A Language for Reconfigurable Robot Control.

Bart Nabbe

September 1998
CMU-RI-TR-98-32

Submitted to the Department of Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science

at the

University of Amsterdam

Author .....
Department of Computer Science
November 24, 1998

Certified by .....
prof. Dr. ir. Frans C.A. Groen
Professor (UvA)
Thesis Supervisor

Certified by .....
Dr. Martial Hebert
Senior Research Scientists (CMU)
Thesis Supervisor

Accepted by .....
prof. Dr. Peter van EmdeBoas
Chairman, Department of Computer Science

A Language for Reconfigurable Robot Control.

by

Bart Nabbe

Submitted to the Department of Computer Science
on November 24, 1998, in partial fulfillment of the
requirements for the degree of
Master of Science

Abstract

When robots are to be deployed in urban environments, these robots need to be able to cope with all kinds of terrain. Includes curbs, steps and even stairs. A reconfigurable robot is best used in these cases.

This thesis presents a framework that deals with configuration (posture) selection for this type of robotic systems. The system consists of a compiler and an execution engine, which provides the coupling between the low level control software and the high level planning software. The Configuration Selection Module (CSM) is generated from a specification that consists of a priority list of observation, action tuples for all possible goals. The planner stacks its desired goals on the CSM execution stack. This stack is reduced according the methods given in the specification.

The system has been tested on a regular mobile robot and on a reconfigurable robot simulator. The result from these experiments suggest that the system will be able to function as desired.

Thesis Supervisor: prof. Dr. ir. Frans C.A. Groen
Title: Professor (UvA)

Thesis Supervisor: Dr. Martial Hebert
Title: Senior Research Scientists (CMU)

Acknowledgments

I would like to thank Frans Groen for making the initial contact with CMU, his support and the freedom to shape my research. At the CMU side I would like to thank Chuck Thorpe for taking on yet another student. Most of all I would like to thank Martial Hebert for his ideas, support (both intellectual and financially) and his work on the Pandora simulator.

Contents

List of Figures

Chapter 1

Introduction

Reconfigurable mobile robotic systems, such as walkers or other articulated locomotion systems, are more difficult to control than the regular wheeled or tracked systems. Because of this complexity, numerous of these robots have been used for testing adaptive and learning algorithms. Although the results of this ongoing work is interesting, for simple articulated locomotion systems a careful analysis of the task and the locomotion system can be used to generate a control algorithm.

When we would like to control such a locomotion system, it would be of interest to find out how the perception of the environment could be used to choose between configurations. An other key issue is to determine if such a relation could be described in a formal way so that a specification of these relations could be used to automate the generation of a control algorithm.

1.1 Background

The application of robots in urban environments for inspection, search and rescue or other missions present an challenging problem in robot design. Such a robotic system must be able to traverse all kinds of terrain, curbs, stairs and other obstacles. The CMU Robotics Institute started research in this area when they began investigating the usage of robotic systems as scouts or pointmen. This type of robot could then assist in such missions. A typical setting for such an application is depicted in figure 1-1.

Figure 1-1: A typical simulated urban environment.

1.2 Mission and Operational Requirements

Several mission scenarios were taken into account while the design requirements were laid out. The following list outlines these scenarios.

- **Transport/Staging**

  The collapsed vehicle system is transported to the outskirts of a town in the transport vehicle assigned to the team, typically, a three-man team. The robot will already have been preloaded with batteries and local area and city maps. The system is carried from the transport vehicle to a safe staging area by the team. There it is put down and activated. The Differential Global Positioning System (DGPS)-coded position of the team(s) and the vehicle are displayed on the operator's wearable graphical user interface. The operator then decides on how to best plan a short recon mission for the area, whether by selecting a goal point from an image or a map and allowing the robot to plan its route, or by entering a more detailed sequence of waypoints manually.

- **Area Recon**

  The robot is commanded, either by entering end-/waypoints or through teleoperation, to traverse a few streets to determine what potential hazards and hiding places might exist. The robot drives at fairly high speeds through the open-access areas with its spherical camera system fully deployed to view the surrounding areas. During these maneuvers, safeguarding sensors provide sufficient information for on-board computers to drive the robot in cluttered terrain.

  Should the vehicle be able to drive around the obstacle, the on-board vision system will generate a set of steering, speed and distance commands to the vehicle-controller

CPU for it to execute and monitor. Should the vehicle need to straddle the obstacle, the system commands itself to get into the necessary straddling configuration, before proceeding with the driving maneuvers. In the case that the vehicle needs to climb over an obstacle, such as a cinder block, wall or curb, it will automatically perform any approach and coordination maneuvers to autonomously accomplish the maneuver.

- **Spot-Recon**

  The vehicle can be sent to perform individual recon operations in suspect areas such as alleys, sewers, or buildings. In a designated suspect area, the operator can command the vehicle to proceed into and through the area to gather intelligence data (audio/video and motion).

- **Surveillance**

  The system can also be used for surveillance, by commanding it to drive to a specific vantage point from which it can monitor certain areas for suspect operations. The system can either use live video transmission to the operator or on-board motion-detection software, and can scan video feeds from either the forward-looking or omnispherical camera systems.

- **Rear-Guard**

  The robot can also be used as a leave-behind post-guard, continuously monitoring activities in a rearward area.

- **Redeployment**

  The robot is intended to be reused; thus the system will be picked up by the designated operating-soldier and driven back to the staging area or to the transport vehicle for storage and transport to the next deployment zone.

- **Special Missions**

  Typical special missions that the robot system could be used for are listed below: The robot can be used as a mothership system, by carrying several smaller static or semi-mobile sensing systems with it, which it drops at key locations. These individual sensing systems can be used to monitor key areas with audio or video sensors. If these sensors are able to achieve some small level of locomotion, they can relay intelligence back.

Such a mission scenario shows clearly that robotic system might be exposed to a very hostile environment. It is also clear that it must be fairly autonomous so that the operator is not tied up by operating the system. The key design requirements based on the presented scenario is given below:

- Fast locomotion on flat ground
- Negotiate obstacles
- Negotiate stairs
- Robust to falls
- Man-portable
- Power autonomy sufficient to carry out mission
- Obstacle-strewn, man-made cluttered environment
- Ground-level and elevated danger zones
- Observation and inspection
- Multi-unit coordinated human/machine operations

1.3 The Pandora System

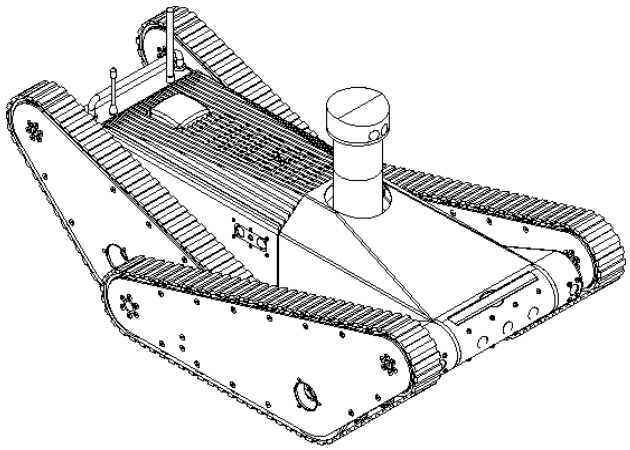


Figure 1-2: Pandora system - CAD rendering

A robotic system called "Pandora" (Figure 1-2) was designed to meet these criteria. The Pandora system is equipped with articulated treads, which can be configured in various

postures depending on the requested terrainability. This allows the system to negotiate a broad range of obstacles. On flat terrain, the number of contact points can be reduced by rotating the threads thereby increasing the systems mobility. Its extensive sensor suite consists of a stereo pair of forward looking cameras, narrow infrared bandpass filtered camera, an omnidirectional camera and a multiplexed sonar obstacle detection ring. The narrow infrared bandpass filtered camera is used in conjunction with a set of two high power laser light stripers on top of the OmniCamera to image the environment and to detect and measure size and location of obstacles. The OmniCamera is a vertically-deployable upside-down CCD camera imaging into a parabolically shaped mirror to view a 360 degree field-of-view. The on-board navigation sensors consist of a differential GPS receiver, an integral solid state compass and pitch and roll inclinometer, a solid state gyro and safety limit switches on vehicle pitch/roll and locomotor posture.

1.4 Problem description

The reconfigurability of Pandora's locomotion system as described in the previous section allows the system to adapt to current terrain conditions. It can independently position its front and rear pairs of threads in any position. Given this information, configurations were classified for different kinds of terrain, obstacles and exposure (risk level).

Figure 1-3 displays some of these possible configurations. The Pandora design document [RI98] gives an in depth description of this classification.

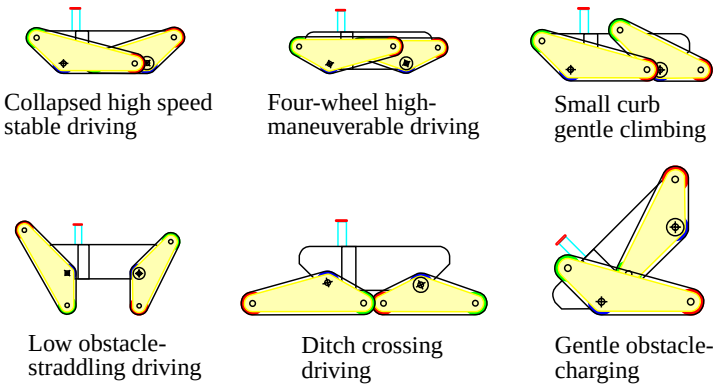


Figure 1-3: Some possible configurations of the Pandora system

The selection of a configuration depends thus on the type of terrain and on the driving

parameters, such as the allowed risk level. For example if a high risk level is allowed, Pandora would charge an obstacle as fast as possible. On the other hand, a low risk level will make Pandora observe the environment more closely so that it will not be taken by surprise if any unexpected terrain feature appears. A simple forward command could thus involve a whole sequence of reconfigurations in order to traverse the requested distance. Typically the robot would start to move in the requested direction until the sensor system suspects an obstacle. The system could then reconfigure itself into an observation posture so that the laser light striper is swept across the obstacle in order to acquire enough data to confirm the obstacle class. Based on this information the robot selects the best configuration to attack and negotiate the obstacle.

The work described in this document will describe the configuration selection decision process and a mechanism to generate a system to do this.

1.5 Thesis overview

A road-map of the Pandora software architecture is given in Chapter 2. This chapter describes the functions of the several modules within the system. It also explains in more detail where the configuration selection decision is made and on what data this decision is based. Chapter 3 gives an overview of related software architectures. It points out the key features and explains the relation to the problem at hand. In Chapter 4 the actual language and its execution engine will be presented. Chapter 5 gives the results of the experiments performed with the Nomadic Scout mobile robot and the Pandora simulator. Finally, Chapter 6 suggests areas of future work, and Chapter 7 draws conclusions about the work discussed herein.

Chapter 2

The Pandora Software Architecture

The Pandora Software Architecture interfaces to the user by means of the communications front-end on one side, on the other side it interfaces to the vehicle front-end which is tied to the robot hardware. All operator interaction goes through the communications front-end. A single driving mode is in control of the robot, this module uses data from the sensor front-end to plan motion commands for the system. These commands are merged together with the allowed directions provided by the safeguarding module. This combined command is fed into the configuration selection module which might generate a sequence of reconfigurations in order to pursue the desired direction of travel. The commands generated by the configuration selection module are executed by the robot hardware. The driving modes component determines the mission capabilities and is modular, therefore new modes with different capabilities can be added easily. The safeguarding module has also a modular structure, this allows an easy sensor upgrade.

Figure 2-1 shows how these separate modules form the complete software architecture. The remaining sections give a brief description of all these modules.

2.1 Communication Front-End

The data provided by or presented to the operator is routed from the user interface to the communications front end. Therefore, the communication front-end is responsible for receiving commands and parameters from the user and sending images and status infor-

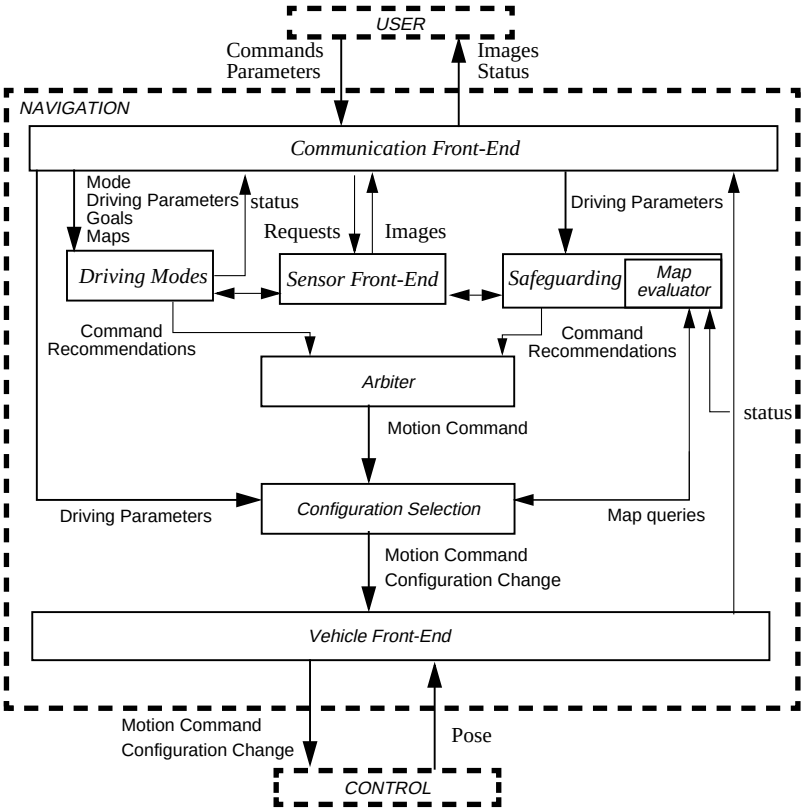


Figure 2-1: The software architecture of the Pandora system

mation. The main commands sent by the user include change of driving mode, desired maximum speed, and requests for images. In addition, the user can send a risk level which guides the on-board navigation system in selecting the appropriate configurations for the robot. Data sent back to the user by the front-end include images, when requested, status information and vehicle poses.

2.2 Driving Modes

The driving modes component provides the planning level in the system. It incorporates currently three semi-autonomous driving modes. The operator can only activate a single driving mode at a time. These driving modes generate a recommendation for both heading and speed. Currently the following driving modes are provided:

Visual Servoing The operator designates a goal region anywhere in the image and the robot drives toward that region using on-board visual servoing. This mode requires minimal operator input and does not require accurate position estimation or map information.

Waypoint Teleoperation The operator designates intermediate points in the image. Those points are used by the on-board navigation system to drive the vehicle. This mode requires more work on the part of the operator but provides better control of the route actually followed by the robot.

Map-based Planning The operator designates a goal in a map of the environment. The robot drives toward the goal by continuously re-planning its route based on sensor input. This mode is the closest to a fully autonomous navigation mode but it requires map and position information which is not always available.

2.3 Sensor Front-End

The sensor front-end controls which of the sensors are active at a given time, and makes the sensor data available to the other components. Sensor data is stored in a global data repository shared by all the software components. In particular, the sensor front-end does not need to know which of the other modules is using the data. Again, this approach allows for flexibility since new modules can be added without modification of the sensor front-end.

2.4 Safeguarding

A safeguarding component collects the data generated by the safeguarding sensors, integrates it in an internal local map, and analyzes it to evaluate the traversability of the terrain surrounding the robot. The local map is periodically evaluated in several candidate directions of travel, to generate measures of height and slopes of potential obstacles in those directions. For a given heading, the distribution of height and slope is compared with the limits of the current robot configuration, in order to determine the traversability of the terrain. This evaluation is the basis for the input to the command arbitration component and the configuration selection module. The local map scrolls along with the robots movement so that the robot remains the center.

2.5 Vehicle Front-End

The commands are sent to a vehicle front-end which is the interface to the robot hardware. The vehicle front-end verifies the commands for validity before sending them to the robot. The vehicle front-end is also responsible for receiving and integrating the pose estimates - position and orientation - from the robot hardware. The robot hardware aborts a command if it exceeds the limits of the system.

2.6 Command Arbitration

Pandora uses a command arbitration scheme for merging the command recommendations of the driving modes and the safeguarding system. In this approach, the scores for the different headings are combined into a single motion command. Individual components, such as a driving mode and a safeguarding component, generate preferences for the next planned heading of the robot. Those preferences are represented by a set of scores for a discrete set of angles. A zero score indicates that a heading is either blocked, or that insufficient sensor data is available to evaluate traversability in this direction, or that the current configuration prevents the robot from driving in this direction. The recommendations are combined into a single set of recommendations. The combination is performed using essentially a linear combination, with the added provision that headings with zero score - "vetoed" headings - always remain vetoed after combination. The weights used in the combination reflect the relative importance of the components being combined. The last step of the command generation is the selection of the optimal command from the combined recommendations. The command with the highest score is selected and send to the configuration selection module.

2.7 Configuration Selection

Based on the motion command received from the arbiter, the configuration selection module has to select the best configuration for Pandora. This decision is based on several parameters such as:

- **Map-evaluation**

  Most important in the configuration selection decision is the observed obstacle or

terrain. If for example flat terrain is observed, the system could change to the four point high speed driving configuration to exploit this feature. The evaluation of the map could also lead to an observation posture, this would occur if there is not enough data available for the evaluator to determine the obstacle type. The robot would then configure to an observation pose which would yield more data, so that the evaluator could determine the obstacle type.

- **Risk Level**

  In general, different configurations could be used to negotiate the same type of obstacle. Therefore, a division is made into low-risk configurations, which are safe but require more component motion and have slower execution, and higher-risk configurations, which are faster but may induce more disturbances - shocks and vibrations. The allowed "risk level", which is supplied by the user, is used to choose between these sets. In missions in which speed of execution is critical, the user would select a high risk mode.

- **Current configuration**

  In order to reduce the reconfiguration time, the current configuration is taken into consideration when selecting a new configuration. Nevertheless it is not necessary to generate the most optimal configuration sequence. The system is allowed to use an opportunistic approach towards selecting a configuration. This is allowed because the higher level software will provide the configuration selection module with the most promising direction of travel.

2.8 Summary

The software integrates the functions of supervised autonomy, safeguarding, and command generation to the robot hardware. The architecture allows easy replacement or upgrade of its components. The software architecture is divided into: A communication front-end, which receives goals and mission parameters from the user. A vehicle front-end which receives positions from the configuration selection module and sends motion commands to the robot hardware. A set of navigation modules. The navigation modules include modules that implement the three driving modes. A safeguarding module, which uses the output of

the safeguarding sensors to maintain a local map around the vehicle. And a map evaluation module which evaluates the map for navigability, based on Pandora's current situation.

Chapter 3

Related Work

The Configuration Selection Module (CSM) receives a motion command (goal)from the arbiter, depending on the map evaluation and possible driving parameters, the CSM decomposes this goal or selects a configuration (action). This problem has a strong reactive character. Likewise a study of existing reactive robot control languages and architectures was carried out. The following section describes these systems.

3.1 SubSumption and SSS architecture

Brooks's SubSumption architecture [Bro86], [Bro91] consists of simple networks of small finite state machines joined by "wires" which connects output ports to input ports. By overriding the value of one wire with a value from another wire, these basic building blocks can be put together to produce complex control mechanisms. The SubSumption architecture advocates that a controller is build by stacking more complex behaviors on top of more trivial behaviours and inhibiting (subsume) there behavior. However, there is no architectural mechanism to support this. During compilation of these behaviours, the perceived parallelism is reduced to a single execution thread.

The Servo, Subsumption, Symbolic (SSS) architecture [Con92], [Con89] is a hybrid architecture for mobile robots that integrates the three independent layers of servo control, SubSumption behavior and a symbolic layer. These separate layers have an increasing latency time on which they react on real-time external input.

3.2 DAMN

A Distributed Architecture for Mobile Navigation (DAMN) is presented in [RP89], [Pay86], This architecture has four levels: mission planning, map-based planning, local planning, and reflexive planning. All levels operate in parallel. Higher levels are charged with tasks requiring much computation and allow high latency times. The lower levels operate on tasks which require a small latency time and are computational inexpensive. The reflexive planning is designed to consist of pairs of <virtualsensor>, <reflexivebehavior>. Each reflexive behavior has an associated priority, and a central blackboard style manager arbitrates among the reflex behaviors.

3.3 REX, GAPP and RULER

[Kae87], [Kae88] and [RK95] describes a framework consisting REX, GAPP and RULER which, given a task and a descriptions of the world construct a reactive control mechanism known as situated automata. Their architecture consists of perception and action components. The robot's sensory input and its feedback are inputs to the perception component. The action component computes actions that suit the perceptual situation. This mapping is always correct, but is not guaranteed to be complete, that is, no output might be generated. This framework is mainly intended to produce circuits that operate in real-time, and some properties of their operation are provable.

3.4 RAP

The Reactive Action Packages (RAP) system as presented in [Fir89] proposes a plan and task representation based on program-like reactive action packages, or RAPs. A plan consists of RAP-defined goals, or tasks, at a variety of different levels of abstraction. The RAP system executes the tasks in its task-list depending on the satisfaction of the associated constraints. In this manner, different methods will be used in different situations. These constraints can also be used to monitor execution, thereby eliminating the need for separate replanning in case of failure. RAPs can also change the current task-list, so that tasks can be decomposed into sub-tasks.

3.4.1 ALFA and ATLANTIS

[Gat92] describes ATLANTIS, an architecture for mobile robot control. This architecture has three components: control, sequencing, and deliberation. The control layer is designed as a set of circuit-like functions using Gat's language for circuits, ALPHA [Gat91]. The sequencing component is based on Jim Firby's RAP system. The deliberation layer advises the sequencing layer for more complicated problems. The deliberator is out of the control loop, so its time consuming pondering has no influence on the latency time of the whole architecture.

3.4.2 3T

3T, [BK] and [BK96], is very similar to Gat's architecture, its 3 layers are called the skill layer, the sequencing layer, and the planning layer respectively. Whereas in ATLANTIS these layers were known as the controller, the sequencer, and the deliberator. The difference between 3T and ATLANTIS is the sequencing layer. 3T uses the plain RAP system whereas ATLANTIS incorporates a more flexible and expanded version of this.

3.5 Conclusion

All systems presented in this section have evolved into 3 layered architectures. This 3 layer model incorporates a controller, handling tight control loops between sensors and actuators, a sequencer, which selects primitive behaviours, and a planner which provides the sequencer with goals. Nevertheless they have quite some disparities which made some of them difficult or even not applicable.

Like the previous architectures, the software architecture of the Pandora system is also a 3 layered architecture. The driving modes module function as the planning layer, the vehicle front end provides the control layer and the sequencing layer matches the functionality of the Configuration Selection Module (CSM). Likewise the sequencing components of these systems are candidate for the role of CSM.

The SubSumption architecture could be used for implementing the CSM, although this would be very cumbersome. The SubSumption network would need virtual sensors for all possible observations and a lot of "wiring" to implement the negative feedback in order to mutual exclude the occurrence of multiple selected configurations. Therefore the SubSump-

tion approached was not further considered for usage. For the DAMN architecture, the same arguments can be given as for the SubSumption architecture. The main difference between the DAMN and the SubSumption architecture is the way commands are merged. The SubSumption architecture suppresses the lower priority command whereas DAMN uses a weighted arbitration mechanism. Therefore DAMN could be applied although cumbersome.

Although the sound fundamentals of the REX, GAPP and RULER triplet are very attractive, the framework does not exploit opportunism. All state transitions need to be specified, the system is therefore not able to handle an unstructured environment. The RAP system on the other hand is based on opportunism, it does not necessarily find the most optimal solution, but it will make its decisions in an ad hoc manner. If for a particular goal no matching action is found immediately, the system is allowed to try different action templates. These actions could create new opportunities, which might lead in turn towards the goal. Therefore, the Configuration Selection Module was modeled after the RAP system.

Chapter 4

A Configuration Selection System

The Situation Driven Execution paradigm makes its decisions based on the current situation. A configuration selection mechanism will be presented that uses this principle to select configurations. This mechanism makes no assumptions about its application domain. This Situation Driven Selection paradigm is described in section 4.1. From this paradigm, a specification language has been derived. This language is described in section 4.2. The execution engine that the CSM uses is described in section 4.3. This engine was inspired by the STRIPS system ([Nil80]), one of the earliest planning systems that could deal with compound goals. Section 4.4 explains how the CSM interfaces with the rest of the system. Chapter 5 will show how this system was used to control a robot and gives also the results of the completed experiments.

4.1 Situation Driven Execution

The Situation Driven Execution paradigm cannot be expected to solve complex problems or generate optimal action sequences. These activities require looking into the future to see how every action affects the rest of the "plan". The assumption is that a traditional type of planner has sorted out many of these issues and provides the system with the most promising "sketchy plan". This plan lacks all details which can not be known until the robot actually starts to muddle towards the goal. In our case this "plan" is as simple as a desire to move in a specific direction. Execution of such a sketchy plan assumes that there is a prescribed method for carrying out tasks based on perception of the current situation. A method can be any sequence of actions that will accomplish the task in the given situation.

So, execution consists of choosing a task to work on, assessing the current sensed situation, choosing an appropriate method and carrying it out.

This task is a goal that is to be achieved, as before in our case such a task is just a desired heading which was received from a higher planning level. A stack is used to administrate these goals. At execution the goal on top is used as an index to identify possible methods that could lead to accomplishment of the goal. A method is only selected when it is relevant to the current situation, this could be any kind of expression including results of previous methods. If the constraints from more than one method are satisfied, the method with the highest priority (the one that is encountered first) is selected. A method (action sequence) is a prescription for changing the world situation so that a given goal becomes satisfied. There may be many different methods for reaching the same goal, each of them applicable in different situation or preferred in different priority. This diversity allows the system to respond and adapt to changing situations. All methods are known to the system in advance, so the system must muddle trough with what it knows or give up.

Robust behaviour can be achieved as long as every goal has one or more general methods for satisfying it in any situation. It is important to bear in mind that successful execution of a certain method does not necessarily mean that this resulted in the expected change of the current situation. A goal may therefore only be considered satisfied when this change of situation is observed, a separate method can be used to remove this goal from the execution stack. This strategy allows the premature termination of a method which makes it possible to introduce a fault recover mechanism. For all methods a time window can be specified in which this sequence is relevant. If this window expires, the sequence is aborted, thereby leaving the current situation as is. The situation is reassessed by the system and execution is resumed.

When for example such a system would use the following methods.

Goal	Constraint expression	Action Sequence	Method no.
move	mapEval = MapIsClear	move(), goalReached()	1
	default	newGoal(reconfigure), newGoal(observe)	2
observe	mapEval = SmallObstacleSuspected	setConfiguration(ObserveSmallObstacle2), goalReached()	3
	default	requestOperatorAssistance(), goalReached()	4
reconfigure	mapEval = SmallFlatObstacle	setConfiguration(SmallFlatObstacle4), goalReached()	5
	default	newGoal(observe)	6
default	default	fail()	7
idle	default	newGoal(move)	8

The system could receive an external move goal from a higher level or if no goal is received bootstrap itself with the idle statement (method 8). On reception of a move goal, the system has two options. It will therefore would assess its situation and if the map is clear (method 1) it will move straight ahead, or otherwise decompose the problem by pushing subgoals on the stack (method 2). Execution continues, but the top of the stack contains now the observe goal. The relevant methods in this case are method 3 and 4. If the map evaluator suspects a small obstacle, is suspected, an observation posture is requested in order to yield more data about the obstacle (method 3). If on the other hand the evaluator was not able to determine how to handle the encountered obstacle, the system would rely on the operator to handle the unknown situation (method 4). Method 5 is applied afterwards to reduce the remaining reconfigure goal, where after the original move is reduced by method 1.

4.2 Language description

Based on the Situation Driven Execution paradigm a specification language was developed. This language is used to specify the action templates for the Configuration Selection Module. When we consider the example from the previous section we can easily identify the key entities; goals, constraints and actions. In the following section we will show how the specification language is build around these entities and how some fault tolerance is added. The syntax diagrams for this language can be found in the appendix.

Goals

A goal is represented by an identifier and may have optional arguments. The arguments are provided to allow parametrized goals. The parameters provided when a goal is pushed

on the stack are available within the scope of the associated templates. These arguments are especially useful for providing subgoals with data. In our example there is one goal that might be provided by an external planner: move, the observation and reconfigure goals are internal subgoals. Goals can be stacked by an external high level planner or can be stacked as a result of an internal task decomposition. Section 4.3 describes in more detail how external goals are retrieved.

Constraints

The perception of the situation is expressed in constraints, these are used to select a method for execution. These constraints are not restricted to observations only, they may also contain parameters or any other expression. Call-back functions are provided for retrieving (sensor) data representing the current situation. Section 5.1.1 shows how such a call-back function can be used to extract high level data from a perception system such as a laser range finder. Additional internal state or global parameters can be monitored if variables are used within these constraints. The risk level is such a global parameter in the Pandora system.

Actions

Once a method is selected, its sequence of actions are executed. These actions may manipulate variables, stack new goals, remove the current goal or call action call-back functions. These call-back functions interact normally with the environment. In our case these functions interface to the robot hardware. The example shows two call-back functions controlling the robot: move() and setConfiguration(), also shown is the operator to remove the current goal: goalReached() and the operator to stack a new (sub)goal: newGoal().

4.2.1 Template description

The templates are composed of the previous mentioned components are listed per goal. A single goal can have one or more methods and may have an optional default method. There is also an idle template which will be selected if the goal stack is empty and no new goals are provided. The mandatory default template is selected when no method was found for the current goal.

Methods

A priority list of methods for a goal contains at least a single method. If the goal corresponding to these methods is considered to be volatile, the goal is stacked with a timeout value. The execution engine must purge the stack up to and including the expired goal when this value is exceeded. This feature is added to unstuck the system when it becomes stuck in a loop.

As said before, every single method has an expression which is evaluated during execution in order to determine if this method is to be selected for execution. When this expression holds, the corresponding actions are executed. An asynchronous timeout interval could be specified for these actions.

A default method can be specified for a particular goal. This default method will be tried if no other method for this goal matches, but before the mandatory global default method.

The following fragment shows the templates for our example in the proposed syntax.

```
TEMPLATES
move  {
         mapEval == MapIsClear :{
                                        move();
                                        GOAL_REACHED;
                                };
         DEFAULT :              {
                                        NEW_GOAL(reconfigure);
                                        NEW_GOAL(observe);
                                };
      };
observe  {
           mapEval == SmallObstacleSuspected :{
                                                        setConfiguration(ObserveSmallObstacle2);
                                                        GOAL_REACHED;
                                                };
           DEFAULT :                            {
                                                        requestOperatorAssistance();
                                                        GOAL_REACHED;
                                                };
         };
reconfigure  {
                mapEval == SmallFlatObstacle :{
                                                        setConfiguration(SmallFlatObstacle4);
                                                        GOAL_REACHED;
                                                };
                DEFAULT :                       {
                                                        NEW_GOAL(observe);
                                                };
             };
IDLE: {
         DEFAULT: {
```

```
                    NEW_GOAL(move());
                };
    };
DEFAULT: {
          DEFAULT: {
                    fail();
                };
        };
END
```

4.2.2 Fault recovery

We mentioned before that action sequences could be guarded with a timeout. This feature allows us to recover if this sequence can not be completed because of an error. The following fragment shows how the original setConfiguration() command is guarded with a timeout for 20 seconds. In combination with a variable which is set within the guarded section, fault detection can be done.

```
reconfigure  {
               lock == 1                     :              {
                                                               setConfiguration(SmallFlatObstacle1);
                                                               lock = 0;
                                                               GOAL_REACHED;
                                                            };
               mapEval == SmallFlatObstacle : TIMEOUT(2.0E4) {
                                                               lock = 1;
                                                               setConfiguration(SmallFlatObstacle4);
                                                               lock = 0;
                                                               GOAL_REACHED;
                                                            };
               DEFAULT :                                    {
                                                               NEW_GOAL(observe);
                                                            };
            };
```

The variable lock is initially zero, so the guarded sequence is executed and lock is set to 1. Because of an error, setConfiguration(SmallFlatObstacle4) does not return. After 20 seconds, this sequence is aborted and because lock is set to 1, an alternative configuration is tried.

In addition, the NEW_GOAL operator can be given an extra timeout argument. This timeout allows the execution engine to purge the stack up to and including the corresponding goal. More opportunistic goals can now be abandoned after a certain period of time so that the robot will not pursue a goal that can not be achieved. If for example the goal stack would contain the goal returnToBase and on top of that the goal findPerson which was stacked with an expire-time of 20 minutes. The robot could work on this goal for 20 minutes. If after

twenty minutes, this goal is still not reached, the stack is purged up to the returnToBase base goal. The robot will now abandon its previous mission and return to base.

4.3 Execution engine

The Configuration Selection System pursues the goal on top of the goal stack. The methods as given in the specification are used by the system to reduce the top goal. New goals from the planner can only be received by the system if the goal stack is empty. As a safety measure, goals on the stack can have an expire time. If this time expires the stack will be purged up to and including the expired goal. This assures that the CSM will respond to the planner regardless of the feasibility of the pursued goal. The main loop as described is depicted in figure 4-1.

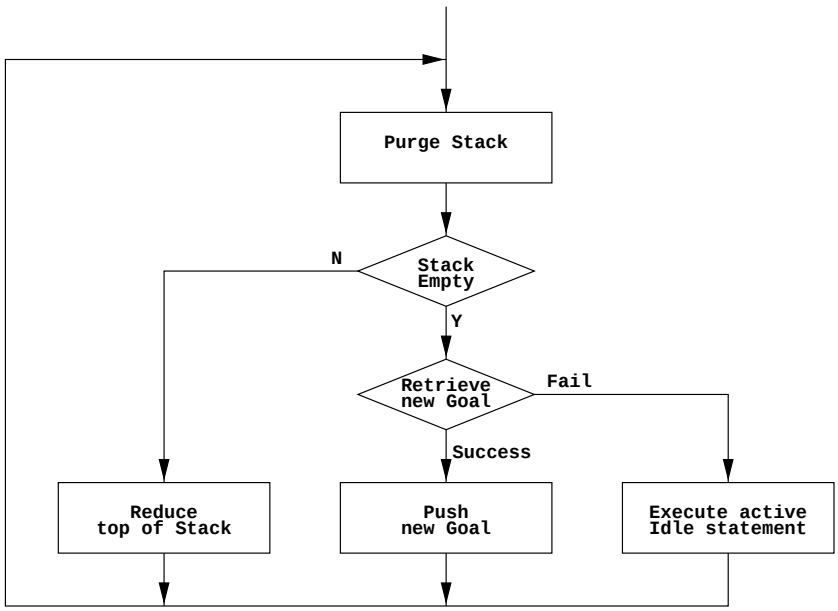


Figure 4-1: The Configuration Selection Module's main loop.

4.4 Interface

For the Configuration Selection Module, support routines from the QNX real-time operating system were used to keep track of the expire time and to communicate with the other modules in the system. The timeout as used to safeguard an action sequence is constructed

around an asynchronous signal handler, so the guarded action sequence must be specified in such a way that no harm is done when this sequence is aborted regardless of its execution boundary. New goals are received by the CSM on demand. When the goal stack is empty, a signal is sent to the planning layer in order to retrieve a new goal. The CSM will not wait indefinitely until a new goal is received, but will timeout and execute the idle statements. The actions as executed by the CSM can be terminated asynchronously by a timeout or by the control layer, eg. the control layer detects that the motor current threshold is exceeded and aborts the action. There is no direct feedback for such a failure. This imposes that the input data as used in the expressions to select an action sequence for execution must be coherent with the actual state of the world. So in the case that an action sequence is aborted, the state change of the world (and robot) is used in the next cycle to select an action sequence which recovers from the aborted sequence.

4.5 Implementation

Based on the described grammar, a compiler was build with the LEX and YACC tools. This compiler generates the CSM implementation in C code from the specification file. It also generates a header file from the declarations as found in the specification file. These declarations include the enumeration types, goal, observations, actions and variables. This header file also contains the public interface to the call-back and interface functions. These two files can then be compiled with a regular ANSI-C compiler into an executable.

4.6 Summary

The Configuration Selection Module as described in the previous sections is generated from a specification. In this specification, the action sequences are ordered per goal. The action sequence with the highest priority is scheduled for execution if and only if the corresponding selection expression is valid. Default action sequences and timeouts have been incorporated to add fault tolerance. The goals are administrated by a stack. The goal on top of the stack will be reduced according to the methods given in the specification. If the stack is empty, the CSM requests a new goal from the planner. If no new goal is available the CSM will schedule the highest priority action statements with a valid expression as specified in the idle declaration. In order to recover from failure, the input as fed into the CSM must be

coherent with the actual world state so that a failure can be detected and an appropriate method can be selected. This mechanism allows the control layer to abort a command without any need for direct acknowledgement.

Chapter 5

Results

Experiments were conducted with the Pandora simulator and with the Nomadic Scout mobile robot in order to validate the proposed architecture. The Scout was used to evaluate the dynamic behaviour of the system, whereas the simulator was used to evaluate its applicability.

5.1 Pandora Simulator

Before getting into the detailed description of the results, it is important to describe the experimental setup that was used for validating the proposed architecture. The following section describes therefore the features of the simulator. Also included are the results of the experimental perception system after which the simulator's map evaluator was build. This section is concluded with the results from the experiments done with this simulator.

5.1.1 Simulation model

The simulator is able to address the following functions: Motion of the robot on surfaces represented by polygons can be simulated. The speed of the robot can be adjusted dynamically. The configuration of the robot can be changed by rotating the articulated treads to the appropriate angles. Measurements from the laser line sensor can be simulated with a realistic model of measurement error. Specifically, the measurement error is controlled by the amount of error in detecting the line in the image, measured in pixels. The frequency of measurement can be adjusted arbitrarily. In the experiments, the pixel noise was set to 1 and a 50ms sampling time was used. The accumulation of range data from the laser into a

local map is also simulated, as well as the scrolling of the local map as the vehicle advances. Processing of the local map to evaluate navigability, and compute height and slopes of potential hazards is also simulated. The map evaluator classifies the data as found in the local map and returns entities such as "no-obstacle", "flat-obstacle", "tall-obstacle", "stairs" etc. If the evaluator lacks sufficient data to classify the obstacle, it returns something like "flat-obstacle-suspected". It is then up to the CSM to schedule an observation posture in order to gather more data so that the evaluator can then confirm this observation. An internal, simulated time is maintained so that the order and frequency of occurrence of events over time, e.g., range measurements, configuration changes, etc. is correctly simulated. Also, the computation time of the various components and their synchronization over time is simulated. Because the simulator was intended only for validating some of the critical components of the software architecture, it does not include a full three-dimensional model of the robot and its kinematics. Also, it uses a simplified model of physical interactions, such as friction.

The laser range finder and the map evaluator in the simulator are based on the results from experiments with an experimental laser range finder. Some of the results from these experiments are presented here to show that the simulated model is valid. We refer to the Pandora design document [RI98] for a more detailed description and the underlying fundamentals. Figure 5-1 shows a sequence of images from a staircase which was scanned by the laser range finder with a 10cm distance interval.

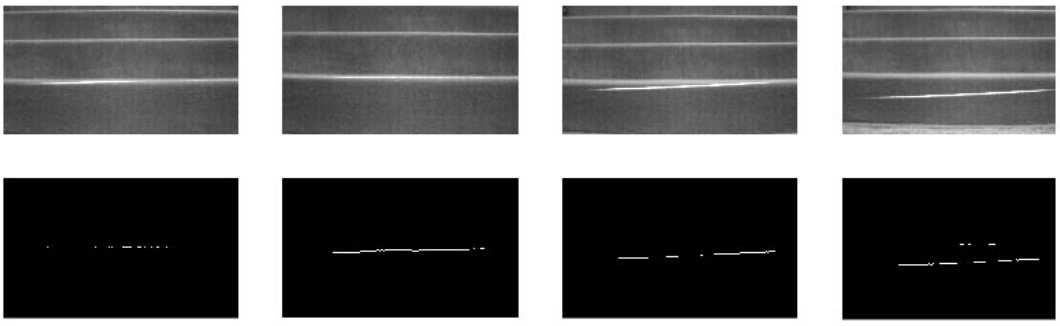


Figure 5-1: Laser data on outdoor sequence of images of stairs; (upper) raw video images; (lower) filtered images for stripe detection.

The line is detected in the filtered images, the position of the line in the image is then used to calculate the range. The result of this operation is plotted in figure 5-2. The map evaluator extracts a feature curve from this data by plotting the elevation of the range data for the direction of travel. Figure 5-3 is included to give an impression of the resolution and accuracy of the system,

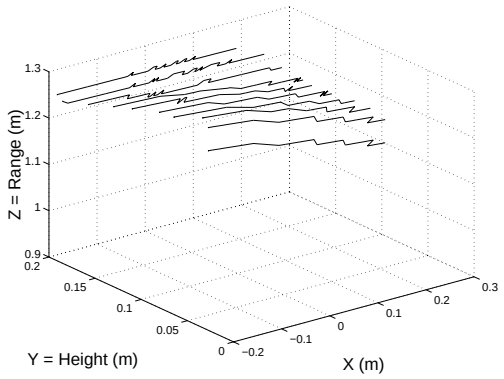


Figure 5-2: Three-dimensional view of the data from Figure 5-1 scans were obtained by moving the sensor at intervals of 0.1m from 0.9m to 2.0m from the steps; the figure shows the 3-D location of the points measured on a 0.2m step.

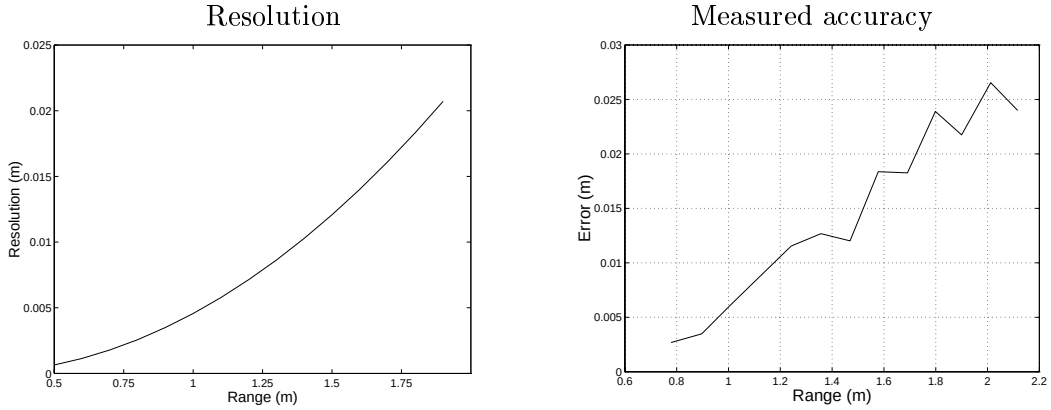


Figure 5-3: Laser range finder resolution and accuracy.

Based on the results presented in this section, we can conclude that we can safely use the Pandora simulator to test our architecture. The model as used by the simulator incorporates a model for the locomotion system, obstacles, surfaces and an accurate sensor model which was modeled after the results of data from experiments with a real laser range finder.

5.1.2 Simulation results

The Configuration Selection module was integrated with the simulator. The simulated robot can only move forward in the x direction. As a consequence, there is no high level planner available and the CSM will therefore only receive a move forward or stop command in this simulation. The CSM used during the experiments was generated from the following specification:

Goal	Constraint expression	Action Sequence	Method no.
move	mapEval = MapIsClear	move(), goalReached()	1
	default	newGoal(reconfigure), newGoal(observe)	2
observe	mapEval = SmallObstacleSuspected & riskLevel $\geq$ Normal	setConfiguration(ObserveSmallObstacle2), goalReached()	3
	mapEval = SmallObstacleSuspected & riskLevel $\leq$ Careful	setConfiguration(ObserveSmallObstacle1), goalReached()	4
	mapEval = TallObstacleSuspected	setConfiguration(ObserveTallObstacle), goalReached()	5
	default	goalReached()	6
reconfigure	mapEval = SmallFlatObstacle & riskLevel = Aggressive	setConfiguration(SmallFlatObstacle4), goalReached()	7
	mapEval = SmallFlatObstacle & riskLevel = Normal	setConfiguration(SmallFlatObstacle2), goalReached()	8
	mapEval = SmallFlatObstacle & riskLevel = Careful	setConfiguration(SmallFlatObstacle3), goalReached()	9
	mapEval = SmallFlatObstacle & riskLevel = VeryCareful	setConfiguration(SmallFlatObstacle1), goalReached()	10
	mapEval = TallFlatObstacle & riskLevel $\geq$ Normal	setConfiguration(TallFlatObstacle1), goalReached()	11
	mapEval = TallFlatObstacle & riskLevel $\leq$ Careful	setConfiguration(TallFlatObstacle2), goalReached()	12
	mapEval = LowSlopeObstacle & riskLevel = Aggressive	setConfiguration(LowSlopeObstacle3), goalReached()	13
	mapEval = LowSlopeObstacle & riskLevel = Normal	setConfiguration(LowSlopeObstacle1), goalReached()	14
	mapEval = LowSlopeObstacle & riskLevel $\leq$ Careful	setConfiguration(LowSlopeObstacle2), goalReached()	15
	default	newGoal(observe)	16
default	default	requestOperatorAssistance()	17

With the generated CSM, the simulated Pandora was able to traverse all the cases as specified in the design specification. Some of the traverses are shown in the figures figure 5-4 to figure 5-8. In 5-4, Pandora receives a move command. Because no obstacles are detected, this command is immediately reduced and a move command is posted (method 1). It receives another move, but now the map-evaluator returns "SmallObstacleSuspected", no match is made, so the default for this goal (method 2) is applied and two new sub goals are stacked. This new goal is then reduced by method 4 and Pandora raises to sweep the

field of perception over the obstacle. The map evaluator has now enough information, and returns a confirmation of a "SmallFlatObstacle". Method 10 matches now and Pandora reconfigures for attacking the obstacle. The obstacle is now traversed and a new goal is requested which is as before reduced by method 1. For figure 5-5 The beginning is the same, a move is received and the sequence of applying the methods 1,2,3,7,1 results is Pandora negotiating the same obstacle but now in an aggressive mode. In the next figure (5-6), Pandora is confronted again with a flat obstacle, but this time the obstacle is higher than can be negotiated by the configurations used before. Based on the observed obstacle and the careful risk level, the CSM selects now the methods 1,2,4,16,5,12. This sequence contains two observation postures. The result of the first observation is still ambiguous, therefore a different observation posture is selected which yields the required information. If a more aggressive mode is allowed, a different observation pose is used which yields immediately a confirmation for the appropriate obstacle class. Finally, in figure 5-8 the CSM applies the sequence 1,2,3,14,1 of methods in order to negotiate a staircase in "normal mode".

The Configuration Selection Module was not designed to be interfaced to the simulator, nevertheless it was just a matter of providing the necessary call-back functions for retrieving sensor data and sending motion commands. The methods as given previously were presented to the compiler in the required syntax and compiled and linked together with the simulator code. The resulting system was tested successfully in a variety of situations and parameters. We experienced that generating a controller from a compact specification is straightforward. Nevertheless, the system needs to be tested on more complicated systems to evaluate its true value.

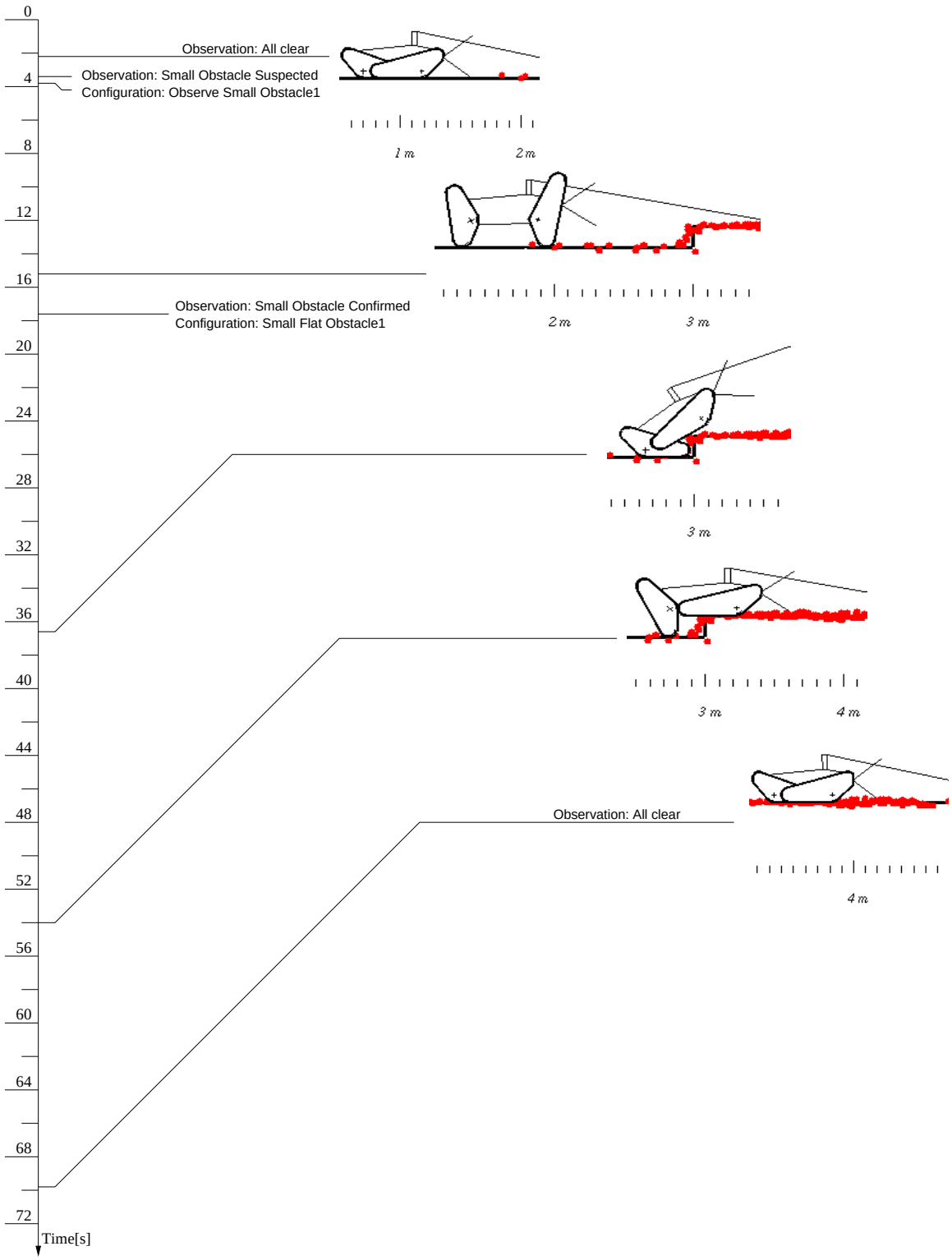


Figure 5-4: Pandora negotiating a small step in "very careful" mode.

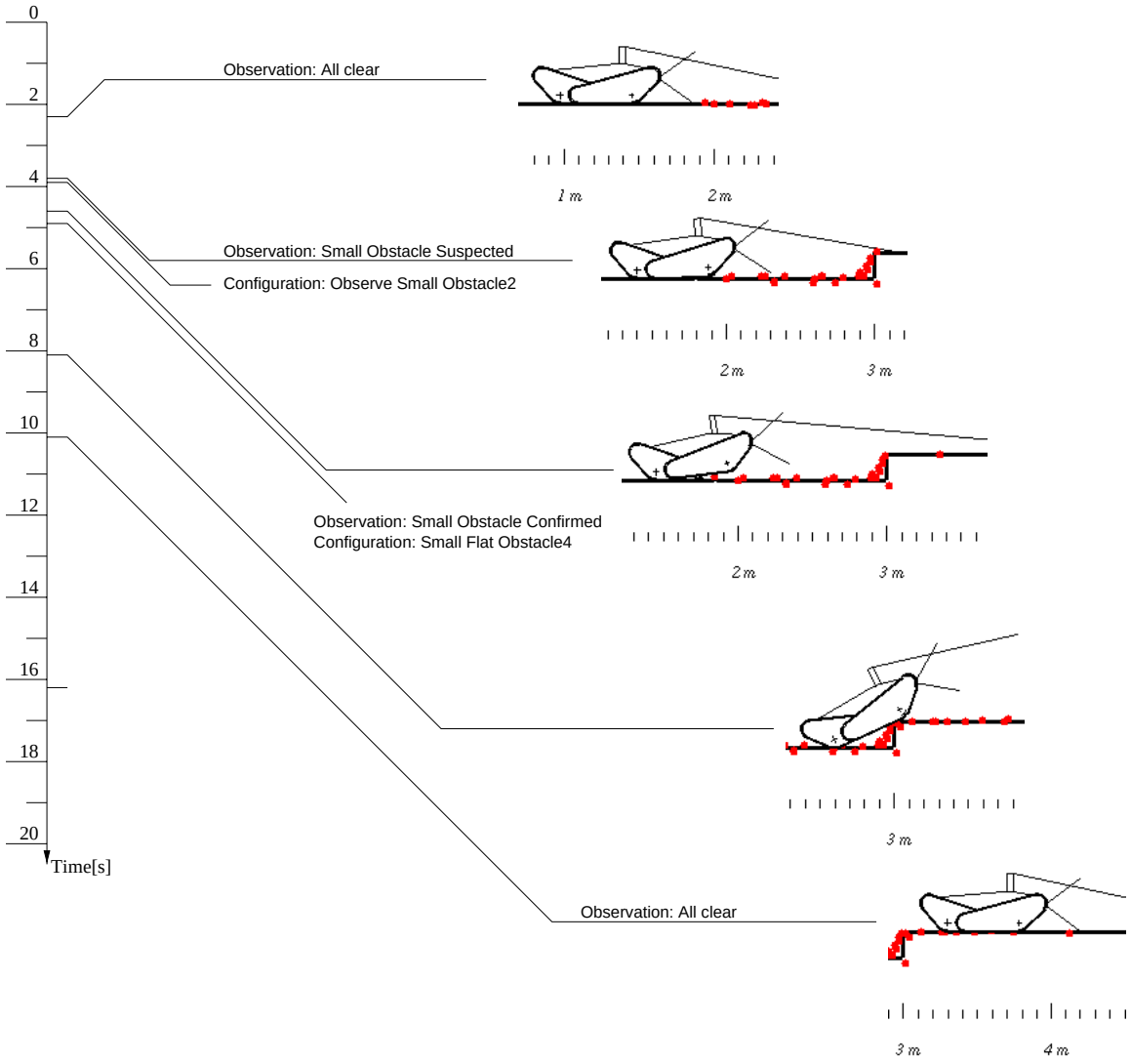


Figure 5-5: Pandora negotiating a small step in "aggressive" mode.

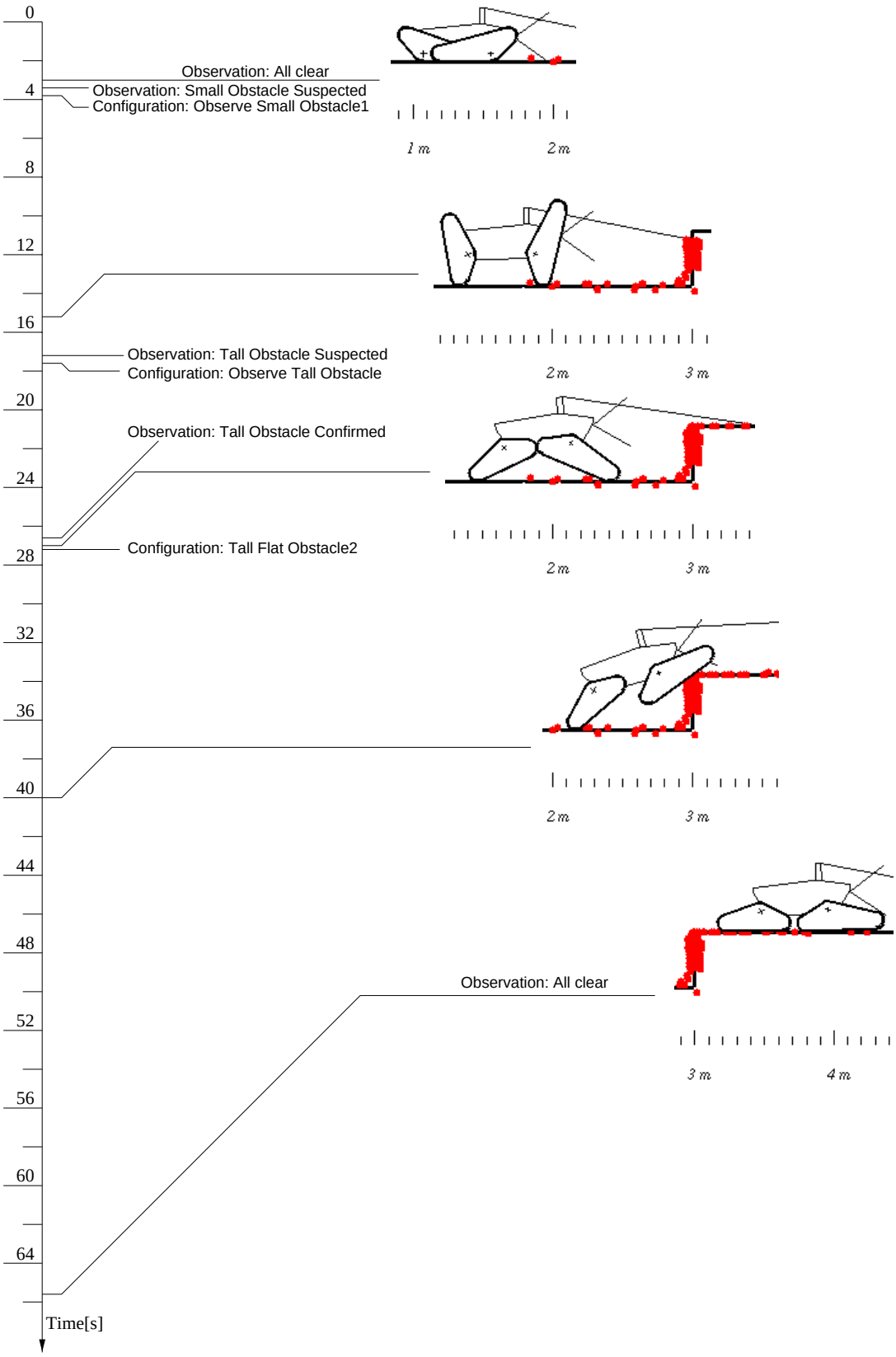


Figure 5-6: Pandora negotiating a tall step in "careful" mode.

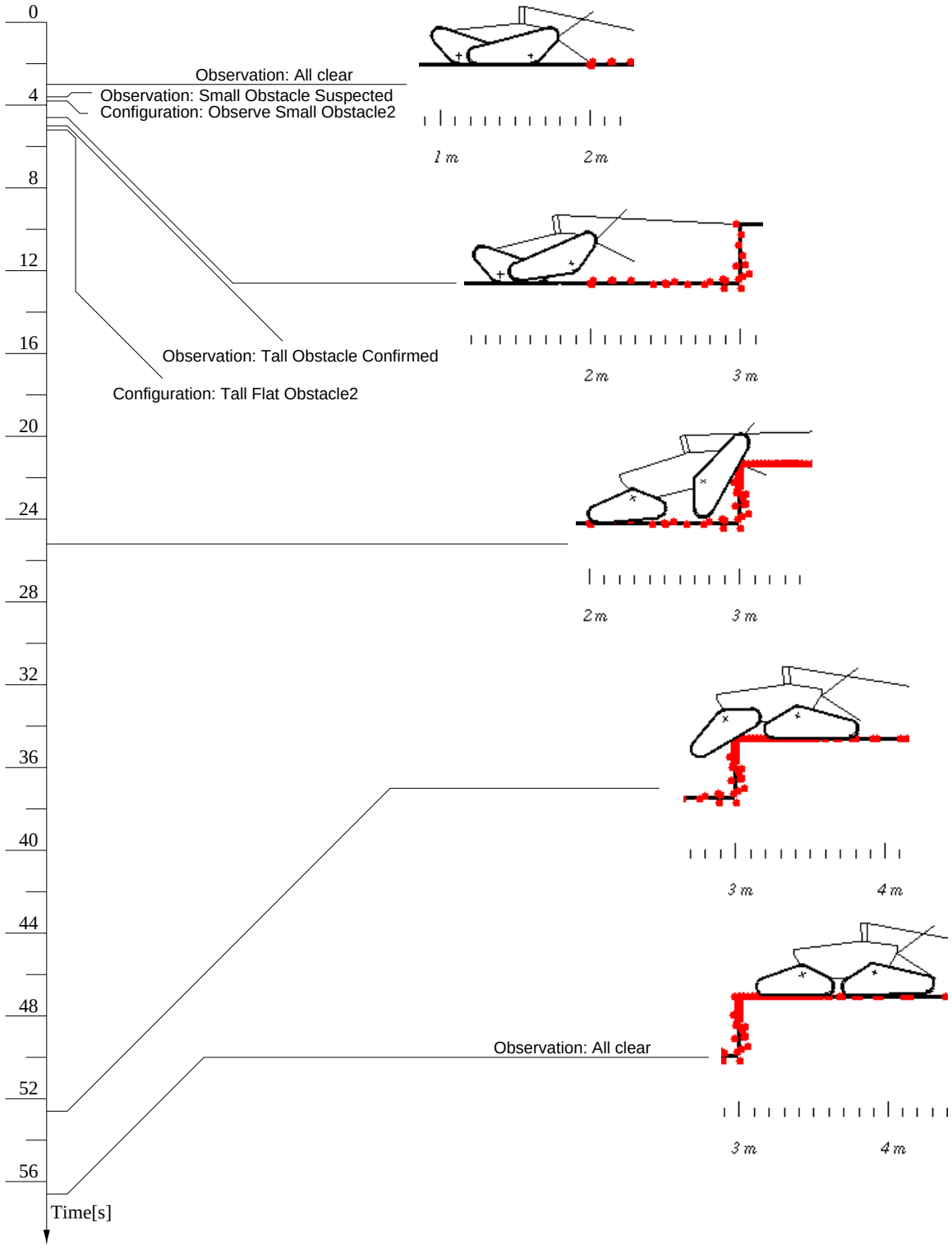


Figure 5-7: Pandora negotiating a tall step in "aggressive" mode.

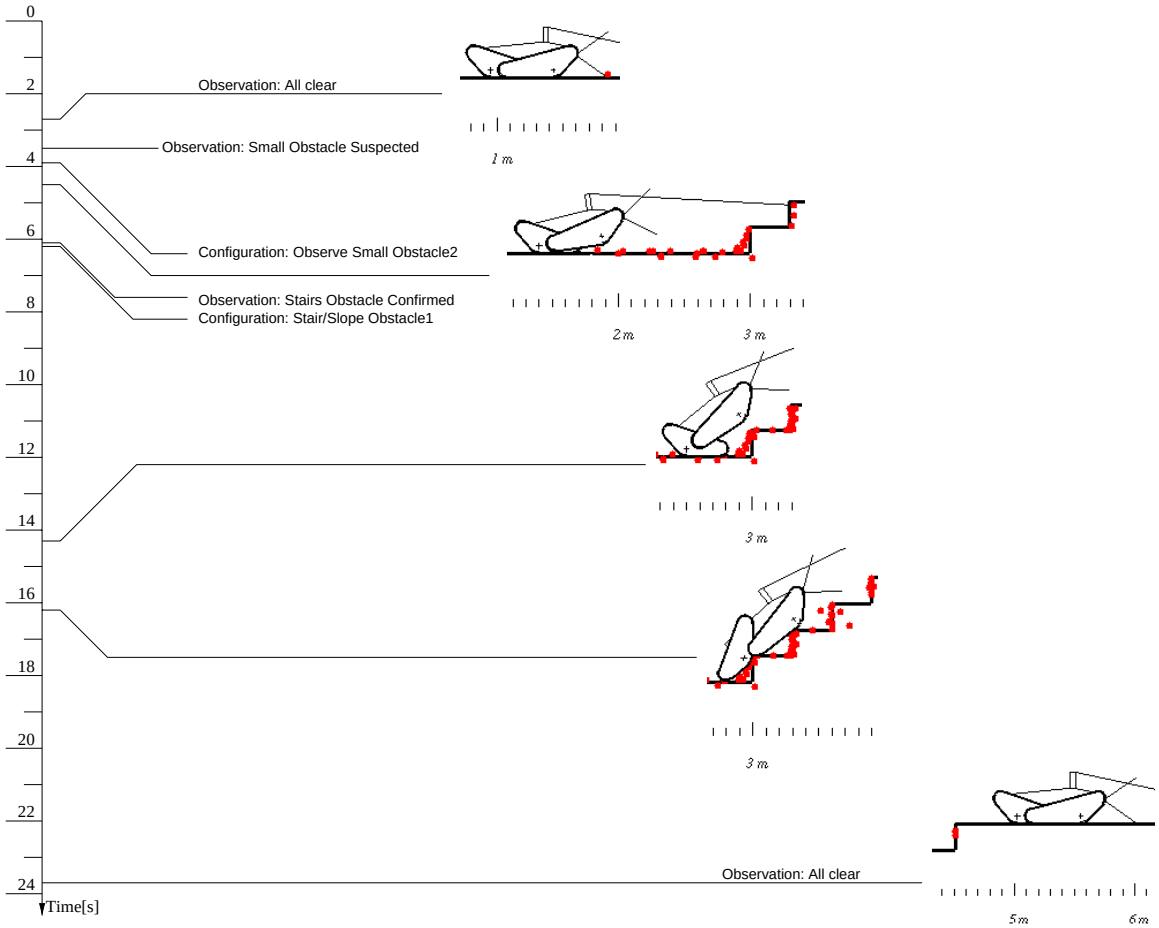


Figure 5-8: Pandora negotiating a staircase in “normal” mode.

5.2 Nomadic Scout

The Nomadic Scout as shown in figure 5-9 is obviously not configurable. The experiments conducted with the Scout were focused on testing the responsiveness, recovery and fault tolerance of the system. Instead of reconfiguring the robot, the Scout would be given a different orientation.

Figure 5-9: The Nomadic Scout.

To simulate a similar architecture as proposed for the Pandora system, the Scout control software architecture was build in 3 layers. The planning layer was simulated with a joystick interface, the sequencing layer was provided by the CSM and the control layer provided the interface to the Scout's sensors and actuators. Interfacing to the sensors was a bit cumbersome due to the fact that the system does not support the array type, this problem was circumvented by processing the sensor data outside the system and returning a more abstract value to the system. The system was not adapted to incorporate more versatile data types, because it is expected, and shown in the previous section, that the processed data of the perception system can be represented with an enumeration type. The layered architecture also incorporated a safeguarding system that would abort the current motion command if the robot was very close to an obstacle or if the motor current exceeded a predefined threshold. New goals were provided by the operator by means of the joystick. The templates used to control the robot contain methods that depending on the desired heading (goal) would generate the appropriate motion command. If an obstacle was present in that direction, a new goal would be stacked on top of the stack which would lead to reorienting the robot such that it was as close to the desired direction and not blocked by obstacles.

From the experiments done we could draw the following conclusions. The off-line compilation of methods into a stack machine generated a controller that was capable of presenting an output signal without any noticeable delay. When the Nomad was placed in a completely blocked environment, it was not able to find a way out, therefore the stack was purged after the goal's timeout expired and a default recovery method was then matched which asked for operator assistance. In order to test some more fault recovery behaviour, the sampling time was increased so that the robot could bump into an obstacle before detecting it. As expected, the control layer would abort such an operation before the CSM would notice the obstacle. In the next cycle, the CSM would detect this violation and would stack a reorientation goal. Failure of a method can not always be detected, for example the case of the timeout when the Nomad is trying to find a way out in a completely blocked environment is not detectable from sensor data only. This problem was solved by setting a variable before executing such a method and resetting it after successful completion. The state of this variable can then be used to trigger fault recover methods.

5.3 Summary

From the experiments we can conclude that a reactive controller architecture based on action templates can be used to select configurations for a reconfigurable robotic system. Although not tested with the real Pandora system, the proposed system seems to be capable to control such a reconfigurable robotic system in the real world. This assumption is based on the positive results from both the experiments done with the Nomadic Scout and the Pandora simulator. The experiments done with the Simulator validated the proposed concept, whereas the experiments with the Scout provided data about the system's dynamic behavior in the real world.

Chapter 6

Conclusion

This thesis describes a control architecture for reconfigurable robots. The system consists of a compiler and an execution engine, which provides the coupling between the low level control software and the high level planning software. A specification is used by the compiler to generate the robot dependent code. The execution engine uses a stack to administrate the goals. The methods as found in the specification are used to reduce the goals on the stack. These methods consist of a selection criteria and an action sequence. An action sequence is scheduled for execution if this selection criteria holds. The system makes no assumptions about the actual robot, so it could be applied to any reconfigurable robot. The methods are off-line evaluated and the generated stack machine is therefore fast, the price that is to be paid for this is that no methods can be added during execution, so it can not learn or adapt. The specification language is compact and not much code is needed to specify the configuration selection module. The generated C-code implementing the system can be integrated by using the generated header-files. The syntax of these interface functions is very strict, which can be somewhat cumbersome.

This framework was tested with the Nomadic Scout robot and a 2D simulation of a reconfigurable robot. The complementary result from these experiments give a strong indication that the system will be capable to cope with a reconfigurable robotic system. In order to validate the system for its true purpose, it needs testing on a reconfigurable robot. Validation on other reconfigurable robots would be necessary to show show that the system is generic. The research in bi-pedal walking robots at the Carnegie Mellon University will pursue these goals.

Appendix A

Syntax diagrams

The language as depicted in the following syntax diagrams consists of two sections: the declaration and the template definition section. The following sections describe these sections in some more detail.

A.1 Declaration section

This section contains prototypes for the goals, observations and actions. It also contains an enumeration type declaration and variable declaration. These prototypes are used for type checking and the generation of the header files.

Enumeration type definitions

The enumeration type declaration is provided to allow the usage of symbolic values for the observation and action parameters. In our case, we use the enumeration type to label the results from the map evaluator and the robot commands.

Goal prototypes

A goal declaration specifies the identifier token and has an optional argument declaration. The argument list is provided to allow parametrized goals. The parameters provided when a goal is pushed on the stack are available within the scope of the associated templates. These arguments are especially useful for providing subgoals with data.

Observations and Action prototypes

These declarations are prototypes for the user defined call-back functions. The observation functions are used to retrieve (sensor) data, whereas the action functions are primarily used to change the current situation (command the robot).

A.2 Template definition section

The action templates are listed per goal. A single goal can have one or more methods and may have an optional default method. There is also an idle template which will be selected if the goal stack is empty and no new goals are provided. The mandatory default template is selected when no method was found for the current goal.

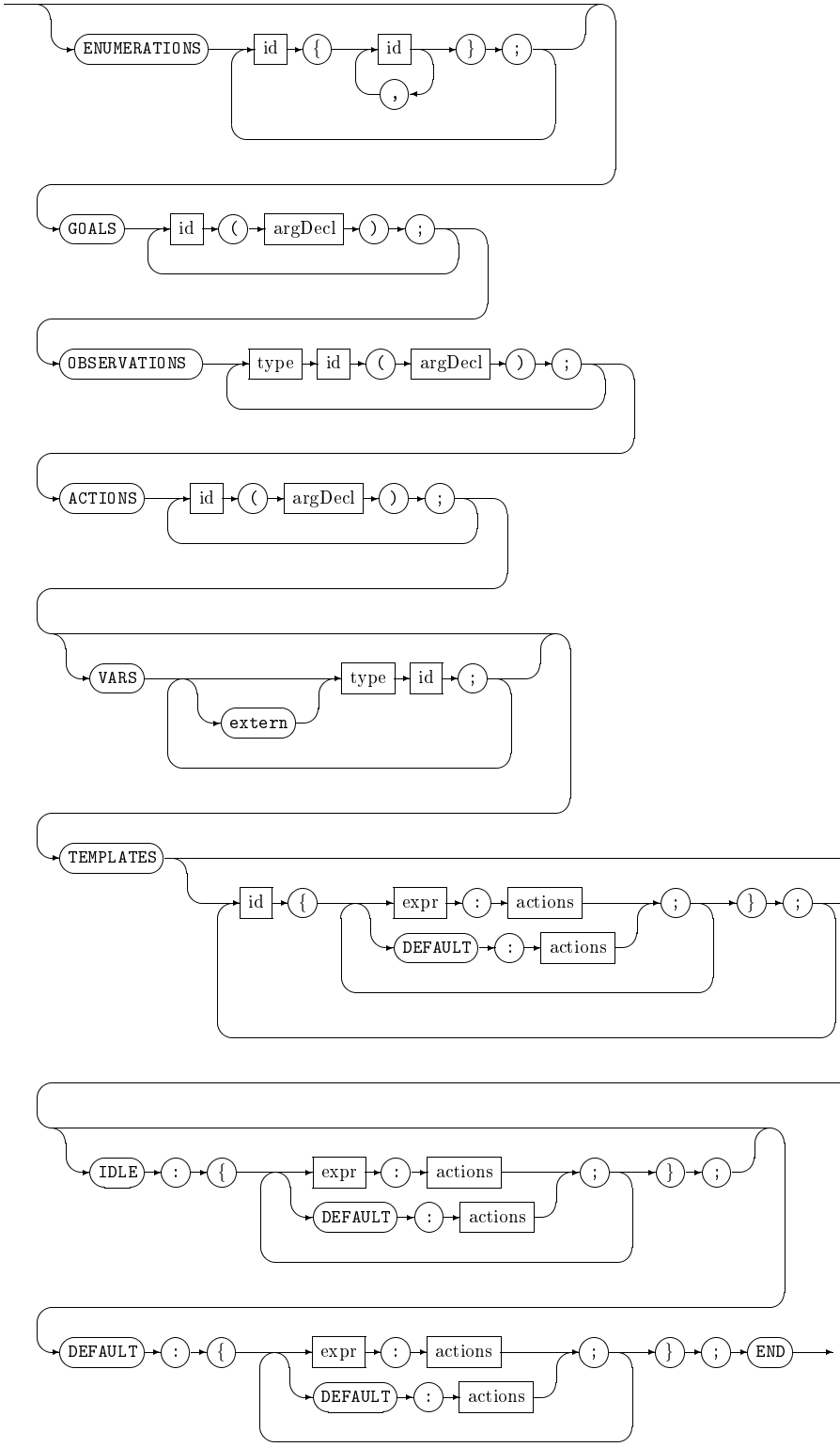
Goal related Methods

A priority list of methods for a goal must contain at least a single method. If the goal corresponding to these methods is considered to be volatile, the goal is stacked with a timeout value. The execution engine will purge the stack up to and including the expired goal when this value is exceeded. This feature is added to unstuck the system when it becomes stuck in a loop.

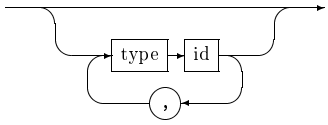
Every single method has an expression which is evaluated during execution in order to determine if this method is to be selected for execution. When this expression holds, the corresponding actions are executed. As mentioned before an asynchronous timeout interval could be specified for these actions. An action sequence could also push new (sub)-goals onto the stack with the NEW_GOAL operator or remove the current goal with the GOAL_REACHED operator.

A default method can be specified for a particular goal. This default method will be tried if no other method for this goal matches, but before the mandatory global default method.

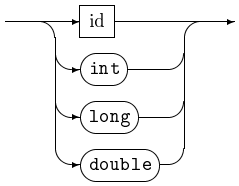
templateSystem



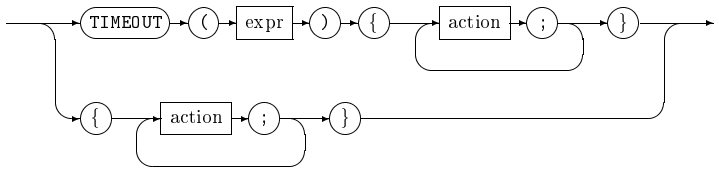
argDecl



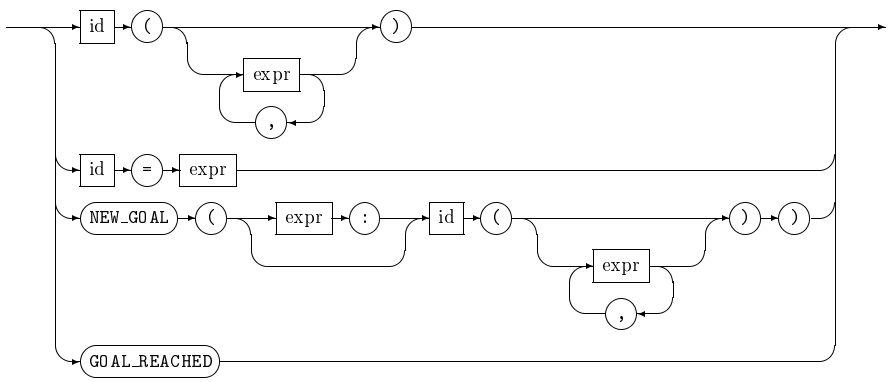
type



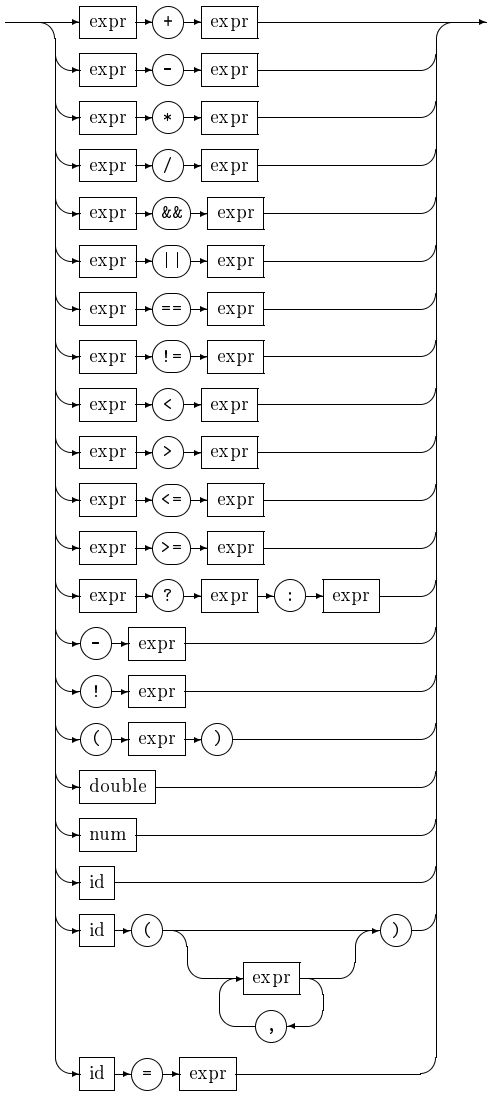
actions



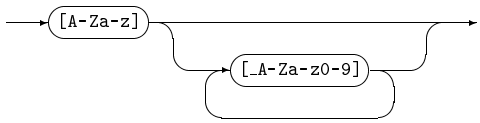
action



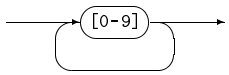
expr


expr + expr
expr - expr
expr * expr
expr / expr
expr && expr
expr || expr
expr == expr
expr != expr
expr < expr
expr > expr
expr <= expr
expr >= expr
expr ? expr : expr
- expr
! expr
(expr)
double
num
id
id ( ) expr ,
id = expr

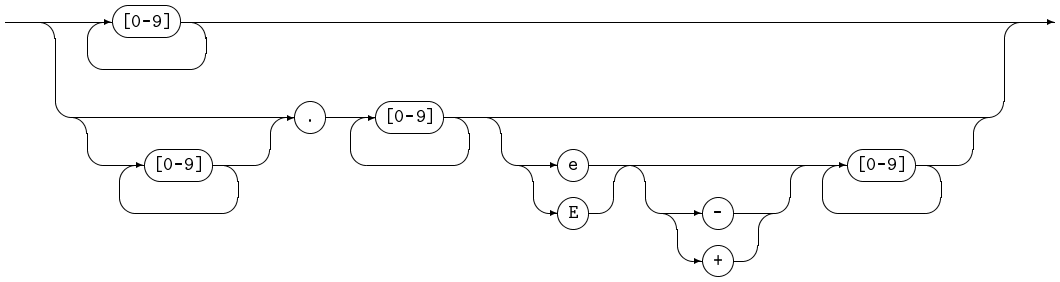
id


[A-Za-z]
[_A-Za-z0-9]

num


[0-9]

double



Bibliography

[BK] R. Peter Bonasso and David Kortenkamp. Characterizing an architecture for intelligent, reactive agents. Metrica Inc. Robotics and Automation Group NASA Johnson Space Center,1996.

[BK96] R. Peter Bonasso and David Kortenkamp. Using a layered control architecture to alleviate planning with incomplete information. In *National Conference on Artificial Intelligence (AAAI)*, 1996.

[Bro86] Rodney A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal on Robotics and Automation*, RA-2(1), 1986.

[Bro91] Rodney Brooks. Intelligence without representation. In *Artificial Intelligence*, volume 47, pages 139–160, 1991.

[Con89] Jonathan Connell. A colony architecture for an artificial creature. Technical Report 1151, Massachusetts Institute of Technology Artificial Intelligence Laboratory, 1989.

[Con92] Jonathan Connell. "sss: A hybrid architecture applied to robot navigation. In *IEEE Conference on Robotics and Automation (ICRA)*, 1992.

[Fir89] R. James Firby. *Adaptive Execution in Complex Dynamic Domains.* PhD thesis, Yale University, 1989.

[Gat91] Erann Gat. Alfa: A language for programming reactive robotic control systems. In *IEEE Conference on Robotics and Automation (ICRA)*, 1991.

[Gat92] Erann Gat. Integrating planning and reaction in a heterogeneous asynchronous architecture for controlling mobile robots. In *Tenth National Conference on Artificial Intelligence (AAAI)*, 1992.

[Gat94] Erann Gat. Behavior control for robotic exploration of planetary surfaces. In *IEEE Transactions on Robotics and Automation*, volume 10, 1994.

[Gat97] Erann Gat. On three-layer architecture. Jet Propulsion Laboratory, 1997.

[HF90] Steve Hanks and R. James Firby. Issues and architectures for planning and execution. In *DARPA Workshop on Innovative Approaches to Planning, Scheduling and Control*, pages 59–70, 1990.

[Kae87] Leslie Pack Kaelbling. Rex: A symbolic language for the design and parallel implementation of embedded systems. In *AIAA conference on Computers in Aerospace*, 1987.

[Kae88] Leslie Pack Kaelbling. Goals as parallel program specifications. In *National Conference on Artificial Intelligence (AAAI)*, 1988.

[Kae90] Leslie Pack Kaelbling. Specifying complex behavior for computer agents. In *DARPA Workshop on Innovative Approaches to Planning, Scheduling and Control*, pages 433–438, 1990.

[Lat91] Jean-Claude Latombe. *Robot Motion Planning.* Kluwer Academic Publishers, 1991.

[LMB92] John R. Levine, Tony Mason, and Doug Brown. *Lex & Yacc.* O'Reilly & Associates, 1992.

[McD90] Drew McDermott. Planning reactive behavior: A progress report. In *DARPA Workshop on Innovative Approaches to Planning, Scheduling and Control*, pages 450–458, 1990.

[McD91] D. McDermott. A reactive plan language. Technical Report 864, Yale University, Department of Computer Science, 1991.

[NG96] Illah R. Nourbakhsh and Michael R. Genesereth. Assumptive planning and execution: A simple, working robot architecture. *Autonomous Robots*, 3:49–67, 1996.

[Nil80] Nils J. Nilsson. *Principles of Artificial Intelligence*. Tioga, Palo Alto, 1980.

[Nom97] Nomadic Technologies inc., Mountain View, CA. *Language Reference Manual*, 1997.

[Pay86] David W. Payton. An architecture for reflexive autonomous vehicle control. In *Robotics Automation*, pages 1838–1845, 1986.

[RI98] The Robotics Institute. Pandora: A robotic system for operations in urban environments. Carnegie Mellon University, 1998.

[RK95] Stanley J. Rosenschein and Leslie Pack Kaelbling. A situated view of representation and control. *Artificial Intelligence*, 73, 1995.

[RP89] J. Kenneth Rosenblatt and David W. Payton. A fine-grained alternative to the subsumption architecture. In *AAAI Stanford Spring Symposium Series*, 1989.

[Sim90] Reid Simmons. An architecture for coordinating planning, sensing and action. In *DARPA Workshop on Innovative Approaches to Planning, Scheduling and Control*, pages 292–297, 1990.