compare sinus to splines
- Handle maximum speeds

08.05.09, Construction 1 - 2

Attendees

StJu, GiAc, HaHu

Location

HSR, 6.010

Decisions

- State of the project management documents is okay
- Add Personal Report (can be written in German or in English) and Wiki Pages to Project management docs
- Add Timereport to project management documents
- Next meeting will be held on the 26th Mai, 10:15 at HSR

28.04.09, Construction 1 - 1

Attendees

StJu, GiAc, JoLe

Location

Tecan, Männedorf

Decisions

- Project management docs should be sent in until Friday
- Gather more information about round moves using Cubic Spline Interpolation or Sinus/Cosinus curves on selected edges -¿ Create a feasibility document with the results of the research
- Next meeting will be held on the 12th Mai, 13:30 at Tecan

23.04.09, Elaboration 4

Attendees

StJu, GiAc, HaHu

Location

HSR , 6.010

Decisions

- Docs should be sent in in the next iteration for a review
- Next meeting will be held on the 7th Mai, 16:30 at HSR

20.04.09, Meeting Regarding Mathematical Question

Attendees

GiAc, Louis-Sepp Willimann

Location

HSR

Decisions

- Investigate in the usage of Splines (Cubic Splines)

08.04.09, Elaboration 3 - 1

Attendees

StJu, GiAc, JoLe, HaHu

Location

HSR, 6.010

Decisions

- The split in TestReport and TestCases is ok
- Bounding boxes are sufficient for collision calculation
- Low computation time is more important than low memory usage

Tasks

- Evaluate possibilities of late movement updates
- Update waypoint generation and viewer so the mechanism can be demonstrated

19.03.09, Elaboration 2 - 2

Attendees

StJu, GiAc, JoLe

Location

Tecan, Männedorf

Decisions

- Don't list snapshots on a per-revision basis in wiki
- Bounding boxes are sufficient for collision calculation
- Low computation time is more important than low memory usage

Tasks

- Use brighter colors in composition device examples
- Use different font for code sections
- Create diagram showing which algorithm is used where

19.03.09, Elaboration 2 - 1

Attendees

StJu, GiAc, HaHu

Location

HSR, 6.010

Decisions

- The Software Architecture Document does not have to be placed in its own document for now

Tasks

- Create a conceptual model about the interaction of the 3D move engine with the target system
- Create a document model
- Add new work packages for the two tasks defined above
- Name terminals in grammar
- Organize meeting with JoLe before Eastern

05.03.09, Elaboration 1 - 1

Attendees

HaHu, GiAc, StJu

Location

HSR, 6.010

Decisions

- Next meeting will be held at HSR, 6.010 on the 19th of March, 16:00

Tasks

- Add "Deliverables" subsection to "Work Packages" in Software Development Plan
- Add "Work Packages" for every iteration and fill them as much as possible
- Update Risk Management Document: Describe fallbacks for risks and not just actions to eliminate them

04.03.09, Inception 1 - 1

Attendees

JoLe, StJu

Location

Tecan, Männedorf

Decisions

- Upload snapshots of management documents after every iteration

19.02.09, Kick Off

Attendees

StJu, GiAa, HaHu, JoLe, RaKe

Location

Tecan, Männedorf

Decisions

Tasks

- Define project schedule, milestones and deliverables until next meeting (StJu,GiAc)
- Set up svn accounts for JoLe and HaHu (StJu)

Part V

Personal Reports

Giuseppe Accaputo

Seventeen weeks have passed since the beginning of this adventure, and now I am writing my personal report, describing how the project went and the vast amount of new things I have learned so far.

First of all, I have to say that it was an amazing time and experience to work on the Bachelor Thesis, consisting of such an interesting mix of topics (software engineering, computer science, math and more)x. I also liked the fact that the Bachelor Thesis required quite a lot of research and evaluation, during which I have learned a lot of new things. But more about this later.

Project Management We used the RUP to manage our project and software development, splitting the available time into appropriate iterations and lifecycle phases and tailoring everything to our needs. We knew *ab initio* that we would use RUP, since we knew we could benefit a lot from its iterative approach. Also, we decided to work with RUP because we used it successfully in two preceding projects (*Software Engineering 2* project and the Seminar Paper), knowing exactly how we were going to use it in regard to our Bachelor Thesis.

Since the beginning of the project we knew about the necessity to invest a lot of time in research, resulting in an extended Elaboration phase. It turned out it was a good move we had made, since we had used the planned time successfully to evaluate various algorithms and to choose the most suitable ones. Then, during the Construction phase we could save a lot of time in the implementation of the algorithms thanks to a detailed evaluation. As adapted already during the Seminar Paper, we decided to write Iteration Assessments during the Bachelor Thesis, too. The Iteration Assessments helped a lot, since after each iteration we had a detailed overview of all the completed tasks of the actual iteration and the planned tasks for the next iteration. Writing Iteration Assessments has been proven once again to be very useful and it is definetly something I will use in future projects.

Learning Benefits As already mentioned in the first paragraphs of the Personal Report, I have learned a lot of new things during the Bachelor Thesis. I have to say that the most things I have learned were during the research part, since it required reading through research papers and other material, and required a lot of thinking. I have to admit that I really enjoyed the research part of the Bachelor Thesis, probably because I liked the fact that I were going to learn a lot of new things.

Thanks First of all, I would like to thank Prof. Dr. Hansjörg Huser, Joas Leeman and Rainer Kerkmann for the kind help, the dedicated time for meetings, reviews and more, and for the availability.

I also want to thank Tecan Schweiz AG for the provided workspace in the office, which was very useful and I somehow regret that we did not ask for a workspace in an earlier stage of the project.

I would also like to thank my team mate Stephan for his amazing team work and commitment. Since we have both worked together on the Seminar Paper in the last semester, working together on the Bachelor Thesis was the logical corollary - and it worked very well.

In conclusion I want to say that it was a very interesting, exciting and - above all - an instructive period of time, the Bachelor Thesis. I can say that of all the lectures I visited and all the practices I made during my study at the HSR, the Bachelor Thesis is the one I most benefitted from. So, thanks for this great opportunity

Stephan Jud

Soon the Bachelor Thesis will be finished. After one semester of intensive work, a lot has been achieved. The final days were quite busy during which my team mate and I have been reviewing documents and source code.

In retrospect this was the first demanding project in school for me – Also after a few weeks it was still not evident how we could achieve the goals. However, we knew from the beginning that it will not be easy but choosing a more theoretical project turned out to be way more fun than the previous ones we had.

The mix of different topics like software engineering, math, hardware issues and computer science was a truly interesting experience. The limitations of the target environment discarded many of our approaches and stayed a challenge during the whole project. In the end however, it was one of the main factors why it stayed an interesting project throughout its whole duration. I really enjoyed the thesis although we had to work hard to get a basic prototype working. We also learned to appreciate simple solutions over complicated ones and tried hard to reduce the implementation to its essentials. Keeping project management documents up to date took a lot o time but helped us in the end to better inform stakeholders before meetings and to control the progress of the project on our own.

Using LaTeXenabled us to easily check changes made by the other member which led to an improved quality of the documentation. It turned out to be a superior to other tools especially as it allowed us to use our own scripts and familiar tools. The Trac environment has turned out to be an effective and appropriate management tool for our project.

Studying related papers turned out to be an exciting activity. In the end we could not use many ideas or approaches. However, it gave us a great overview on related topics which often did not affect us directly, but was interesting to read about anyway. Throughout the whole project we could decide on our own in which way we would like to move on. I would like to take this opportunity to thank for the trust we received.

Special thanks go to my team mate Giuseppe. Without his contributions and patience the project would not have been possible.

Part VI

Appendix

Appendix A

Images

A.1 Path Finding

A.1.1 Logical Map Traversal Example

This section shows how a logical map (section 2.2.1) is traversed. Just one evaluation branch is shown though. Different branches can be evaluated iteratively as long as the costs used up to a particular point are maintained correctly and used for proper synchronization. Once it has been detected that several branches end up in the same node, the one with the least costs used is tracked and the others are dropped.

Figure A.1: The current position and the desired target positions are known and placed as nodes into the logical map. Logically they are placed next to each other because it unknown yet how many nodes lie between them.

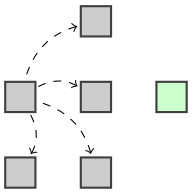


Figure A.2: After querying the underlying components a first set of graphs can be appended to the start node.

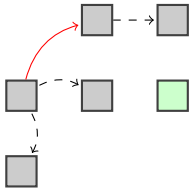


Figure A.3: The algorithm traverses through the graph carrying the costs with it. Note: Depending on the costs which have been accumulated (which represents the time consumed until a node) nodes are added and removed and costs of graph are updated.

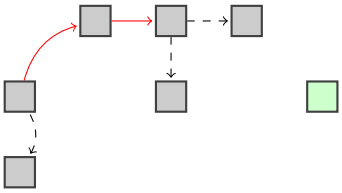


Figure A.4: Collision Avoidance components are queried to get new graphs for the currently active node. The accumulated costs are given to those components so they can generate a snapshot of the environment according to a particular time.

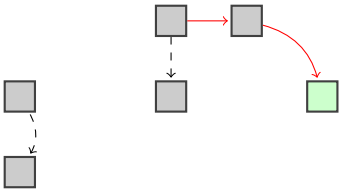


Figure A.5: The map is traversed until the target node is found, a loop is detected or no other paths are left to try out. Note: It is possible that a graph gets removed the current node depended on once the traversal advances. This is not something bad and can be ignored because the moving object already passed that graph and is not affected with changes “behind” it.

A.2 Collision Avoidance

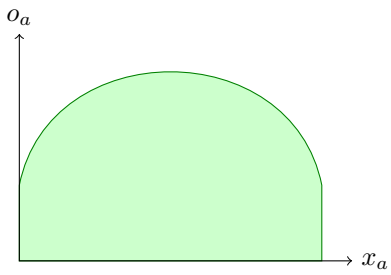


Figure A.6: It is possible that the represented object is turning which results in a parabolic axis allocation while the object travels within its axis range. This behavior can be represented with quadratic functions

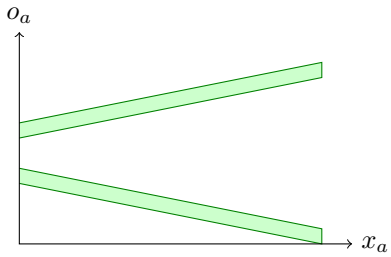


Figure A.7: It is possible that one axis is bound to several objects which do move away or towards each other when travelling within the axis' range. To model this, every object has to be associated with its own function

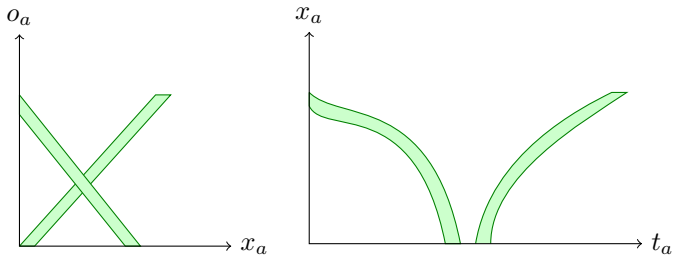


Figure A.8: There may be cases where a collision would be detected when applying a function graph. If both of the colliding objects are active their positions must be evaluated relative to execution time as it might happen that they move out of their way just in the right moment.

A.3 Collision Resolution

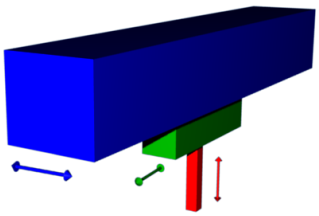


Figure A.9: The main arm (blue) can move within x range. Its green subdevice can move within y range. The red subdevice can move in z range. Obviously, all subdevices are affected when the main arm moves which means that the collision avoidance mechanism must consider all underlying subdevices when moving a device.

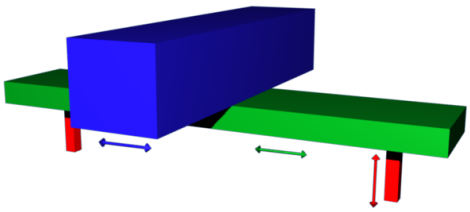


Figure A.10: The main arm (blue) can move within x range. Its green subdevice can magnify the x movement. The two red subdevices can move in z axis. Solving a collision on the x range can be solved in three ways: Move the main arm, move the green arm or move both of them. Ideally, both arms are moved half of the required distance (assuming they have the same acceleration factor), doubling the speed at which a device is able to evade a collision.

List of Figures

List of Algorithms

Acronyms

A* A* search algorithm. 57, 63, 64, 66, 67

DSL Domain Specific Language. 43

FS Fringe Search. 65, 67

IDA* Iterative deepening A*. 22, 63–68

IDDFS Iterative deepening depth-first search. 63–65, 68

ME-IDA* Memory-enhanced IDA*. 63

RUP Rational Unified Process. 13, 14

Glossary

Axis Allocation
Describes how much space a device occupies on an axis. 25

Collision Avoidance
Term which is used to describe the process of detecting and resolving collision between devices and obstacles or other devices. 21

Collision Detection
The term Collision Detection is used to describe the process of determining when a roboter collides with another roboter or obstacle. 25

Collision Resolution
The term Collision Resolution is used to describe the process of calculating an evasion point in order to avoid a collision. 29

Dimensional Properties
Term used to describe the current dimensional situation of a roboter in x, y and z axis. 24

Monotonic
A property of a mathematical function which preserves its order. It may be either increasing or decreasing.. 28

Robotic Arm
A roboter for the liquid handling platform. Depending on their purpose there are different capabilities how the roboter can interact with its environment. 25

Sweep Prune
Name of an algorithm which allows efficient collision detection between objects. This is achieved by reducing the number of checks by keeping a sorted list of obstacles. 25

Track
Robotic arms do not move within the environment by making use of wheels. Instead, they are mounted on rails and can move within its range. 22

Waypoint
State and position of a roboter at a given moment. 23

Bibliography

[1] A* Search Algorithm, [theory.stanford.edu]

[2] A* Pathfinding, [policyalmanac.org]

[3] Informed Search, [cs.umd.edu]

[4] Dijkstra's Algorithm, [http://www.cse.ust.hk]

[5] Yngvi Bjoernsson, Markus Enzenberger, Robert C. Holte and Jonathan Schaeffer, "Fringe Search: Beating A* at Pathfinding on Game Maps", [ualberta.ca]

[6] Sweep and prune algorithm, http://en.wikipedia.org/wiki/Sweep_and_prune

[7] R.G. Lavender, D.C. Schmidt, "Active Object: An Object Behavioral Pattern for Concurrent Programming", http://www.cs.wustl.edu/~schmidt/PDF/Act-Obj.pdf

[8] Design Patterns: Elements of Reusable Object-Oriented Software